A FORMAL MODEL FOR A GENERAL-PURPOSE COMPILER FOR SECURE MULTIPARTY COMPUTATIONS

by

Amy Marie Rathore

August 2022

A dissertation submitted to the
faculty of the Graduate School of
the University at Buffalo, The State University of New York
in partial fulfillment of the requirements for the
degree of

Doctor of Philosophy
Department of Computer Science and Engineering

Copyright by

Amy Marie Rathore

2022

All Rights Reserved

To my loving and supportive husband Aditya and our dog Ryker, always pushing me forward and providing me with moral support, and in loving memory of my Grandpa Bean, whose passion for learning and understanding has always inspired me.

Acknowledgments

I would like to express my gratitude to all who have supported me in the pursuit of my Ph.D. at the University at Buffalo. To the professors, students, and fellow Ph.D. candidates I have had the privilege of learning from and working with, thank you for helping make this such a rich and comprehensive experience.

I am forever grateful to my advisor, Dr. Lukasz Ziarek, without whom I would not be writing this thesis. His enthusiasm about programming languages and research inspired me to start on this journey, and his breadth and depth of knowledge, particularly in compilers and formal methods, has proved invaluable throughout this process. I aspire to be as knowledgable and as excellent of a mentor as he is.

Additionally, I would like to thank my committee member Dr. Marina Blanton as well as Dr. Marco Gaboardi, whom I have had the opportunity to work closely with throughout the course of my Ph.D., for all of their advice and insights into their specialities. The works of Dr. Blanton in the area of Secure Multiparty Computation helped to show a need and a starting point for this project, and her knowledge in this area has ensured we have upheld the standards of SMC in our model and implementation. Dr. Gaboardi's knowledge of formal methods has been invaluable in developing our formal models and proofs, with his insights helping in giving a forward direction in making improvements to each model.

I would also like to thank my committee member Dr. Bharat Jayaraman, who has provided me with valuable advice in and out of the classroom. Dr. Jayaraman's interest in software verification has helped me improve my knowledge and breadth of understanding of this area.

Finally, I would like to thank my parents Matthew and Anita, my sister Mindy, my Grandma Wanda, my friends Desir'ee and Amilyn, and especially my husband Aditya and dog Ryker, for their never-ending encouragement and support along this journey.

Abstract

Although Secure Multiparty Computation (SMC) has seen considerable development in recent years, its use is challenging, resulting in complex code which obscures whether the security properties or correctness guarantees hold in practice. For this reason, several works have investigated the use of formal methods to provide guarantees for SMC systems. However, these approaches have been applied mostly to domain specific languages (DSL), neglecting general-purpose approaches and the relation of the DSL to the underlying cryptographic implementation. In this paper, we consider a formal model for an SMC system for annotated C programs. We choose C due to its popularity in the cryptographic community and being the only general-purpose language for which SMC compilers exist. Our formalization supports all key features of C – including private-conditioned branching statements, mutable arrays (including out of bound array access), pointers to private data, etc. We use this formalization to characterize correctness and security properties of annotated C, with the latter being a form of non-interference on execution traces. We realize our formalism as an implementation in the PICCO SMC compiler, providing evaluation results on SMC programs written in C and extending PICCO with a DSL to allow further optimizations to improve performance.

Table of Contents

Dedication . . . . . **iii**

Acknowledgments . . . . . **iv**

Abstract . . . . . **v**

List of Tables . . . . . **viii**

List of Figures . . . . . **x**

1 Introduction **1**

2 Background and Literature Review **4**

3 Basic SMC^2 . . . . . **9**
3.1 Formal Semantics 9
3.1.1 SMC^2 Grammar 9
3.1.2 SMC^2 Memory Model 11
3.1.3 Vanilla C 17
3.1.4 Basic SMC^2 34
3.1.5 Vanilla C Algorithms 59
3.1.6 Basic SMC^2 Algorithms 74
3.2 Correctness 103
3.2.1 Erasure Function 108
3.2.2 Definitions 115
3.2.3 Lemmas 117
3.2.4 Proof of Correctness 131
3.3 Noninterference 242
3.3.1 Supporting Metatheory 245
3.3.2 Proof of Noninterference 250

4 Location-tracking SMC^2 . . . . . **317**
4.1 Formal Semantics 319
4.1.1 Location-tracking SMC^2 321
4.1.2 Algorithms 341
4.2 Correctness 354
4.2.1 Supporting Metatheory 358
4.2.2 Proof of Correctness 363
4.3 Noninterference 486
4.3.1 Supporting Metatheory 488
4.3.2 Proof of Noninterference 490

5 Multiparty SMC^2 . . . 567
5.1 Formal Semantics . . . 568
5.1.1 Multiparty Vanilla C . . . 570
5.1.2 Multiparty SMC^2 . . . 582
5.1.3 Multiparty Vanilla C Algorithms . . . 612
5.1.4 Multiparty Protocols for Multiparty SMC^2 . . . 613
5.1.5 Multiparty SMC^2 Algorithms . . . 620
5.2 Correctness . . . 645
5.2.1 Definitions . . . 654
5.2.2 Lemmas . . . 658
5.2.3 Multiparty Computation Axioms . . . 678
5.2.4 Confluence . . . 679
5.2.5 Proof of Correctness . . . 681
5.3 Noninterference . . . 758
5.3.1 Supporting Metatheory . . . 762
5.3.2 Multiparty Computation Axioms . . . 771
5.3.3 Location Access Tracking Supporting Metatheory . . . 773
5.3.4 Proof of Noninterference . . . 778

6 Realization . . . 848
6.1 Implementing Multiparty SMC^2 in PICCO . . . 848
6.1.1 Conditional Code Block Tracking Implementation . . . 848
6.1.2 Multiparty SMC^2 Evaluation . . . 850
6.2 Domain Specific Language (DSL) . . . 874
6.2.1 Design . . . 874
6.2.2 Formalization . . . 876
6.2.3 Implementation . . . 889
6.2.4 Evaluation . . . 890

7 Conclusion . . . 908

Reference . . . 914

List of Tables

2.1 Language features supported in SMC compilers. 4

6.1 Average runtimes and standard deviation for local computation. 852
6.2 Average runtimes and standard deviation for distributed computation. 852
6.3 h_analysis - local PICCO 861
6.4 h_analysis - local SMC^2 861
6.5 LR-parser - local PICCO 862
6.6 LR-parser - local SMC^2 862
6.7 pay-gap - local PICCO 863
6.8 pay-gap - local SMC^2 863
6.9 private-branching - local PICCO 864
6.10 private-branching - local SMC^2 864
6.11 private-branching-mult - local PICCO 865
6.12 private-branching-mult - local SMC^2 865
6.13 private-branching-add - local PICCO 866
6.14 private-branching-add - local SMC^2 866
6.15 private-branching-reuse - local PICCO 867
6.16 private-branching-reuse - local SMC^2 867
6.17 h_analysis - distributed PICCO 868
6.18 h_analysis - distributed SMC^2 868
6.19 LR-parser - distributed PICCO 869
6.20 LR-parser - distributed SMC^2 869
6.21 private-branching - distributed PICCO 870
6.22 private-branching - distributed SMC^2 870
6.23 private-branching-mult - distributed PICCO 871
6.24 private-branching-mult - distributed SMC^2 871
6.25 private-branching-add - distributed PICCO 872
6.26 private-branching-add - distributed SMC^2 872
6.27 private-branching-reuse - distributed PICCO 873
6.28 private-branching-reuse - distributed SMC^2 873
6.29 Runtime statistics for each optimal variable sizing test program. 892
6.30 Test 1 - local SMC^2 896
6.31 Test 1 - local optimized 896
6.32 Test 1 - distributed SMC^2 897
6.33 Test 1 - distributed optimized 897
6.34 Test 2 - local SMC^2 898
6.35 Test 2 - local optimized 898
6.36 Test 2 - distributed SMC^2 899
6.37 Test 2 - distributed optimized 899
6.38 Test 3 - local SMC^2 900
6.39 Test 3 - local optimized 900
6.40 Test 3 - distributed SMC^2 901

6.41 Test 3 - distributed optimized 901
6.42 Test 4 - local SMC^2 902
6.43 Test 4 - local optimized 902
6.44 Test 4 - distributed SMC^2 903
6.45 Test 4 - distributed optimized 903
6.46 Test 5 - local SMC^2 904
6.47 Test 5 - local optimized 904
6.48 Test 5 - distributed SMC^2 905
6.49 Test 5 - distributed optimized 905
6.50 Test 6 - local SMC^2 906
6.51 Test 6 - local optimized 906
6.52 Test 6 - distributed SMC^2 907
6.53 Test 6 - distributed optimized 907

List of Figures

2.1 Securely calculating the gender pay gap for 100 organizations. 6

3.1 Combined Vanilla C/SMC^2 Grammar. The color red denotes terms specific to programs written in SMC^2, and the color blue denotes elements synthesized by the semantics. 10
3.2 Basic SMC^2 configuration: environment γ, memory σ, accumulator acc, and statement s. . . 11
3.3 Types of overshooting array accesses. 15
3.4 Examples of alignment between SMC^2 and Vanilla C in overshooting accesses by incrementing pointer `p` three times. 17
3.5 Vanilla C semantic rules for basic variable declarations, reading, and writing; loops; sequencing; binary operations; and incrementing. 19
3.6 Vanilla C semantic rules for functions, casting, memory management, and input / output. . . 22
3.7 Vanilla C semantic rules for branches and pointer declarations, reading, writing, and incrementing. 26
3.8 Vanilla C semantic rules for pointer dereference write and dereference read. 29
3.9 Vanilla C semantic rules for one-dimensional array declarations, reading, and writing. 31
3.10 Vanilla C semantic rules for managing an entire array. 33
3.11 Basic SMC^2 semantics for basic declarations, reading, writing, sequencing, loops, finding the size of a type, and finding the address of a variable. 35
3.12 Basic SMC^2 semantics for addition, subtraction, multiplication, and division. 37
3.13 Basic SMC^2 semantics for not equal to and equal to comparison operations. 39
3.14 Basic SMC^2 semantics for less than comparison operations. 40
3.15 **`if else`** branching on private data example (3.15a, 3.15b) matching to the public Basic SMC^2 (3.15d, 3.15e) and private Basic SMC^2 (3.15c) if else rules. Coloring in the rules highlight the corresponding code and rule execution. 40
3.16 Basic SMC^2 semantics for the pre-increment operator (`++x`). 42
3.17 Additional Basic SMC^2 semantics for the pre-increment operator (`++x`) on private pointers. . 43
3.18 Basic SMC^2 semantics for functions and casting. 44
3.19 Basic SMC^2 semantics for input / output 46
3.20 Basic SMC^2 memory management. 47
3.21 Basic SMC^2 pointer declaration, read, and write rules. 48
3.22 Basic SMC^2 pointer dereference write rules for the first level of indirection. 49
3.23 Basic SMC^2 semantic rules for pointer dereference writes at higher levels of indirection. . . 51
3.24 Basic SMC^2 semantic rules for pointer dereference read. 52
3.25 Basic SMC^2 semantic rules for array declarations and writing an entire array. 54
3.26 Basic SMC^2 semantic rules for reading from arrays. 55
3.27 Basic SMC^2 semantic rules for writing to an array. 56
3.28 Basic SMC^2 array reading out of bounds rules 57
3.29 Basic SMC^2 array writing out of bounds rules 58
3.30 Table of SMC^2 evaluation codes in $SmcC \setminus SmcCX$ and their congruent Vanilla C evaluation codes in $VanC \setminus VanCX$. 104
3.31 The Erasure function, broken down into various functionalities. 110
3.32 Erasure function over the environment and memory 111

4.1 Examples of the challenges of private-conditioned branching examples, with pointer challenges shown in 4.1a and array challenges shown in 4.1b. We show values in memory that change in the table to the left, and values for temporary variables that do not change in the table to the right. We indicate correct updates in memory in green, and problematic values in memory in red. 318
4.2 Location-tracking SMC^2 configuration with added location map Δ, local variable list $\overline{\chi}$, and branch identifier bid 319
4.3 **`if else`** branching on private data example (4.3a, 4.3b) matching to the Basic SMC^2 (4.3d), and Location-tracking SMC^2 (4.3e) rules. Coloring in the rules highlight the corresponding code and rule execution. The public **`if else`** rules (4.3f, 4.3g) are shown for reference. . . . 322
4.4 Location-tracking SMC^2 semantic rules for basic variable declarations, reading, writing, and loops. 324
4.5 Location-tracking SMC^2 semantic rules for input. 325
4.6 Location-tracking SMC^2 semantic rules for output. 326
4.7 Location-tracking SMC^2 semantic rules for the pre-increment operator. 327
4.8 Location-tracking SMC^2 semantics for addition and subtraction. 328
4.9 Location-tracking SMC^2 semantics for multiplication and division. 329
4.10 Location-tracking SMC^2 semantics for less than comparisons. 330
4.11 Location-tracking SMC^2 semantics for equal to comparisons. 331
4.12 Location-tracking SMC^2 semantics for not equal to comparisons. 332
4.13 Location-tracking SMC^2 semantic rules for sequencing and functions 333
4.14 Location-tracking SMC^2 semantic rules for memory allocation, deallocation, and casting. . . 334
4.15 Location-tracking SMC^2 semantic rules for array declarations and writing an entire array. . . 335
4.16 Location-tracking SMC^2 semantic rules for reading from an array at a public or private index and reading an entire array. 336
4.17 Location-tracking SMC^2 semantic rules for writing to an array at an index. 337
4.18 Location-tracking SMC^2 semantic rules for reading and writing out-of-bounds for arrays. . . 338
4.19 Location-tracking SMC^2 semantic rules for pointer declarations, reading, and writing. 339
4.20 Location-tracking SMC^2 semantic rules for pointer dereference write. 340
4.21 Location-tracking SMC^2 semantic rules for pointer dereference read. 341
4.22 The Erasure function from Location-tracking SMC^2 configurations to Vanilla C configurations. 357
4.23 Table of Location-tracking SMC^2-specific evaluation codes in $SmcCT$ and their congruent Vanilla C evaluation codes in $VanC$. These evaluation codes are in addition to the SMC^2 evaluation codes shown in Figure 3.30. 358

5.1 Multiparty SMC^2 configuration: party identifier p, environment γ, memory σ, location map Δ, accumulator acc, and statement s. 568
5.2 Multiparty Vanilla C semantics for multiparty evaluation of binary operations, if else statements, and pre-incrementing floats. 573
5.3 Multiparty Vanilla C semantics for multiparty evaluation of pointers, deallocation, and arrays. 574
5.4 Multiparty Vanilla C semantic rules for binary operations. 575
5.5 Additional Multiparty Vanilla C semantic rules within the scope of the grammar shown in Figure 3.1. 576
5.6 Multiparty Vanilla C semantic rules for branching, functions, and casting values. 577
5.7 Multiparty Vanilla C semantic rules for input and output, memory allocation and deallocation, and casting locations. 578
5.8 Additional Multiparty Vanilla C semantic rules for pointers. 579

5.9 Multiparty Vanilla C semantic rules for arrays. 580
5.10 Multiparty Vanilla C semantic rules for array out of bounds. 581
5.11 Multiparty SMC^2 semantics for reading from or writing to a private index of an array and binary operations involving private data. 584
5.12 Multiparty SMC^2 semantics for reading from or writing to a private index of an array and binary operations involving private data. 585
5.13 Multiparty SMC^2 semantics for private free with multiple locations and the pre-increment operation over private float values. 587
5.14 Multiparty SMC^2 semantic rules for dereference writing multiple location to a private pointer of a higher level of indirection. 588
5.15 Multiparty SMC^2 semantic rules for private pointer dereference read with multiple locations and dereference write with a public value. 594
5.16 Multiparty SMC^2 semantic rules for the multiparty execution of private-conditioned if else statements. 595
5.17 Multiparty SMC^2 semantic rules for public branches, loops, and sequencing. 596
5.18 Multiparty SMC^2 semantic rules for pointer declarations and writing. 597
5.19 Multiparty SMC^2 semantic rules for reading from a pointer and dereferencing a pointer at a single location. 598
5.20 Multiparty SMC^2 semantic rules for pointer dereference write. 599
5.21 Multiparty SMC^2 semantic rules for array declarations and reading from a public index. . . . 600
5.22 Multiparty SMC^2 semantic rules for writing to an array. 601
5.23 Multiparty SMC^2 semantic rules for reading and writing an entire array. 602
5.24 Multiparty SMC^2 semantic rules for reading and writing out of bounds for arrays. 603
5.25 Multiparty SMC^2 pre-increment rules for private int variables and for private pointers. . . . 604
5.26 Multiparty SMC^2 semantic rules for the public pre-increment operator. 605
5.27 Multiparty SMC^2 semantic rules for memory allocation and deallocation, casting, and finding the address of a variable. 606
5.28 Multiparty SMC^2 semantic rules for functions and finding the size of a type. 607
5.29 Multiparty SMC^2 semantics for public binary operations. 608
5.30 Multiparty SMC^2 semantic rules for declarations, reading, writing, and sequencing. 609
5.31 Multiparty SMC^2 semantic rules for input. 610
5.32 Multiparty SMC^2 semantic rules for output. 611
5.33 Table of more complex Multiparty SMC^2 evaluation codes and their congruent Multiparty Vanilla C evaluation codes. 646
5.34 Table of Multiparty SMC^2 evaluation codes in $MSmcC$ and their congruent Multiparty Vanilla C evaluation codes in $MVanC$. 647
5.35 The Erasure function from Multiparty SMC^2 configurations to Multiparty Vanilla C configurations. 653

6.1 Example illustrating why single-statement resolution is more costly when modifying variables multiple times in both branches. 849
6.2 Simple private-conditioned branching examples. An example for private integer variables is shown in 6.2a, and for private pointers 6.2b. We show values in memory that change in the table to the left, and values for temporary variables that do not change in the table to the right. We indicate updates in memory in green. 850
6.3 Percentage Runtime Differences 852
6.4 Benchmarking program: pay-gap.c 854
6.5 Benchmarking program: LR-parser.c (Part 1/2) 855

6.6 Benchmarking program: LR-parser.c (Part 2/2) 856
6.7 Benchmarking program: h_analysis.c 857
6.8 Benchmarking program: private-branching.c 857
6.9 Benchmarking program: private-branching-mult.c 858
6.10 Benchmarking program: private-branching-add.c 859
6.11 Benchmarking program: private-branching-reuse.c 860
6.12 Computation of Hamming distance: notation $<n>$ denotes the size of integers in bits and @ the dot product. 875
6.13 DSL Grammar. The color red denotes terms specific to programs written in the DSL extension of SMC^2, and the color blue denotes terms synthesized by the semantics. 876
6.14 Percentage of runtime speedup for optimized variable size tests. 891
6.15 Optimal Variable Sizes - Test 1 892
6.16 Optimal Variable Sizes - Test 2 893
6.17 Optimal Variable Sizes - Test 3 893
6.18 Optimal Variable Sizes - Test 4 894
6.19 Optimal Variable Sizes - Test 5 894
6.20 Optimal Variable Sizes - Test 6 895

7.1 Securely calculating the gender pay gap for 100 organizations with additional information released. 909

1 Introduction

Secure Multiparty Computation (SMC) allows multiple parties to jointly compute over private data, revealing only the outcomes of the computation to designated recipients. Secure computation is needed in many domains, particularly the medical, military, and financial sectors. SMC is commonly implemented using low-level techniques like secret sharing [1], garbled circuits [2], or homomorphic encryption [3, 4]. These low-level techniques are designed to enable computations among parties which are secure and efficient. While these low-level techniques provide efficiency, their use makes programming SMC applications challenging and error prone. To address this concern, several works have proposed high-level languages, DSLs, or language extensions providing abstractions, which can then be compiled down to low-level SMC primitives, to support programmers in writing SMC applications. As a result, there now exists a plethora of languages providing different expressivity, offering different features and performance trade-offs, using different threat models, and suitable for different domains. Similarly, there exist a number of SMC DSLs. Although DSLs can make it easier to write SMC programs, there remains a disconnect between the DSL and its integration with the language of the underlying implementation of the compiled protocol.

In the effort to unify our knowledge in this space, a recent work [5] compared several compilers and tools in terms of their expressivity and usability. We highlight two items among the lessons learned and recommendations to the community. First, there were numerous correctness issues and undocumented limitations present in the works surveyed. This finding is also echoed in [6], which found correctness issues in several two-party compilers. Second, the authors of [5] recommend that the community take a more principled approach to language design and verification, e.g., by defining and implementing type rules. This would help with ensuring correctness as well as reduce security-related corner cases overlooked by the compiler designers.

To help achieves these goals, we present a formalism of a general-purpose SMC system designed for C. We choose C because it provides the low level language framework targeted by most DSLs and there exist numerous direct language extensions for which multiple SMC compilers have been developed. This allows programmers to write secure multiparty programs in C, which the compiler will translate into secure

computation protocols, avoiding managing the interactions between different languages. Given the maturity of SMC compilers today, modern implementations provide support for all C features, allowing private-conditioned branches (i.e., **`if`**–**`else`** statements whose conditional expressions use private data), use of private arrays, private indexing into arrays, and working with pointers to private data; all while ensuring that no private data is leaked over the execution of any given program. However, formally modeling semantics and translation of these features as done by SMC compilers presents non-trivial challenges not attempted in any prior work. Furthermore, what is interesting about C (and not present in the available well-typed DSLs) is that features such as pointer manipulation allow one to write programs that erroneously access unintended regions in memory. However, even in those circumstances, it is possible to show that the compiled protocol will not reveal any unintended information about private data that it handles.

Our contributions in this thesis are:

1. Basic SMC^2, a basic formal model for a general-purpose secure multiparty computation compiler, formalizing state of the art SMC techniques in C. Our formal model supports distributed multi-party computation in the presence of private-conditioned branches, pointers to private data, pointer arithmetic and select pointer operations in private-conditioned branches.

2. Location-tracking SMC^2, an extended formal model enabling the above as well as support for full pointer operations inside private-conditioned branches.

3. Multiparty SMC^2, an optimized, multiparty formal model that explicitly shows how SMC protocols will connect with the formal model while enabling all C functionality for general-purpose programming in an efficient manner.

4. formal proofs based on each model, illustrating that common SMC approaches guarantee correctness and a strong form of non-interference over execution traces consisting of multiple computing parties. This shows that pointer operations can be safely managed with no restrictions on the program.

5. an implementation of our formal model, Multiparty SMC^2, in the PICCO SMC compiler and evaluation over micro-benchmarks and SMC programs.

6. a formalization of a domain specific language (DSL) extension to our formal model, including an optimization for obtaining and utilizing optimal variable sizing to increase program efficiency.

7. an implementation of the DSL and optimization on top of our implementation of Multiparty SMC^2 in the PICCO SMC compiler, with an evaluation over micro-benchmarks.

In this, we have found there are still some aspects (e.g. indirectly modifying data inside branches based on private data) that are valid C code, but not accepted by PICCO currently. With our implementation, we aim to ensure the compatibility of PICCO with the entire C language.

2 Background and Literature Review

SMC compilers To encourage more widespread use of SMC systems, we must make such systems easy and desirable to use. In order to achieve this, we need to ensure several things. First and foremost, we need to assure the user that the SMC system can be trusted; to accomplish this, we need to prove that the system will maintain the security of their private data and execute the program properly (i.e., by providing a formal model that is proven to be secure and correct). Another important component is allowing users to write general-purpose programs, which can be enabled by including a comprehensive set of language features and adding syntax to facilitate common functionalities (e.g., built in functions for input and output of data, enabling operations over arrays, etc.). We will explore several SMC systems and how well they accomplish these goals here.

Work on SMC compilers was initiated in 2004 and a significant body of work has been developed. Notable examples include two-party computation compilers and tools Fairplay [7], TASTY [8], ABY [9], PCF [10], TinyGarble [11], Frigate [6], SCVM [12], and ObliVM [13]; three-party Sharemind [14]; and multi-party FairplayMP [15], VIFF [16], and more recently SCALE-MAMBA, which evolved from [17–19]. These compilers use custom DSLs to represent user programs, and notable exceptions are CBMC-GC [20] (intended to support general-purpose ANSI-C programs in the two-party setting, but not all features were realized at the time) and PICCO [21, 22] (takes programs written in an extension of C, supports all C features, and produces multi-party protocols). The above compilers did not come with a formalism of their type system [1], while this was later developed for Sharemind [23]. There are also SMC DSLs with formal models, such as Wysteria [24] with a formal model based on an operational semantics and Wys* [25] which provides support for SMC by means of an embedded DSL hosted in F*, a dependently typed language supporting full verification. A different approach is given in [26] with an automated technique to prove SMC protocols secure.

We provide a summary of significant features supported in recent compilers in Table 2.1 (Wys* [25] inherits its expressivity from Wysteria and is omitted). They are supporting loops, private-conditioned

[1] The ObliVM publication [13] suggests that there is a type system behind the ObliVM language, but no further information could be found.

Compiler	Supported features					
	loops	private conditional	mixed mode	floating point	dynamic memory	semantic formalism
Fairplay [7]	◑	●	○	○	○	○
Sharemind [14, 27]	●	◑	●	●	○	●
CBMC-GC [20]	◑	●	○	◑	○	○
PICCO [21, 22]	●	●	●	●	●	○
SCALE-MAMBA	●	●	●	●	○	○
Wysteria [24]	●	●	●	○	○	●
Frigate [6]	●	●	○	○	○	○
ABY [9]	◑	◑	●	●	○	○
ObliVM [13]	●	●	●	●	●	○
SCVM [12]	●	●	●	○	○	●

Table 2.1: Language features supported in SMC compilers.

branches, supporting both private and public values (mixed mode), floating point arithmetic on private values, dynamically allocated memory, and having semantic formalism. Note that compilers that translate computation into Boolean circuits such as CBMC-GC need to unroll loops and thus can only support a bounded number of loop iterations, denoted as ◑ in the table. ABY also appears to have this limitation and for that reason expects input sizes at compile time. Recent compilers that work with a circuit representation (e.g., Wysteria, ObliVM) store compiled programs using intermediate representation and perform loop unrolling and circuit generation at runtime. To the best of our knowledge, Sharemind permits updating only a single variable in a private-conditioned branch (i.e., `if (cond) a = b; else a = c;`). Similarly, in ABY the programmer has to encode all logic associated with conditional statements using multiplexers. CBMC-GC did not support floating point arithmetic based on open-source software at the time of publication.

Dynamic memory management is often not discussed in prior work. CBMC-GC is said to support dynamic memory allocation, as long as this can be encoded as a bounded program, but the use of dynamic arrays and memory deallocation is not mentioned. PICCO explicitly supports C-style memory allocation and deallocation as well as dynamic arrays. ObliVM does not explicitly discuss dynamically allocated arrays, but we believe they are supported. Similarly, out-of-bounds array access in user programs is also not typically discussed in the SMC literature. Therefore, it is difficult to tell what the behavior might be, i.e., whether the compiler checks for this and, if not, whether the behavior of the corresponding compiled program is undefined. Wysteria and PICCO are two notable exceptions: Wysteria has a strongly typed language and will prevent such programs from compiling (recall that it supports only static sizes). PICCO will compile programs with out-of-bounds memory accesses. While the behavior of such programs is undefined in C (and no correctness guarantees can be provided), its analysis demonstrates that no privacy violations take place.

We formalize this behavior in this work.

Non-interference Non-interference is a standard information flow security property guaranteeing that information about private data does not directly affect publicly observable data. We will show non-interference over executions of programs using the formal model and its extension developed in this paper to prove security when SMC techniques and C language primitives are composed. Non-interference and its several variants have been extensively studied by means of language-based techniques, including type systems [28, 29], runtime monitor [30, 31], and multi-execution [32], to cite a few. One of the challenges in guaranteeing non-interference when attackers can inspect the state of the computation is to guarantee that private information is not implicitly leaked by means of the control flow path, i.e., that the computation is data-oblivious. Several language-based methods have been designed to guarantee that systems are secure against leakage from branching statements, including timing analysis [33] and multi-path execution [34–36]. In particular, [35] considered an approach similar to the one we use here. However, these approaches do not prevent private data leakage from explicit memory management. Building on these early works, several recent works [37, 38] have shown that in the context of secure compilation the natural notion that one needs to consider is a form of non-interference extended to traces. Inspired by this work, this is the notion we use in this paper when reasoning about non-interference.

Motivation We next motivate the need for formalization and discuss the challenges formalizing main language features presents. Let us consider an example SMC program that demonstrates how computation with private values can be specified and carried out. Figure 2.1 presents an SMC program that securely computes the average salary of employees in a particular field by gender. This is representative of a real-world SMC deployment in the City of Boston gender pay gap study [39], which evaluated pay inequalities by gender and race. While we list a simple program, the computation can be extended to securely compute comprehensive statistical information by gender, race, and other relevant attributes.

In this example, there are 100 organizations (line 1), each contributing a number of records about their employees, specified in the form of salary-gender pairs (lines 3–4). All records are private and must be protected from all participants. The computation proceeds by checking the gender field in a record and using the corresponding salary in the computation of either female or male average salary. Once the average salaries are computed privately, they are combined with publicly available historical averages using weighted average

```
1 public int numParticipants=100, maxInput=100, inputSize[numParticipants];
2 public int historicFemaleSalAvg, historicMaleSalAvg, i, j;
3 private int salary[numParticipants][maxInput], maleCount=0;
4 private int gender[numParticipants][maxInput], femaleCount=0;
5 private int avgMaleSalary=0, avgFemaleSalary=0;
6
7 smcinput(historicFemaleSalAvg,1); smcinput(historicMaleSalAvg,1);
8 for (i=0; i < numParticipants; i++){
9   smcinput(inputSize[i], i+1);
10  smcinput(gender[i], inputSize[i], i+1);
11  smcinput(salary[i], inputSize[i], i+1);}
12
13 for (i=0; i < numParticipants; i++){
14   for (j=0; j < inputSize[i]; j++){
15     if (gender[i][j] == 0) {
16       avgFemaleSalary += salary[i][j];
17       femaleCount++;}
18     else {
19       avgMaleSalary += salary[i][j];
20       maleCount++;}}}
21
22 avgFemaleSalary=(avgFemaleSalary/femaleCount)/2 + historicFemaleSalAvg/2;
23 avgMaleSalary=(avgMaleSalary/maleCount)/2 + historicMaleSalAvg/2;
24
25 for (i=1; i < numParticipants+1; i++){
26   smcoutput(avgFemaleSalary, i);  smcoutput(avgMaleSalary, i);}
```

Figure 2.1: Securely calculating the gender pay gap for 100 organizations.

computation (lines 22-23).

When we talk about secure computation, we need to distinguish between the values which must be protected throughout the computation (and are not revealed to the participants) and the values requiring no protection which are therefore observable during the computation. In our specification, the former are marked as *private* (e.g., salary-gender pairs and information derived from them) and the latter are marked as *public* (e.g., the number of records that each organization contributes and historical average salaries). The ability to combine computation with private and public values is called *mixed-mode* execution.

In our example, all inputs (public or private) are entered into the computation using the `smcinput` interface that expects the variable name, its dimension(s) (for array variables) and the index of the party supplying the input. When a private input enters the computation, it uses a different, cryptographically protected representation in accordance with the underlying secure multi-party computation techniques. That representation is maintained throughout the execution, which means that all operations on private data are carried out using the corresponding secure multi-party protocols and their true values are not observable. All

public values are handled in the same way as in conventional computation, i.e., an SMC compiler does not modify the computation and the values are observable by any participant running the computation. Upon computation completion, any output is communicated to the intended recipient via the `smcoutput` interface. If `smcoutput` is used with a private variable, its true value can be recovered from its cryptographically protected representation by the output recipient only (and remains unknown to all other parties). Because output recovery happens after the computation completes, the disclosed output is the intended outcome of secure computation and is not subject to the security guarantees maintained during the computation (such as non-interference).

The above interface distinguishes between different types of participants: those who *supply inputs*, those who *learn outputs*, and the parties *carrying out the computation*. The computational parties can be different from input owners and output recipients and their selection may be based on the properties of the underlying secure multi-party computation techniques. For example, there are often constraints with respect to what fraction of computational parties can collude, i.e., combine their individual views during the computation, in order to maintain security of private values. This means that in our example, the participants can select a subset of them to run the computation or employ other parties such as cloud computing providers.

The main property this design guarantees is that a computational party that is not an output recipient should learn nothing about the private values it handles during the computation. To formalize this property about SMC programs we will show non-interference between private and public computations in mixed-mode programs. Non-interference ensures that private data does not directly affect publicly observable data and is crucial for mixed-mode execution. For the example given in Figure 2.1, non-interference ensures that the private data (e.g., `salary[i][j]`) does not affect any public data (e.g., `historicFemaleSalaryAvg`).

This is a distributed, mixed-mode computation, computed between multiple participants. Each individual salary should be kept private and none of the participants should be able to deduce the salary (up to some number of colluding computational parties – this is dependent on the cryptographic protocols used). The computation is mixed-mode as it includes portions which are *protected* – computations over private data (e.g., finding the sum of all the salaries and counting the total number of males and females), and portions which are *unprotected* – local computations over public data (e.g., dividing the historic salaries by 2 in line 22 and 23). Secure computations by definition are distributed and governed by cryptographic protocols (e.g., conditional addition `sumFemaleSalary += salary[i][j];` on line 11). This occurs because to do the increment operation to compute the total sum, each participant must interact during the operation.

Given a program such as the one in our example, an SMC compiler will perform certain transformations such as substituting an operation on private values with the corresponding SMC protocol for performing that operation securely. We detail these in the following subsection through a series of motivating examples. In addition, the SMC compiler also has to perform structural transformations to programs to ensure that there is no information flow from private to public variables based on the instructions that a computational party executes. This is called *data-oblivious* (or data-independent) execution. This, for example, means that for private-conditioned branches, the execution must not reveal which branch gets executed. Our formal model must also ensure *data-oblivious* execution. We formalize this by showing that non-interference holds not only when considering the input-output functional behavior of a program, but also when considering the traces of execution of programs.

3 Basic SMC^2

In this Chapter, we present Basic SMC^2, a formalization of a general purpose SMC compiler. This formalization follows common conventions in SMC as discussed in the previous chapter for handling private-conditioned branches and array accesses at private indices, as well as providing support for pointers following PICCO's pointer implementation. It is created to address the need for an SMC compiler with a formal model that embodies the entire system, from the higher-level source program to the lower-level SMC libraries used by the compiler. To do this, we have created a formal model for the semantics of general purpose SMC programs written in C that allows for easy substitution of lower-level SMC libraries, and proceeded to proofs of correctness and noninterference.

We will first present the formal semantic model, starting with the memory model, then the corresponding standard C semantics, the Basic SMC^2 semantics, and the algorithms used within the semantics. Afterwards, we will present the metatheory, showing correctness of Basic SMC^2 with respect to standard C semantics and non-interference to guarantee that no leakage of private data occurs in Basic SMC^2.

3.1 Formal Semantics

In this Section, we introduce our semantic and memory model. We therefore introduce two models, one for standard C (referred to as Vanilla C) as well as the semantics for the SMC compiler (referred to as Basic SMC^2). We do not abstract away memory, instead we introduce a byte-level memory model, inspired by the memory model used by CompCert [40], a formally verified C compiler. Specifically, we build from their approach of byte-level representation of data and permissions. First, we present the grammar and memory model; second, the Vanilla C semantics corresponding to our Basic SMC^2 model; third, the formal semantic model for Basic SMC^2; and finally, the algorithms used in these model.

3.1.1 SMC^2 Grammar

Figure 3.1 gives the combined Vanilla C and SMC^2 grammar, which is a subset of the ANSI C grammar. This grammar remains consistent throughout our improvements to the SMC^2 model presented later in Chapter 4

$$
\begin{array}{llll}
ty & \in \mathit{Type} & ::= & a\ bty \mid a\ bty* \mid bty \mid bty* \mid \overline{ty} \rightarrow ty \\
bty & \in \mathit{BasicType} & ::= & \text{int} \mid \text{float} \mid \text{void} \\
a & \in \mathit{PrivacyLabel} & ::= & \text{private} \mid \text{public} \\
 & & & \\
s & \in \mathit{Statement} & ::= & var = e \mid *x = e \mid s_1;\ s_2 \mid decl \mid \{s\} \mid \text{if}\ (e)\ s_1\ \text{else}\ s_2 \mid \text{while}\ (e)\ s \mid e \\
e & \in \mathit{Expression} & ::= & e\ bop\ e \mid uop\ x \mid var \mid x(\overline{e}) \mid prim \mid (ty)\ e \mid (e) \mid v \\
decl & \in \mathit{Declaration} & ::= & ty\ var \mid ty\ x(\overline{p}) \mid ty\ var = e \mid ty\ x(\overline{p})\{s\} \\
var & \in \mathit{Variable} & ::= & x \mid x[e] \\
v & \in \mathit{Value} & ::= & n \mid \overline{v} \mid \text{NULL} \mid (l, \mu) \mid ptr \mid \text{skip} \\
ptr & \in \mathit{PointerData} & ::= & [\alpha, \overline{l}, \overline{j}, i] \\
\overline{l} & \in \mathit{LocationList} & ::= & [\,] \mid (l, \mu) :: \overline{l} \\
 & & & \\
prim & \in \mathit{PrimitiveFunction} & ::= & \text{sizeof}(ty) \mid \text{malloc}(e) \mid \text{pmalloc}(e,\ ty) \mid \text{free}(e) \mid \text{pfree}(e) \\
 & & & \mid \text{smcinput}(x, e) \mid \text{smcoutput}(x, e) \mid \text{mcinput}(x, e) \mid \text{mcoutput}(x, e) \\
bop & \in \mathit{BinaryOperation} & ::= & - \mid + \mid \cdot \mid \div \mid == \mid != \mid < \\
uop & \in \mathit{UnaryOperation} & ::= & \& \mid * \mid ++ \\
\overline{e} & \in \mathit{ExpressionList} & ::= & \overline{e},\ e \mid e \mid \text{void} \\
\overline{p} & \in \mathit{ParameterList} & ::= & \overline{p},\ ty\ var \mid ty\ var \mid \text{void}
\end{array}
$$

Figure 3.1: Combined Vanilla C/SMC^2 Grammar. The color red denotes terms specific to programs written in SMC^2, and the color blue denotes elements synthesized by the semantics.

and Chapter 5. We include one dimensional arrays, branches, loops, dynamic memory allocation, and pointers. Arrays are zero-indexed, and it is possible to overshoot their bounds, as in C. We chose not to include structs or multi-dimensional arrays, as they are an extension of this core subset. We currently assume there are no arrays of pointers, as this is a trivial extension of arrays and pointers separately and is not fully supported by the model due to the number of elements in an array and the number of locations for a pointer using the same meta-data within each memory block. If one wishes to extend the model to support arrays of pointers, they simply need to add another piece of meta-data to each memory block so as to have the number of elements of an array and number of locations stored by pointers as separate elements.

Binary operations follow the standard order of operations. We use the bar notation to indicate a list of elements (e.g., $\overline{x}$ as a list of variables), with the exception of location lists $\overline{l}$ being a list of the pair of the memory block identifier and offset (l, μ), as this constitutes a specific location within our memory model (this concept is described further in the following subsection). We use the color red to denote terms in the SMC^2 grammar that are not present in Vanilla C, including annotated types ($a\ bty$, $a\ bty*$), privacy labels (public, private), and primitive functions (pmalloc, pfree) for allocation and deallocation of memory for private pointers. Additionally, we have primitive functions for facilitating input and output in SMC^2 (smcinput(x, e), smcoutput(x, e)) and Vanilla C (mcinput(x, e), mcoutput(x, e)). We use the color blue to denote terms that are synthesized by the semantics. These include the function type $\overline{ty} \rightarrow ty$; locations (l, μ), pointer data structures ptr, and the terminal skip.

We denote types as ty, basic types as bty ($bty*$ as a pointer type), privacy annotations as a, and function types as $\overline{ty} \rightarrow ty$ (where $\overline{ty}$ is a type list). Values v include numbers n, lists of values, NULL, locations (l, μ) consisting of a memory block identifier l and an offset μ, and skip (to show a statement being reduced to completion). Declarations include variable and function declarations, where $\overline{p}$ is the function parameter list. For unary operations, we include: $\&$, to obtain the address of a variable; $*$, to allow dereferencing pointers; and $++$, to allow pre-incrementing and to model a basic pointer arithmetic.

3.1.2 SMC^2 Memory Model

$$
\begin{array}{rcll}
C & \in \mathit{Configuration} & ::= & \epsilon \mid (\gamma, \sigma, \mathrm{acc}, s) \\
\gamma & \in \mathit{Environment} & ::= & f : x \rightarrow (l,\ ty) \\
\sigma & \in \mathit{Memory} & ::= & f : l \rightarrow (\omega,\ ty,\ \alpha,\ \overline{perm}) \\
perm & \in \mathit{Permission} & ::= & \mathrm{Freeable} \mid \mathrm{None} \\
\overline{perm} & \in \mathit{PermissionList} & ::= & [\,] \mid (n,\ a,\ perm) :: \overline{perm} \\
n, m, i, \alpha & \in \mathit{Numbers} & \in & \mathbb{N} \\
l & \in \mathit{MemoryBlockIdentifier} & \in & \mathbb{N} \\
\mu & \in \mathit{LocationOffset} & \in & \mathbb{N} \\
j & \in \mathit{TagBit} & ::= & 0 \mid 1 \\
\omega & \in \mathit{ByteRepresentation} & ::= & \{0 \mid 1\}^+ \\
d & \in \mathit{EvaluationCode} & ::= & \{a...z \mid 0...9\}^+
\end{array}
$$

Figure 3.2: Basic SMC^2 configuration: environment γ, memory σ, accumulator acc, and statement s.

Our memory model encodes each memory as a contiguous region of *blocks*, which are sequences of bytes and metadata. We introduce an execution environment γ and memory σ, shown in Figure 3.2. Each block is assigned an identifier l (to be discussed more later in this section). Blocks are never recycled nor cleared when they are freed. We chose this view of memory to preserve all allocated data, which, in conjunction with data-oblivious execution, represents the worst case for maintaining privacy. Direct memory access through pointers or manipulation of array indices allows programs to access any block for which the memory address is computable (e.g., as an offset or direct pointer access). To obtain the byte representations of data we leverage functions similar to CompCert, using $\mathrm{EncodeVal}$ for values, $\mathrm{EncodePtr}$ for pointers, and $\mathrm{EncodeFun}$ for functions. Likewise, to obtain the human-readable data back, we use respective decode functions such as $\mathrm{DecodeVal}$. We use a specialized version ($\mathrm{DecodeArr}$) for obtaining a specific index within an array data block. We introduce the particulars in the following subsections.

Environment

In our semantics, we implement standard C scoping through our use of the environment γ, which maintains a mapping of each live variable x to its memory block identifier (where the data x is stored) and its type. At the start of a program, the environment is empty, i.e., $\gamma = [\,]$. Variables that are no longer live are removed from the environment, based on standard scoping within the semantic rules. We use the environment to facilitate the lookup of variables (i.e., for reads, writes, function calls) in memory σ.

Memory Blocks and Identifiers

The memory, σ, is a mapping of each identifier l to its memory block, which contains the byte representation ω of data stored there and metadata about the block. Metadata consists of a type ty associated with the block, the number of elements n of that type stored in ω, and a list of byte-wise permissions tuples $[(0,\ a_0,\ perm_0), ..., (m,\ a_m,\ perm_m)]$, where $m = \tau(ty) \cdot n - 1$ and function τ provides the size of the given type in bytes. A new memory block identifier is obtained from function ϕ. These identifiers are monotonically increasing with each allocation. Every block is added to memory σ on allocation, and is never cleared of data nor removed from σ upon deallocation. Metadata cannot be accessed or modified directly by the program (the semantic rules control modification). A memory block can be of an arbitrary size, which is constant and determined at allocation (with the exception of private pointers, to be discussed later in subsection 3.1.4). We represent a memory location as a two-tuple of a memory block identifier and an offset. This allows us to use pointers to refer to any arbitrary memory location, as in C.

Permissions

A permission $perm$ can either be $\mathrm{Freeable}$ (i.e., can be written to, read from, etc.) or None (i.e., already freed). These byte-wise permissions are modeled after a subset of those used by CompCert, and we extend their permission model by including a privacy label. Each memory block has a list of permission tuples, one for each byte of data stored in that block. A permission tuple consists of the position of the byte that it corresponds to, and the privacy label a and permission $perm$ for that byte of data. These permissions are important in reading and writing data to memory, especially when it comes to overshooting arrays and other out-of-bound memory accesses possible through the use of pointers. In particular, permissions allow us to keep track of deallocated memory (e.g. a block freed - note that the memory stored in the block itself is not

overwritten or cleared and the block can still be accessed indirectly through direct memory manipulations).

All permission tuples corresponding to a memory block of a function type will have public privacy labels, as the instructions for a function are accessible from the program itself. Those for a normal variable or an array will have privacy labels corresponding to their type (i.e., public for public types, private for private types); those for pointers are more complex and will be discussed later in Subsection 3.1.2. For simplicity, in the semantics we use Algorithm 36 (PermL) to highlight the main information (permission, type, privacy label, and number of elements) about the byte-wise permissions for each memory block. Based on these arguments, this algorithm returns the full list of permission tuples for every byte of data stored in the memory block.

Malloc and Free

Allocation of dynamic memory in C is provided by `malloc`, which takes a number of bytes as its argument. When `malloc` is called, a new memory block with identifier l is obtained, initialized as a void type of the given size, and returned. This block then needs to be cast to the desired type. However, when dealing with private data, the programmer is unlikely to know the internal representation and the size of the private data types. For that reason, when allocating memory for private data, we adopt PICCO's `pmalloc` functionality which takes two arguments: the type and the number of elements of that type to be allocated. The semantics then handle sizing the new memory block for the given private type. When `free` or `pfree` is called, if the argument is a variable of a pointer type, the permissions for all bytes of the location the pointer refers to will be set to None, but the data stored there will not be erased, and the location will not be released back into the pool of available locations accessed by ϕ within the semantics. When `pfree` is called with a pointer with a single location, it behaves identically to `free`. The use of `pfree` with multiple locations is a bit more involved, to be discussed later in Subsection 3.1.4. It is important to note that memory allocation and deallocation are public side effects, and therefore are *not allowed* within private-conditioned branches.

Public vs. Private Blocks

To distinguish between public and private blocks, we assume that private blocks will be encrypted and we will use basic private primitives implementing specific operations to manage them. For modeling purposes, these primitives can decrypt the required blocks, perform the operations they are meant to implement, and encrypt the result. In our model, a program can also access private blocks by means of standard non-private

operations or through of pointers. In this case, the operation will just interpret the encrypted value as a public value. This approach gives us a conceptual distinction between a *concrete memory* and its corresponding *logical content*, i.e. public values and values of the private data prior to encryption. Our model as described in the next section will work on concrete memories, but in Section 3.3, for the proof of noninterference, it will be convenient to refer to the logical content of a memory. We will use the notation $\sigma\ell$ to denote the logical content of σ.

Pointer Data Structure and Permissions

In order to maintain data-oblivious execution, we need to allow storing multiple locations for pointers when they are modified within a private-conditioned branching statement. To achieve this, the structure of the data stored by pointers (i.e., $[\alpha, \bar{l}, \bar{j}, i]$) is as follows: the number α of locations being pointed to; a list $\bar{l}$ of α locations being pointed to; a list $\bar{j}$ of α tags; and the level of indirection i of the pointer. The number α of locations being pointed to will always be one for public pointers, and will only ever increase for private pointers through **`if else`** statements branching on private data (or being assigned the data from such a private pointer).

The privacy labels of the byte-wise permissions corresponding to the number α of locations to which the pointer refers will always be public, as it is visible to an observer of memory the number of locations that are touched by a pointer. The list of the locations being pointed to by the pointer will always only contain one location for public pointers, and for private pointers will correspond to the number α given above. The privacy labels of the permissions corresponding to these bytes will always be public (it is visible to the an observer of memory the locations touched by a pointer). Likewise, the permissions corresponding to the list of locations will will always be public, as it is visible to the an observer of memory the locations touched by a pointer. The list of tag bits for a public pointer will contain only the public integer 1. For a private pointer, the list will consist of α private integers; of these α integers, only one contains the encrypted representation of the value 1 to indicate the true location; all others will contain the encrypted representation of the value 0. The privacy labels of the permissions corresponding to the tags of public pointers will always be public; for private pointers, they will be private, as these protect an observer of memory from being able to tell which location is the true location. Lastly, the level of indirection of the pointer will be greater than or equal to one. This is defined at the time the pointer is declared (i.e., $\mathrm{int}* \rightarrow 1$, $\mathrm{int}**\rightarrow 2$, etc.). The privacy labels of the permissions corresponding to the level of indirection will always be public, as this is visible in the source

program.

Overshooting Memory Bounds

It is possible to overshoot memory bounds in both Vanilla C and SMC^2. When overshooting occurs, we read the bytes of data as the type we expected it to be (i.e., bytes containing private data accessed by a public variable would be decoded as though they are public - no encryption or decryption occurs, but computations using the variable beyond that point will be garbage). This ensures that no information about private data can be leaked when overshooting. We will explain the different possibilities for overshooting using the example of an array, although each case we discuss below is also possible through the use of pointers as well.

Figure 3.3: Types of overshooting array accesses.

Figure 3.3 shows an example of an array read that overshoots the bounds of the array `x` (for simplicity, `x` is of size 1). The first access shown is an in-bounds access – this is the default behavior of a correct program. The second access is an out-of-bounds access of a memory block of a different type, but the same size. This data would be read as if it was the type of the array, and may not be meaningful. This corresponds to an access where implicit conversions between types is possible, but not always correct. The third out-of-bounds access corresponds to reading out of a memory block of the same type. This data would be meaningful from a type perspective, but the specific value read may not be semantically meaningful to the program. The fourth out-of-bounds access is of a memory block of a different type of a smaller size. This read would grab the data from the smaller memory block, then grab data from the next memory block(s) to obtain the correct amount for the expected type. In this situation a value, which may not be meaningful, is constructed from two, or more, values in memory. The last out-of-bounds access is of a larger memory block, not aligned. This read would obtain a portion of the data of the larger memory block, and read it as the type of the array, thereby reading a partial value from memory.

With SMC^2, when dealing with array overshooting, we have the added complexity of private data, which has a different representation and is of a larger size than the corresponding C representation of the type. Additionally, we need to ensure that no leakage can occur, so we must consider all possible combinations of bytes from public and private data with either public or private variables. Consider reading a value from an overshot array and storing it into a variable. If both the data read and variable are private or both are public, no leakage can occur as these are the default cases. Next, consider reading public data and storing into a private variable. The public data will be grabbed at the byte-level, and interpreted as though it is private (no encryption will occur), so no leakage occurs. Third, we consider reading private data and storing in a public variable. The private data will be grabbed at the byte-level, and interpreted as though it is public. No decryption will occur, so no leakage can occur. This is similar in nature to reading a partial value in Figure 3.3. Fourth, consider if the data read is a mix of public and private data and stored in a public variable. Given that the private data will not be decrypted, this read will not result in any leakage, but a value is constructed from a mix of private (encrypted) data and public data. Lastly, consider reading a mix of public and private data and storing into a private variable. Like before, the byte-level data will be merged and read as the expected type.

Writes that occur out-of-bounds of an array have situations similar to out-of-bound read accesses (and can be illustrated as with the reads shown in Figure 3.3). Writing private data out-of-bounds to a private location results in the data still residing in a private memory block, so no leakage will occur. Writing public data out-of-bounds to a public location is safe, as the data is already public. When writing private data out-of-bounds to a public location, the data will be written as-is – no decryption will occur when the data is written to or later read back from that location – therefore, there is no leakage. Writing public data out-of-bounds to a private location or a mix of public and private locations is safe, as the data was already public; no encryption will occur. Lastly, writing private data out-of-bounds to a mix of locations will result in the data being written to the locations as-is. No decryption will occur when the data is written to any location or later read back, therefore, there is no leakage.

In SMC^2, we ensure this behavior, using algorithms $\mathrm{ReadOOB}$ and $\mathrm{WriteOOB}$. In particular, $\mathrm{ReadOOB}$ ensures that no matter what mix of byte-wise data we grab from memory, we will decode it as a value of the type of data in the array, ignoring it's true type. Similarly, $\mathrm{WriteOOB}$ ensures that we will write to memory the byte-wise encoding of the given value as the type for the array, without taking into consideration the type of the memory block(s) and without modifying any of the metadata within the memory block(s) we write to.

(a) Well-aligned accesses

(b) Not well-aligned accesses

Figure 3.4: Examples of alignment between SMC^2 and Vanilla C in overshooting accesses by incrementing pointer $\mathtt{p}$ three times.

Overshooting Alignment

In proving the correctness of SMC^2 with relation to Vanilla C, the various possible alignments for reading and writing out-of-bounds pose complications due to the different sizes of private and public data (an example of this is shown in Figure 3.4). Therefore, we can only prove correctness over *well-aligned* accesses (i.e., those that iterate only over aligned elements of the same type, as with one array spilling into a subsequent array), as these would still produce readable data that is not garbage. For the correctness proof, we provide a formal definition of a *well-aligned* overshooting memory access in Definition 3.2.2. We discuss this in more detail in the following section. When proving noninterference, we must prove that these cases (particularly those involving private data) cannot leak any information about the private data that is affected.

3.1.3 Vanilla C

In order to facilitate the correspondence between the Vanilla C and SMC^2 semantics, we model our semantics using big-step evaluation judgements and define our C semantics with respect to multiple *non interacting* parties that evaluate the same program. In Vanilla C, we use $\widehat{\ }$ to distinguish elements in this semantics from those we use in the next section for SMC^2 semantics, which may differ due to privacy labels and private data being encrypted. The semantic judgements in Vanilla C are defined over a four-tuple configuration $\widehat{C} = (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$, where each rule is a reduction from one configuration to a subsequent. We denote the environment as $\widehat{\gamma}$; memory as $\widehat{\sigma}$; a placeholder as $\square$; and a big-step evaluation of a statement $\widehat{s}$ to a value $\widehat{v}$ using $\Downarrow'$. We use $\square$ in Vanilla C as a placeholder for the level of nesting of private-conditioned branches acc to maintain the same shape of configurations as that of Basic SMC^2 used in the next section. We annotate each evaluation with evaluation codes (i.e., $\Downarrow'_{\widehat{d}}$) to facilitate reasoning over evaluation trees, and we annotate

evaluations that are not *well-aligned* with a star (i.e., $\Downarrow'^{*}_{\widehat{d}}$) to identify the rules that we cannot prove correctness over, as they produce unpredictable behavior in implementation. We show the semantic rules that are not *well-aligned* in this chapter to illustrate how they are defined and handled, but we omit showing them in later chapters as they are nearly identical to their corresponding rules and the proof of noninterference over these rules are handled similarly to the cases of the corresponding rules. The assertions in each semantic rule can be read in sequential order, from left to right and top to bottom.

In this section, we will present the Vanilla C semantics with respect to the grammar (Figure 3.1). These semantic rules follow standard C; however, we will describe the Vanilla C rules to familiarize the reader with our notation. It is worthwhile to note here that all permissions in Vanilla C will be set to public, and all types will be implicitly public, as there is no notion of privacy labels in standard C. We will store pointer data within the pointer data structure to facilitate reasoning about pointers between Vanilla C and SMC^2, but the Vanilla C pointers can only have a single location and will always have the single tag in the tag list set to 1, as that is the only possible location for the pointer to refer to.

Figure 3.5 gives the semantic rules for declarations, reading, writing, and pre-incrementing of regular (non-pointer, non-array) variables, loops, sequencing, finding the size of a type, retrieving the address of a variable, and binary operations. Figure 3.6 provides the semantics for functions, casting, memory allocation and deallocation, and inputting and outputting data from files. Figure 3.7 gives the semantics for branches as well as pointer declarations, reading and writing, and pre-incrementing. Figure 3.8 provides the semantics for pointer dereferences and pointer dereference writes. Figure 3.9 gives the semantics for array declarations, writing and reading, including reading and writing out of bounds. Figure 3.10 gives the semantics for array declaration assignment and reading from and writing to an entire array. These model regular C semantics and are obtained from the SMC^2 semantics using the Erase function (discussed further in Section 3.2.1). The behavior of these rules is mostly standard or mirrors the description of the rules presented in the paper itself.

In semantic rule Declaration, we need to obtain a new memory block and create new mappings in the environment and in memory for the variable being declared. We have the starting configuration ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{bty}\ \widehat{x}$) with our starting statement being the declaration $\widehat{bty}\ \widehat{x}$. To evaluate this statement, we first obtain a new memory block identifier $\widehat{l}$ from the pool of unassigned memory blocks using ϕ. We then create a mapping from $\widehat{x}$ to its new memory block and its given type and add the mapping to environment $\widehat{\gamma}$ to obtain the resulting environment, $\widehat{\gamma}_1$. We create initial NULL byte-data $\widehat{\omega}$ for the memory block using Algorithm 7 ($\mathrm{EncodeVal}$). Then, we create a mapping from the memory block identifier $\widehat{l}$ to the four-tuple of the byte-data

Declaration

$$\frac{\widehat{l} = \phi() \qquad \widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty})] \qquad \widehat{\omega} = \text{EncodeVal}(\widehat{bty}, \text{NULL}) \qquad \widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{bty},\ 1,\ \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))]}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{bty}\ \widehat{x}) \Downarrow'_{\widehat{d}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})}$$

Declaration Assignment

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ x) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip}) \qquad (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ x = \widehat{e}) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ x = \widehat{e}) \Downarrow'_{\widehat{ds}} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})}$$

Read

$$\frac{\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}) \qquad \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1)) \qquad \text{DecodeVal}(\widehat{bty},\ 1,\ \widehat{\omega}) = \widehat{v}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{\widehat{r}} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})}$$

While Continue

$$\frac{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}) \qquad \widehat{n} \neq 0 \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{s}) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip}) \qquad (\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{while } (\widehat{e})\ \widehat{s}) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_2, \widehat{\sigma}_3, \square, \text{skip})}{(\widehat{\gamma}, \widehat{\sigma}, \square, \text{while } (\widehat{e})\ \widehat{s}) \Downarrow'_{\widehat{wlc}} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \text{skip})}$$

Write

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}) \qquad \widehat{v} \neq \text{skip} \qquad \widehat{\gamma}(x) = (\widehat{l}, \widehat{bty}) \qquad \text{UpdateVal}(\widehat{\sigma}_1, \widehat{l}, \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ x = \widehat{e}) \Downarrow'_{\widehat{w}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})}$$

While End

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}) \qquad \widehat{n}\ =\ 0}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{while } (\widehat{e})\ \widehat{s}) \Downarrow'_{\widehat{wle}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \text{skip})}$$

Parentheses

$$\frac{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v})}{(\widehat{\gamma}, \widehat{\sigma},\ \square, (\widehat{e})) \Downarrow'_{\widehat{ep}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v})}$$

Statement Block

$$\frac{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s}) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})}{(\widehat{\gamma}, \widehat{\sigma}, \square, \{\widehat{s}\}) \Downarrow_{\widehat{sb}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \text{skip})}$$

Size Of Type

$$\frac{\widehat{n} = \tau(\widehat{ty})}{(\widehat{\gamma}, \widehat{\sigma}, \square, \text{sizeof}(\widehat{ty})) \Downarrow'_{\widehat{ty}} (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{n})}$$

Subtraction

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2) \qquad \widehat{n}_1 - \widehat{n}_2 = \widehat{n}_3}{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 - \widehat{e}_2) \Downarrow'_{\widehat{bs}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_3)}$$

Statement Sequencing

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip}) \qquad (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{s}_2) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \widehat{v})}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1;\ \widehat{s}_2) \Downarrow'_{\widehat{ss}} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \widehat{v})}$$

Address Of

$$\frac{\widehat{\gamma}(x) = (\widehat{l},\ \widehat{ty})}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \&x) \Downarrow'_{\widehat{loc}} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}, 0))}$$

Addition

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2) \qquad \widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{\widehat{bp}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_3)}$$

Less Than True

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2) \qquad \widehat{n}_1 < \widehat{n}_2}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{\widehat{ltt}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ 1)}$$

Less Than False

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2) \qquad \widehat{n}_1 >= \widehat{n}_2}{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{\widehat{ltf}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, 0)}$$

Multiplication

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2) \qquad \widehat{n}_1 \cdot \widehat{n}_2 = \widehat{n}_3}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 \cdot \widehat{e}_2) \Downarrow'_{\widehat{bm}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_3)}$$

Equal To True

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2) \qquad \widehat{n}_1 = \widehat{n}_2}{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 == \widehat{e}_2) \Downarrow'_{\widehat{eqt}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, 1)}$$

Equal To False

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2) \qquad \widehat{n}_1 \neq \widehat{n}_2}{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 == \widehat{e}_2) \Downarrow'_{\widehat{eqf}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, 0)}$$

Division

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2) \qquad \widehat{n}_1 \div \widehat{n}_2 = \widehat{n}_3}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 \div \widehat{e}_2) \Downarrow'_{\widehat{bd}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_3)}$$

Not Equal To True

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2) \qquad \widehat{n}_1 \neq \widehat{n}_2}{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 != \widehat{e}_2) \Downarrow'_{\widehat{net}} (\widehat{\gamma}, \widehat{\sigma}_2, \square,\ 1)}$$

Not Equal To False

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2) \qquad \widehat{n}_1 = \widehat{n}_2}{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 != \widehat{e}_2) \Downarrow'_{\widehat{nef}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, 0)}$$

Pre-increment Variable

$$\frac{\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}) \qquad \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1)) \qquad \text{DecodeVal}(\widehat{bty},\ 1,\ \widehat{\omega}) = \widehat{v} \qquad \widehat{v}_1 = \widehat{v} + 1 \qquad \text{UpdateVal}(\widehat{\sigma},\ \widehat{l},\ \widehat{v}_1, \widehat{bty}) = \widehat{\sigma}_1}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x}) \Downarrow'_{\widehat{pin}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_1)}$$

Figure 3.5: Vanilla C semantic rules for basic variable declarations, reading, and writing; loops; sequencing; binary operations; and incrementing.

$\widehat{\omega}$, the type of data stored in this block $\widehat{bty}$, the number of elements stored in this block (which is 1, as this is not an array declaration), and the byte-wise permissions for the memory block, which is obtained through Algorithm 1 (PermL). The permissions for each byte of data for the one element of type $\widehat{bty}$ in this memory block are set to $\mathrm{Freeable}$ and public. This mapping is added to the memory $\widehat{\sigma}$ to obtain the resulting memory $\widehat{\sigma}_1$. As there is nothing further to evaluate in this rule and no value to be returned, we have the terminating value skip, as this statement has no possible further steps or uses. This gives us the ending configuration of $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \mathrm{skip})$.

In semantic rule Read, we need to return the value stored for the variable $\widehat{x}$. To evaluate this, we must first look up variable $\widehat{x}$ in the current environment $\widehat{\gamma}$, finding that $\widehat{x}$ is associated with the memory block identifier $\widehat{l}$ and type $\widehat{bty}$. We then look up $\widehat{l}$ in memory $\widehat{\sigma}$ to find the data stored for this variable as well as the number of elements stored within this memory block for this variable. Then we use Algorithm 8 with the type, number of elements, and byte representation $\widehat{\omega}$ to obtain the value $\widehat{v}$ that is stored for this variable. We then return configuration $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{bty}\ \widehat{v})$.

In semantic rule Write, we need to store the value resulting from the evaluation of expression $\widehat{e}$ in the memory block associated with the variable $\widehat{x}$. We first take the initial environment, memory, and expression and evaluate the expression to the ending state with potentially updated memory $\widehat{\sigma}_1$ and value $\widehat{v}$. We assert that $\widehat{v}$ is not skip, as that is not a valid value to store in memory. We perform the look-up of $\widehat{x}$ in the environment, then use the Algorithm 13 with memory $\widehat{\sigma}_1$, memory block identifier $\widehat{l}$, value $\widehat{v}$, and type of $\widehat{x}$ $\widehat{\omega}$ to obtain the final updated memory $\widehat{\sigma}_2$ where the given value $\widehat{v}$ is now stored for $\widehat{x}$. We then return the updated memory and terminating value skip.

In semantic rule Declaration Assignment, we facilitate the evaluation of the declaration of a new variable and the evaluation of the assignment of a value to this variable. The evaluation of the declaration will update both the environment and memory, and the evaluation of the assignment will update the memory again, giving us the resulting configuration with $\widehat{\gamma}_1$ and $\widehat{\sigma}_2$.

In semantic rule While Continue, we facilitate the continuation of loop evaluation. We first evaluate the condition e to some number $\widehat{n}$, then we assert that $\widehat{n}$ cannot be 0 (as 0 that would signal that the loop should end). Next, we evaluate the body of the loop, $\widehat{s}$. We then discard any updates to the environment, as they are out of scope, and proceed to evaluate the entire while statement again. The semantics will recursively enter this rule until it has reached the point where the condition becomes equal to 0, in which case the semantic rule While End will be evaluated to facilitate the end state of loop evaluation. Once While End has been

executed, we will return through all iterations of While Continue, dropping any changes to the environment that are now out of scope, but keeping all changes made to memory.

In semantic rule Parentheses, we evaluate the expression inside the parentheses and return the value and any changes to memory that we obtained. In semantic rule Statement Block, we evaluate the statement inside the braces. We discard any additions to the environment (as they are now out of scope) and pass along any modifications made to memory. In semantic rule Statement Sequencing, we facilitate the evaluation of sequences of statements, evaluating the first statement, then proceeding to evaluate the second statement and returning the resulting value. Here, we keep all changes made to the environment and memory that are returned to this rule from the evaluation of either statement, because this rule does not change scope.

In semantic rule Size Of Type, we are finding the size of the given type. Here, we use Algorithm 4 (τ), which will return the size corresponding to the type it is given. This size $\widehat{n}$ is then returned. In semantic rule Address Of, we are finding the address of the given variable. We perform a lookup in environment $\widehat{\gamma}$, and return the memory block identifier l with offset 0 as the location, as all variables start at offset 0 within their assigned memory block.

In semantic rule Addition, we perform the binary operation addition. First, we evaluate the expression $\widehat{e}_1$ to obtain $\widehat{n}_1$ and memory $\widehat{\sigma}_1$, then $\widehat{e}_2$ using memory $\widehat{\sigma}_1$ to obtain $\widehat{n}_2$ and memory $\widehat{\sigma}_2$, and then add $\widehat{n}_1 + \widehat{n}_2$ to get the sum $\widehat{n}_3$. We then return memory $\widehat{\sigma}_2$ and the resulting value $\widehat{n}_3$. The semantic rules for Subtraction, Multiplication, and Division are similar to Addition, simply substituting the appropriate operation in place of addition. Likewise, in semantic rule Less Than True we evaluate the two expressions sequentially to find their resulting values, then assert that $\widehat{n}_1 < \widehat{n}_2$ holds true. We then return the updated memory $\widehat{\sigma}_2$ and the value 1, indicating that the less than comparison was true. Semantic rules Equal To True and Not Equal To True are similar to Less Than True, simply substituting the appropriate comparison operation assertion in place of less than. In semantic rule Less Than False, we show that $\widehat{n}_1 < \widehat{n}_2$ is false by asserting that the opposite ($\widehat{n}_1 >= \widehat{n}_2$) is true, and return 0. The semantic rules Equal To False and Not Equal To False are similar to Less Than False, substituting in the appropriate comparison assertion that covers the false case for each comparison operation.

In semantic rule Pre-increment Variable, we first look up the variable $\widehat{x}$ in the environment, then in memory, and decode the byte representation to get the value for $\widehat{x}$, we then increment the value by 1 to get the resulting value $\widehat{v}_1$, which we store in memory using Algorithm 13 ($\mathrm{UpdateVal}$). The updated memory $\widehat{\sigma}_1$ and value $\widehat{v}_1$ are then returned.

Function Call

$$\frac{\begin{array}{l}\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{\overline{ty}} \to \widehat{ty}) \qquad \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{\overline{ty}} \to \widehat{ty},\ 1,\ \text{PermL_Fun(public)}) \qquad \text{DecodeFun}(\widehat{\omega}) = (\widehat{s},\ \square,\ \widehat{\overline{p}}) \\ \text{GetFunParamAssign}(\widehat{\overline{p}}, \widehat{\overline{e}}) = \widehat{s}_1 \qquad (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip}) \qquad (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{s}) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \text{skip})\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}(\widehat{\overline{e}})) \Downarrow'_{\widehat{fc}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{NULL})}$$

Pre-declared Function Definition

$$\frac{\begin{array}{l}\widehat{x} \in \widehat{\gamma} \qquad \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{\overline{ty}} \to \widehat{ty}) \\ \text{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}}) = \widehat{\omega} \\ \widehat{\sigma} = \widehat{\sigma}_1[\widehat{l} \to (\text{NULL}, \widehat{\overline{ty}} \to \widehat{ty}, 1, \text{PermL_Fun(public)})] \\ \widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \to (\widehat{\omega}, \widehat{\overline{ty}} \to \widehat{ty}, 1, \text{PermL_Fun(public)})]\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}) \Downarrow'_{\widehat{fpd}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})}$$

Function Definition

$$\frac{\begin{array}{l}\widehat{x} \notin \widehat{\gamma} \qquad \text{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}} \\ \widehat{l} = \phi() \qquad \widehat{\gamma}_1 = \widehat{\gamma}[x \to (\widehat{l}, \widehat{\overline{ty}} \to \widehat{ty})] \\ \text{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}}) = \widehat{\omega} \\ \widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \to (\widehat{\omega}, \widehat{\overline{ty}} \to \widehat{ty}, 1, \text{PermL_Fun(public)})]\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}) \Downarrow'_{\widehat{fd}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})}$$

Function Declaration

$$\frac{\begin{array}{l}\widehat{l} = \phi() \qquad \text{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}} \qquad \widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \to (\widehat{l},\ \widehat{\overline{ty}} \to \widehat{ty})] \\ \widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \to (\text{NULL},\ \widehat{\overline{ty}} \to \widehat{ty},\ 1,\ \text{PermL_Fun(public)})]\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})) \Downarrow'_{\widehat{df}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})}$$

Cast Value

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}) \\ \widehat{n}_1 = \text{Cast}(\text{public},\ \widehat{ty},\ \widehat{n})\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e}) \Downarrow'_{\widehat{cv}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)}$$

Cast Location

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}, 0)) \qquad \widehat{\sigma}_1 = \widehat{\sigma}_2\big[\widehat{l} \to \big(\widehat{\omega},\ \text{void},\ \widehat{n},\ \text{PermL(Freeable, void, public}, \widehat{n})\big)\big] \\ (\widehat{ty} = \widehat{bty*}) \qquad \widehat{\sigma}_3 = \widehat{\sigma}_2\Big[\widehat{l} \to \Big(\widehat{\omega},\ \widehat{ty},\ \dfrac{\widehat{n}}{\tau(\widehat{ty})},\ \text{PermL}\Big(\text{Freeable}, \widehat{ty}, \text{public}, \dfrac{\widehat{n}}{\tau(\widehat{ty})}\Big)\Big)\Big]\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e}) \Downarrow'_{\widehat{cl}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ (\widehat{l}, 0))}$$

Malloc

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}) \qquad \widehat{l} = \phi() \\ \widehat{\sigma}_2 = \widehat{\sigma}_1\big[\widehat{l} \to \big(\text{NULL},\ \text{void}*,\ \widehat{n},\ \text{PermL(Freeable, void}*, \text{public}, \widehat{n})\big)\big]\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{malloc}(\widehat{e})) \Downarrow'_{\widehat{mal}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ (\widehat{l}, 0))}$$

Free

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x}) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*}) \qquad \text{Free}(\widehat{\gamma}, \widehat{l}, \widehat{\sigma}_1) = \widehat{\sigma}_2\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{free}(\widehat{e})) \Downarrow'_{\widehat{fre}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})}$$

Input Value

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x}) \\ (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}) \qquad \text{InputValue}(\widehat{x},\ \widehat{n}) = \widehat{n}_1 \\ (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{x} = \widehat{n}_1) \Downarrow'_{\widehat{s}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2)) \Downarrow'_{\widehat{inp}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})}$$

Output Value

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x}) \\ (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}) \qquad \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}) \\ \widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega},\ \widehat{bty},\ 1,\ \text{PermL(Freeable}, \widehat{bty}, \text{public}, 1)) \\ \text{DecodeVal}(\widehat{bty},\ 1,\ \widehat{\omega}) = \widehat{v} \qquad \text{OutputValue}(\widehat{x},\ \widehat{n},\ \widehat{v})\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcoutput}(\widehat{e}_1,\ \widehat{e}_2)) \Downarrow'_{\widehat{out}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})}$$

Input Array

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x}) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}) \qquad (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const}\ \widehat{bty*}) \qquad \text{InputArray}(\widehat{x}, \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}] \qquad (\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \Downarrow'_{\widehat{s}} (\widehat{\gamma}, \widehat{\sigma}_4, \square, \text{skip})\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3)) \Downarrow'_{\widehat{inp1}} (\widehat{\gamma},\ \widehat{\sigma}_4,\ \square,\ \text{skip})}$$

Output Array

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x}) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}) \qquad (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \text{const}\ \widehat{bty*}) \qquad \widehat{\sigma}_3(\widehat{l}) = (\widehat{\omega},\ \text{const}\ \widehat{bty*},\ 1,\ \text{PermL(Freeable, const}\ \widehat{bty*}, \text{public}, 1)) \\ \text{DecodePtr(const}\ \widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)],\ [1],\ 1] \qquad \widehat{\sigma}_3(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n}_2,\ \text{PermL(Freeable}, \widehat{bty}, \text{public}, \widehat{n}_1)) \\ \text{DecodeVal}(\widehat{bty},\ \widehat{n}_1,\ \widehat{\omega}_1) = [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}] \qquad \text{OutputArray}(\widehat{x},\ \widehat{n},\ [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}])\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcoutput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3)) \Downarrow'_{\widehat{out1}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})}$$

Figure 3.6: Vanilla C semantic rules for functions, casting, memory management, and input / output.

In semantic rule Function Declaration, we need are adding the function to the environment, but leaving it to be fully defined until a later point in the program. We first obtain a new memory block identifier from ϕ and use the Algorithm 22 ($\mathrm{GetFunTypeList}$) to create the type list $\widehat{\overline{ty}}$ from the given parameters. The function type is then the parameter type list to the return type, or $\widehat{\overline{ty}} \rightarrow \widehat{ty}$. We then add mappings to the environment and memory for the function, initializing the function data as NULL. We return the updated environment and memory.

In semantic rule Function Definition, we need to add a new function name and definition into our environment and memory. We first assert that this function was not previously declared, or that function name $\widehat{x}$ is not in environment $\widehat{\gamma}$. We then proceed to get a new memory block identifier, find the parameter type list, and add a new mapping for this function into the environment, as in rule Function Declaration. Here, however, we have the function body $\widehat{s}$, and use Algorithm 11 ($\mathrm{EncodeFun}$) to encode the function body and parameter list into the corresponding byte representation $\widehat{\omega}$, and then add a new mapping for this function into memory. The updated memory and environment are returned.

In semantic rule Pre-declared Function Definition, we need to add the definition of a pre-declared function into memory. We first assert that the function was predeclared by showing that function name $\widehat{x}$ is in the environment. Next, we use Algorithm 11 to get the byte representation $\widehat{\omega}$ of the function data. As we need to discard the mapping with the NULL value for the function that was stored in memory during the function declaration, we remove that mapping to obtain memory $\widehat{\sigma}_1$, then add the new mapping with the function data to $\widehat{\sigma}_1$ to obtain the final memory $\widehat{\sigma}_2$. We return $\widehat{\sigma}_2$.

In semantic rule Function Call, we perform the function call. We look up the function name in the environment, then look up the function data in memory, and use Algorithm 12 ($\mathrm{DecodeFun}$) to obtain the function body $\widehat{s}$ and parameter list for the function. We then use Algorithm 23 ($\mathrm{GetFunParamAssign}$) to create the statement $\widehat{s}_1$ to add the parameter variables to the environment and evaluate the expressions and assign their values to the corresponding parameters. Once we have evaluated these assignment statements, we continue on to evaluate the body of the function, and finally, return the updated memory and NULL, as our subset of C semantics does not currently include a return statement. It is possible to use pointers to store return values and thus including return statements is a trivial extension. We do not return any modifications to the environment, as they are out of scope beyond the evaluation of the function.

In semantic rule Cast Value, we are type-casting a number $\widehat{n}$ from one type to another (e.g., casting from float to int). We first evaluate the expression to obtain a number $\widehat{n}$, then use Algorithm 5 (Cast) to obtain the

corresponding number $\widehat{n}_1$ for the new type. We return the updated memory from the expression evaluation and value $\widehat{n}_1$. In semantic rule Cast Location, we are type-casting location, such as one obtained through the use of `malloc`. We make the assertion that the given type is a pointer type, and evaluate the expression to be a location $(\widehat{l}, 0)$. Here, we only allow locations with offset 0, as we are type-casting the memory block associated with this location and thus must not be at a random point inside a memory block. We then extract the mapping for the location we are updating from memory $\widehat{\sigma}_1$ to obtain $\widehat{\sigma}_2$, asserting that the type of the memory block was void. Finally, we create a new mapping for this location with the given type and updated size (as previously the size was stored as the number of bytes that the memory block contained and we now want to store the number of elements of that given type that can fit within the memory block). We add this mapping to $\widehat{\sigma}_2$ to obtain $\widehat{\sigma}_3$, and return $\widehat{\sigma}_3$ and the location.

In semantic rule Malloc, we allocate a new memory block based on the given size. First, we evaluate the expression to the size $\widehat{n}$, and obtain a new memory block from ϕ. Then we create a new mapping for this memory block, giving the size as $\widehat{n}$ bytes and type as void, and add it to memory $\widehat{\sigma}_1$ to obtain $\widehat{\sigma}_2$. We return $\widehat{\sigma}_2$ and the new location, $(\widehat{l}, 0)$. In semantic rule Free, we are deallocating a memory block previously allocated by `malloc`. We first evaluate the expression to be a variable $\widehat{x}$. We look up $\widehat{x}$ in the environment, finding the location where $\widehat{x}$ is stored and asserting that $\widehat{x}$ must be of a pointer type. We then use Algorithm 26 to check that the location that the pointer refers to is indeed freeable (i.e., was allocated by `malloc`), and if so, change the permissions associated that memory block to be None. As discussed in Subsection Malloc and Free, we do not clear the data stored in the location, nor do we return the location to the pool of available locations accessed by ϕ.

In semantic rule Input Value, we are reading in the value for a specific variable from a file and storing it in memory for that variable. First, we evaluate the first expression to be variable $\widehat{x}$ and the second expression to be the party $\widehat{n}$ which is contributing the input data. We look up the variable $\widehat{x}$ in the environment and assert that it is a regular (non-array, non-pointer) variable (i.e., type $\widehat{bty}$), since this function is reading in a single value. We then use Algorithm 28 ($\mathrm{InputValue}$) to obtain the value for that variable from the file containing that input party's input data. Finally, we evaluate the assignment to store the value we read from the file and we return the updated memory $\widehat{\sigma}_3$.

In semantic rule Input Array, we are reading in the values for an entire array from a file and storing it in memory for the given array variable. First, we evaluate the first expression to be variable $\widehat{x}$ and the second expression to be the party $\widehat{n}$ which is contributing the input data. Next, we evaluate the third expression to be

the length of the array, $\widehat{n}_1$. We look up the variable $\widehat{x}$ in the environment and assert that it is an array variable (i.e., type $\mathrm{const}\ \widehat{bty}*$), since this function is reading in $\widehat{n}_1$ values. We then use Algorithm 29 ($\mathrm{InputArray}$) to obtain the $\widehat{n}_1$ values for that variable from the file containing that input party's input data. Finally, we evaluate the assignment to store the values we read from the file and we return the updated memory $\widehat{\sigma}_4$.

In semantic rule Output Value, we are writing the value of a variable to a file as output. First, we evaluate the first expression to be variable $\widehat{x}$ and the second expression to be the party $\widehat{n}$ which is receiving the output data. We look up the variable $\widehat{x}$ in the environment and assert that it is a regular (non-array, non-pointer) variable (i.e., type $\widehat{bty}$), since this function is giving a single value as output. Next, we look up the data for $\widehat{x}$ in memory and obtain the value $\widehat{v}$ stored using Algorithm 8 ($\mathrm{DecodeVal}$). We then use Algorithm 30 ($\mathrm{OutputValue}$) to output $\widehat{v}$ to output party $\widehat{n}$.

In semantic rule Output Array, we are writing the values of an entire array to a file as output. First, we evaluate the first expression to be variable $\widehat{x}$ and the second expression to be the party $\widehat{n}$ which is receiving the output data. Next, we evaluate the third expression to be the length of the array, $\widehat{n}_1$. We look up the variable $\widehat{x}$ in the environment and assert that it is an array variable (i.e., type $\mathrm{const}\ \widehat{bty}*$), since this function is giving $\widehat{n}_1$ values as output. Next, we look up the data for $\widehat{x}$ in memory and obtain the list of values stored using Algorithm 8 ($\mathrm{DecodeVal}$). We then use Algorithm 31 ($\mathrm{OutputArray}$) to output the list of values to output party $\widehat{n}$.

In semantic rule If Else True, we need to evaluate the **`then`** branch, as the condition was true. We first evaluate the condition to be some number $\widehat{n}$. Then, we assert that $\widehat{n}$ cannot be 0, as this is the true case. We proceed to evaluate the **`then`** statement $\widehat{s}_1$, and return updated memory $\widehat{\sigma}_2$. Modifications to the environment are discarded as they are out of scope beyond the evaluation of this rule. In semantic rule If Else False, we need to evaluate the **`else`** branch, as the condition was false. We first evaluate the condition to be some number $\widehat{n}$. Then, we assert that $\widehat{n}$ must be 0, as this is the false case. We proceed to evaluate the **`else`** statement $\widehat{s}_2$, and return updated memory $\widehat{\sigma}_2$. Modifications to the environment are discarded as they are out of scope beyond the evaluation of this rule.

In semantic rule Pointer Declaration, we need to add mappings for the pointer to the environment and to memory. We first assert that the given type in a pointer type (i.e., $\widehat{bty}*$). We then obtain the level of indirection $\widehat{i}$ using Algorithm 6 ($\mathrm{GetIndirection}$), which will count the number of $*$ used in the type to obtain the level of indirection of this pointer. A new memory block identifier is obtained from the pool of available locations ϕ. Next, we add a mapping for this variable to the environment. We will store the a default

If Else True

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \hat{n}) \qquad \hat{n} \neq 0 \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{s}_1) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \Box,\ \text{skip})}{(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \Downarrow'_{\widehat{iet}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ \text{skip})}$$

If Else False

$$\frac{(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \hat{n}) \qquad \hat{n} = 0 \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{s}_2) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \Box,\ \text{skip})}{(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \Downarrow'_{\widehat{ief}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ \text{skip})}$$

Pointer Declaration

$$\frac{(\widehat{ty} = \widehat{bty*}) \qquad \text{GetIndirection}(*) = \widehat{i} \qquad \widehat{l} = \phi() \qquad \widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{ty})] \qquad \widehat{\omega} = \text{EncodePtr}(\widehat{ty*}, [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}]) \qquad \widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{ty},\ 0,\ \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, 0))]}{(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{ty}\ \widehat{x}) \Downarrow'_{\widehat{dp}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \Box,\ \text{skip})}$$

Pointer Read Location

$$\frac{\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*}) \qquad \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \qquad \text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]}{(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{x}) \Downarrow'_{\widehat{rp}} (\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ (\widehat{l}_1, \widehat{\mu}_1))}$$

Pointer Assign Location

$$\frac{(\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, (\widehat{l}_e, \widehat{\mu}_e)) \qquad \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*}) \qquad \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \qquad \text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}] \qquad \text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}, 0), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}], \widehat{bty*}) = (\widehat{\sigma}_2, 1)}{(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wp}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ \text{skip})}$$

Pre-increment Pointer

$$\frac{\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*}) \qquad \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \qquad \text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ 1] \qquad ((\widehat{l}_2, \widehat{\mu}_2), 1) = \text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty}), \widehat{\sigma}) \qquad \text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_2, \widehat{\mu}_2)],\ [1],\ 1], \widehat{bty*}) = (\widehat{\sigma}_1, 1)}{(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ ++\ \widehat{x}) \Downarrow'_{\widehat{pin2}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ (\widehat{l}_2, \widehat{\mu}_2))}$$

Pre-increment Pointer (Not Aligned)

$$\frac{\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*}) \qquad \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \qquad \text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ 1] \qquad ((\widehat{l}_2, \widehat{\mu}_2), 0) = \text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty}), \widehat{\sigma}) \qquad \text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_2, \widehat{\mu}_2)],\ [1],\ 1], \widehat{bty*}) = (\widehat{\sigma}_1, 1)}{(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ ++\ \widehat{x}) \Downarrow'^{*}_{\widehat{pin2}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ (\widehat{l}_2, \widehat{\mu}_2))}$$

Pre-increment Pointer Higher Level Indirection

$$\frac{\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*}) \qquad \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \qquad \text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ \widehat{i}] \qquad \widehat{i} > 1 \qquad ((\widehat{l}_2, \widehat{\mu}_2), 1) = \text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty*}), \widehat{\sigma}) \qquad \text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_2, \widehat{\mu}_2)],\ [1],\ i], \widehat{bty*}) = (\widehat{\sigma}_1, 1)}{(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ ++\ \widehat{x}) \Downarrow'_{\widehat{pin3}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ (\widehat{l}_2, \widehat{\mu}_2))}$$

Pre-increment Pointer Higher Level Indirection (Not Aligned)

$$\frac{\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty*}) \qquad \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \qquad \text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ i] \qquad i > 1 \qquad ((\widehat{l}_2, \widehat{\mu}_2), 0) = \text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty*}), \widehat{\sigma}) \qquad \text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_2, \widehat{\mu}_2)],\ [1],\ i], \widehat{bty*}) = (\widehat{\sigma}_1, 1)}{(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ ++\ \widehat{x}) \Downarrow'^{*}_{\widehat{pin3}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ (\widehat{l}_2, \widehat{\mu}_2))}$$

Figure 3.7: Vanilla C semantic rules for branches and pointer declarations, reading, writing, and incrementing.

pointer data structure (i.e., our representation of a NULL or uninitialized pointer) as $[1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}]$, indicating that is refers to a single location, which is the default location $(\widehat{l}_{default}, 0)$, the tag indicating that it is the true location, and the pointer has the level of indirection $\widehat{i}$. We create the byte representation for the pointer with the default data structure for the pointer using Algorithm 9, and then add the mapping for the pointer data into memory. We return the updated environment and memory. Further details about the pointer data structure can be found in Subsection 3.1.2).

In semantic rule Pointer Read Location, we are reading which location that the pointer refers to. Here, we first look up the pointer in the environment, then in memory. We use Algorithm 10 ($\mathrm{DecodePtr}$) to obtain the pointer data structure from the byte representation, and then return the location referred to by the pointer. In semantic rule Pointer Assign Location, we are assigning a new location for the pointer to refer to. We first evaluate the expression to obtain the location that we will be assigning to the pointer, then we look up the pointer variable in the environment and its corresponding memory block in memory. We use Algorithm 10 ($\mathrm{DecodePtr}$) to obtain the pointer data structure from the byte representation in order to obtain the level of indirection of the pointer. Finally, we use Algorithm 14 ($\mathrm{UpdatePtr}$) to create the new mapping for the pointer in memory and swap it in place of the old mapping, returning the updated memory with the new mapping and the tag 1. This tag indicates that the location updated in memory was *well-aligned*, as is expected because we are updating the pointer's own memory block with the expected value. This algorithm is called with the memory obtained from the evaluation of the expression, the location of the pointer in memory, the updated pointer data structure, and the type of the pointer. We return the final updated memory $\widehat{\sigma}_2$ from this rule.

In semantic rule Pre-increment Pointer, we are incrementing the location that the pointer refers to. We first look up the pointer in the environment, then in memory, and then obtain the pointer data structure from the byte representation using Algorithm 10 ($\mathrm{DecodePtr}$). We use Algorithm 21 ($\mathrm{GetLocation}$) to obtain the next location beyond the current location that the pointer refers to. We pass this algorithm the current referred to location, then the size of the type of data it refers to (i.e., a non-pointer type, as the level of indirection is 1), and the current memory; we assert that it returned to us the next location, and the tag 1, indicating that that location was *well-aligned* (i.e., it is of the same type and immediately after the previous location). We then use Algorithm 14 ($\mathrm{UpdatePtr}$) to store this new location as the location being referred to by the pointer, as discussed above for rule Pointer Assign Location. This rule then returns the updated memory and the new location that is being referred to by this pointer. Semantic rule Pre-increment Pointer (Not Aligned) is nearly

identical to to rule Pre-increment Pointer – the only difference is that the location returned by Algorithm 21 ($\mathrm{GetLocation}$) is not *well-aligned*, as discussed in Subsection 3.1.2.

In semantic rule Pre-increment Pointer Higher Level Indirection, we are incrementing the location that the pointer refers to, just for a pointer with a higher level of indirection. The difference between this rule and Pre-increment Pointer is that the size we use when obtaining the new location from Algorithm 21 ($\mathrm{GetLocation}$) is determined by the size of a pointer $\tau(\widehat{bty}*)$ rather than the size of a regular variable $\tau(\widehat{bty})$. To enforce this as being a pointer of a higher level of indirection, we add in the assertion that the level of indirection $\widehat{i}$ must be greater than one. Semantic rule Pre-increment Pointer Higher Level Indirection (Not Aligned) is nearly identical to to rule Pre-increment Pointer Higher Level Indirection – the only difference is that the location returned by Algorithm 21 ($\mathrm{GetLocation}$) is not *well-aligned*, as discussed in Subsection 3.1.2.

In semantic rule Pointer Dereference Write Value, we are dereferencing the pointer and writing a value to the location that the pointer refers to. We first evaluate the expression to a value, and make the assertion that this value is not the terminal skip, as that is not a valid value to store in memory. Then we look up the variable in the environment, making the assertion that it is a pointer variable, and proceed to look up the data in memory and obtain the pointer data structure with Algorithm 10 ($\mathrm{DecodePtr}$), asserting that this pointer has level of indirection 1. We use Algorithm 15 ($\mathrm{UpdateOffset}$) to write the given value to the location in memory referred to by the pointer. This algorithm returns tag 1, indicating that the value was written into a location that it was *well-aligned* with. We then return the updated memory. In semantic rule Pointer Dereference Write Value (Not Aligned) is nearly identical to rule Pointer Dereference Write Value, the only exception being Algorithm 15 ($\mathrm{UpdateOffset}$) returning 0 to indicate that the value written into memory was not *well-aligned* with the location it was written to (e.g., we have overwritten chunks of two different memory blocks and introduced garbage into the memory).

In semantic rule Pointer Dereference Write Value Higher Level Indirection, we are dereferencing the pointer and writing a new location to the location that the pointer refers to. First, we evaluate the expression to be a location. Then we look up the variable $\widehat{x}$ in the environment and in memory, asserting that it is a pointer type. We obtain the pointer data structure with Algorithm 10 ($\mathrm{DecodePtr}$), and assert that the level of indirection of this pointer is greater than 1, or that this is a pointer to a pointer. We then use Algorithm 14 ($\mathrm{UpdatePtr}$) to update the location that the lower level referred to pointer is now referring to. This algorithm returns the updated memory and the tag 1, indicating that our update to memory was *well-aligned*. In semantic rule Pointer Dereference Write Value Higher Level Indirection (Not Aligned) is nearly identical to rule

Pointer Dereference Write Value

$$\frac{\begin{array}{lll}(\widehat{\gamma},\widehat{\sigma},\square,\widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\widehat{\sigma}_1,\square,\widehat{v}) & \widehat{\gamma}(x) = (\widehat{l}, \widehat{bty*}) & \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty*}, \mathrm{public}, 1)) \\ \widehat{v} \neq \mathrm{skip} & \mathrm{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ 1] & \mathrm{UpdateOffset}(\widehat{\sigma}_1,\ (\widehat{l}_1, \widehat{\mu}_1),\ \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, 1)\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *x =\ \widehat{e}) \Downarrow'_{\widehat{wdp}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \mathrm{skip})}$$

Pointer Dereference Write Value (Not Aligned)

$$\frac{\begin{array}{lll}(\widehat{\gamma},\widehat{\sigma},\square,\widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\widehat{\sigma}_1,\square,\widehat{v}) & \widehat{\gamma}(x) = (\widehat{l}, \widehat{bty*}) & \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty*}, \mathrm{public}, 1)) \\ \widehat{v} \neq \mathrm{skip} & \mathrm{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ 1] & \mathrm{UpdateOffset}(\widehat{\sigma}_1,\ (\widehat{l}_1, \widehat{\mu}_1),\ \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, 0)\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *x =\ \widehat{e}) \Downarrow'^{*}_{\widehat{wdp}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \mathrm{skip})}$$

Pointer Dereference Write Value Higher Level Indirection

$$\frac{\begin{array}{ll}(\widehat{\gamma},\widehat{\sigma},\square,\widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\widehat{\sigma}_1,\square,(\widehat{l}_e, \widehat{\mu}_e)) & \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*}) \\ \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty*}, \mathrm{public}, 1)) & \mathrm{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}] \\ \widehat{i} > 1 & \mathrm{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1], \widehat{bty*}) = (\widehat{\sigma}_2, 1)\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} =\ \widehat{e}) \Downarrow'_{\widehat{wdp1}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \mathrm{skip})}$$

Pointer Dereference Write Value Higher Level Indirection (Not Aligned)

$$\frac{\begin{array}{ll}(\widehat{\gamma},\widehat{\sigma},\square,\widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\widehat{\sigma}_1,\square,(\widehat{l}_e, \widehat{\mu}_e)) & \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*}) \\ \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty*}, \mathrm{public}, 1)) & \mathrm{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}] \\ \widehat{i} > 1 & \mathrm{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1], \widehat{bty*}) = (\widehat{\sigma}_2, 0)\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} =\ \widehat{e}) \Downarrow'^{*}_{\widehat{wdp1}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \mathrm{skip})}$$

Pointer Dereference

$$\frac{\begin{array}{ll}\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*}) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty*}, \mathrm{public}, 1)) \\ \mathrm{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ 1] & \mathrm{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{v}, 1)\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'_{\widehat{rdp}} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})}$$

Pointer Dereference (Not Aligned)

$$\frac{\begin{array}{ll}\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*}) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty*}, \mathrm{public}, 1)) \\ \mathrm{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ 1] & \mathrm{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{v}, 0)\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'^{*}_{\widehat{rdp}} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})}$$

Pointer Dereference Higher Level Indirection

$$\frac{\begin{array}{lll}\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*}) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty*}, \mathrm{public}, 1)) & \widehat{i} > 1 \\ \mathrm{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}] & \mathrm{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty*}, (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1], 1) & \end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'_{\widehat{rdp1}} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}_2, \widehat{\mu}_2))}$$

Pointer Dereference Higher Level Indirection (Not Aligned)

$$\frac{\begin{array}{lll}\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*}) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty*}, \mathrm{public}, 1)) & \widehat{i} > 1 \\ \mathrm{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}] & \mathrm{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty*}, (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1], 0) & \end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'^{*}_{\widehat{rdp1}} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}_2, \widehat{\mu}_2))}$$

Figure 3.8: Vanilla C semantic rules for pointer dereference write and dereference read.

Pointer Dereference Write Value Higher Level Indirection, with the difference being that the location we are storing into memory was stored into a location that was not *well-aligned*.

In semantic rule Pointer Dereference, we are dereferencing the pointer to read the value stored at the location the pointer refers to. First, we look up the variable in memory and the environment, asserting that it is a pointer type. We obtain the pointer data structure with Algorithm 10 ($\mathrm{DecodePtr}$) and assert that have level of indirection 1. Finally, we use Algorithm 16 ($\mathrm{DerefPtr}$) to read a value of type bty from memory at the referred to location $(\widehat{l}_1, \widehat{\mu}_1)$. We assert that this algorithm also returns the tag 1, indicating that the location that we read from in memory was *well-aligned* and therefore of the expected type. We then return the value we read from memory. In semantic rule Pointer Dereference (Not Aligned) is nearly identical to rule Pointer Dereference, with the difference being that the value read by Algorithm 16 ($\mathrm{DerefPtr}$) was not from a *well-aligned* location (i.e., we have potentially just read a garbage value).

In semantic rule Pointer Dereference Higher Level Indirection, we are dereferencing the pointer to read the value stored at the location the pointer refers to. The main difference between this rule and rule Pointer Dereference is that the value we will read is going to be a location instead of a number $\widehat{n}$. First, we look up the variable in memory and the environment, asserting that it is a pointer type. We obtain the pointer data structure with Algorithm 10 ($\mathrm{DecodePtr}$) and assert that have level of indirection greater than 1. We then use Algorithm 17 ($\mathrm{DerefPtrHLI}$) to read the pointer data structure from memory at the referred to location $(\widehat{l}_1, \widehat{\mu}_1)$. We assert that this algorithm also returns the tag 1, indicating that the location that we read from in memory was *well-aligned* and therefore of the expected type. We then return the location we read from memory. In semantic rule Pointer Dereference Higher Level Indirection (Not Aligned) is nearly identical to rule Pointer Dereference Higher Level Indirection, with the difference being that the pointer data structure read by Algorithm 17 ($\mathrm{DerefPtrHLI}$) was not from a *well-aligned* location (i.e., we have potentially just read a garbage value).

In semantic rule Array Declaration, we are adding the new array to the environment and memory. We first evaluate the expression to obtain the size of the array, $\widehat{n}$. Next, we obtain two new memory block identifiers from ϕ, as arrays in C are stored as a constant pointer to the array data. We will use $\widehat{l}$ for the array pointer and $\widehat{l}_1$ for the array data. We create the byte representation $\widehat{\omega}$ of the pointer data structure for the array pointer using Algorithm 9 ($\mathrm{EncodePtr}$), storing the location for the array data. We add the mapping for the array variable to memory with the location for the array pointer, then add the array pointer location to memory with its corresponding data. Then we create initial NULL data to store for the array using Algorithm 7

Array Declaration

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}) \qquad \widehat{l} = \phi() \qquad \widehat{l_1} = \phi() \qquad \widehat{\omega} = \text{EncodePtr}(\text{const } \widehat{bty}*,\ [1,\ [(\widehat{l_1}, 0)],\ [1],\ 1]) \\ \widehat{\gamma}_1 = \widehat{\gamma}[x\ \rightarrow\ (\widehat{l},\ \text{const } \widehat{bty}*)] \qquad \widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l}\ \rightarrow\ (\widehat{\omega},\ \text{const } \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))] \\ \text{EncodeVal}(\widehat{bty}, \text{NULL}) = \widehat{\omega}_1 \qquad \widehat{n} > 0 \qquad \widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l_1}\ \rightarrow\ (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n},\ \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))]\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{bty}\ x[\widehat{e}]) \Downarrow'_{\widehat{da}} (\widehat{\gamma}_1,\ \widehat{\sigma}_3,\ \square,\ \text{skip})}$$

Array Write

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i}) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v}) \qquad \widehat{v} \neq \text{skip} \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*) \qquad \widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1)) \\ \text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l_1}, 0)],\ [1],\ 1] \\ \widehat{\sigma}_2(\widehat{l_1}) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n})) \qquad \text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}] \\ 0 \leq \widehat{i} \leq \widehat{n} - 1 \qquad [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right) \qquad \text{UpdateVal}(\widehat{\sigma}_2,\ \widehat{l_1},\ [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{\widehat{wa}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})}$$

Array Read

$$\frac{\begin{array}{l}(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i}) \qquad \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1)) \\ \widehat{\gamma}(x) = (\widehat{l}, \text{const } \widehat{bty}*) \qquad 0 \leq \widehat{i} \leq \widehat{n} - 1 \qquad \text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l_1}, 0)], [1], 1] \\ \widehat{\sigma}_1(\widehat{l_1}) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n})) \qquad \text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}]) \Downarrow'_{\widehat{ra}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_{\widehat{i}})}$$

Array Read Out-of-bounds

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i}) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \text{const } \widehat{bty}*) \qquad \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1)) \\ \text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l_1}, 0)],\ [1],\ 1] \qquad \widehat{\sigma}_1(\widehat{l_1}) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n})) \\ (\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n}) \qquad \text{ReadOOB}(\widehat{i},\ \widehat{n},\ \widehat{l_1},\ \widehat{bty},\ \widehat{\sigma}_1) = (\widehat{v},\ 1)\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}]) \Downarrow'_{\widehat{rao}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})}$$

Array Read Out-of-bounds (Not Aligned)

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i}) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \text{const } \widehat{bty}*) \qquad \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1)) \\ \text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l_1}, 0)],\ [1],\ 1] \qquad \widehat{\sigma}_1(\widehat{l_1}) = (\widehat{\omega}_1,\ \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n})) \\ (\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n}) \qquad \text{ReadOOB}(\widehat{i},\ \widehat{n},\ \widehat{l_1},\ \widehat{bty},\ \widehat{\sigma}_1) = (\widehat{v},\ 0)\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}]) \Downarrow'^{*}_{\widehat{rao}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})}$$

Array Write Out-of-bounds

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i}) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v}) \qquad \widehat{v} \neq \text{skip} \\ \widehat{\gamma}(x) = (\widehat{l}, \text{const } \widehat{bty}*) \qquad \widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1)) \\ \text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l_1}, 0)], [1], 1] \qquad \widehat{\sigma}_2(\widehat{l_1}) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n})) \\ (\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n}) \qquad \text{WriteOOB}(\widehat{v},\ \widehat{i},\ \widehat{n},\ \widehat{l_1},\ \widehat{bty},\ \widehat{\sigma}_2) = (\widehat{\sigma}_3,\ 1)\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ =\ \widehat{e}_2) \Downarrow'_{\widehat{wao}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})}$$

Array Write Out-of-bounds (Not Aligned)

$$\frac{\begin{array}{l}(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i}) \qquad (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{e}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v}) \qquad \widehat{v} \neq \text{skip} \\ \widehat{\gamma}(x) = (\widehat{l}, \text{const } \widehat{bty}*) \qquad \widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1)) \\ \text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l_1}, 0)],\ [1],\ 1] \qquad \widehat{\sigma}_2(\widehat{l_1}) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n})) \\ (\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n}) \qquad \text{WriteOOB}(\widehat{v},\ \widehat{i},\ \widehat{n},\ \widehat{l_1},\ \widehat{bty},\ \widehat{\sigma}_2) = (\widehat{\sigma}_3,\ 0)\end{array}}{(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ x[\widehat{e}_1]\ =\ \widehat{e}_2) \Downarrow'^{*}_{\widehat{wao}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})}$$

Figure 3.9: Vanilla C semantic rules for one-dimensional array declarations, reading, and writing.

(EncodeVal), and assert that $\widehat{n}$ is greater than 0, as the array must have at least one element. Finally, we add the mapping for the array data to memory and return the updated environment and memory.

In semantic rule Array Write, we are writing a value to an index of the array. First, we evaluate the first expression to find the index $\widehat{i}$ that we are writing to, then the second expression to find the value $\widehat{v}$ that we will be writing to that index. We assert that this value is not the terminal value skip, as it is not a valid value to write into memory. We then look up the variable in the environment, asserting that it is an array variable (i.e., const $\widehat{bty}*$). We look up the array pointer data in memory, use Algorithm 10 (DecodePtr) to get the pointer data structure, then look up the array data in memory. We use Algorithm 8 (DecodeVal) to obtain the array values from the byte representation. We assert that the index is within bounds (i.e., that it is between 0 and $\widehat{n} - 1$, as arrays are 0-indexed). We then swap the value at index $\widehat{i}$ for the new value $\widehat{v}$ that we are assigning to that index, obtaining the updated array data (i.e., $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}]$). We the use Algorithm 13 (UpdateVal) to place the updated array data into memory for array $\widehat{x}$ and return the updated memory.

In semantic rule Array Read, we are reading a value from an index of the array. First, we evaluate the expression to find the index $\widehat{i}$ that we are reading from. We then look up the variable in the environment, asserting that it is an array variable (i.e., const $\widehat{bty}*$). We look up the array pointer data in memory, use Algorithm 10 (DecodePtr) to get the pointer data structure, then look up the array data in memory. Finally, we use Algorithm 8 (DecodeVal) to obtain the array values from the byte representation, and return the value at index $\widehat{i}$ and the updated memory returned from the evaluation of the expression.

In semantic rule Array Read Out-of-bounds, we are reading a value from an index that is beyond the bounds of the array. This rule differs from rule Array Read after looking up the array data in memory. We then assert that the index $\widehat{i}$ was either less than 0 or greater than or equal to the length of the array (i.e., that this index is beyond the bounds of the array, which ranges from index 0 to index $\widehat{n} - 1$). We then use Algorithm 24 (ReadOOB) to read a value from the place in memory where the index is anticipated to be (i.e., before the array if negative, otherwise after the array). This algorithm returns the byte data at that position interpreted as a value of the same type as the array and a tag, which we assert to be 1. This tag indicates that the location we read the value from (as well as any spaces between) is of the same type as the array, and therefore is likely a valid value. We return this value and the updated memory returned from the evaluation of the expression. Semantic rule Array Read Out-of-bounds (Not Aligned) differs from rule Array Read Out-of-bounds in that the value being read is from a location that is not *well-aligned* with the array, as discussed in Subsection 3.1.2.

Array Declaration Assignment

$$\frac{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}[\widehat{e}]) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip}) \qquad (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{s}} (\widehat{\gamma}_1, \widehat{\sigma}_2, \square, \text{skip})}{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}[\widehat{e}] = \widehat{e}) \Downarrow'_{\widehat{das}} (\widehat{\gamma}_1, \widehat{\sigma}_2, \square, \text{skip})}$$

Array Read Entire Array

$$\frac{\begin{array}{ll} \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1)) \\ \text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1] & \\ \widehat{\sigma}(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n})) & \text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}] \end{array}}{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{ra4}} (\widehat{\gamma}, \widehat{\sigma}, \square, [\widehat{v}_0, ..., \widehat{v}_{n-1}])}$$

Array Write Entire Array

$$\frac{\begin{array}{ll} (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{e}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e - 1}]) & \forall \widehat{v}_m \in [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e - 1}].\quad \widehat{v}_m \neq \text{skip} \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*) & \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1)) \\ \text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1] & \widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n})) \\ \widehat{n}_e = \widehat{n} & \text{UpdateVal}(\widehat{\sigma}_1, \widehat{l}_1, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e - 1}], \widehat{bty}) = \widehat{\sigma}_2 \end{array}}{(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wa5}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{skip})}$$

Figure 3.10: Vanilla C semantic rules for managing an entire array.

In semantic rule Array Write Out-of-bounds, we are writing a value to an index that is beyond the bounds of the array. This rule differs from rule Array Write after looking up the array data in memory. We then assert that the index $\widehat{i}$ was either less than 0 or greater than or equal to the length of the array (i.e., that this index is beyond the bounds of the array, which ranges from index 0 to index $\widehat{n} - 1$). We then use Algorithm 25 (WriteOOB) to write a value from the place in memory where the index is anticipated to be (i.e., before the array if negative, otherwise after the array). This algorithm returns the updated memory and a tag, which we assert to be 1. This tag indicates that the location we wrote the value to (as well as any spaces between) is of the same type as the array, and therefore is we did not introduce garbage into memory. We return the updated memory. Semantic rule Array Write Out-of-bounds (Not Aligned) differs from rule Array Write Out-of-bounds in that the value being written to memory is at a location that is not *well-aligned* with the array, as discussed in Subsection 3.1.2.

In semantic rule Array Declaration Assignment, we are facilitating the declaration and initialization of a new array variable with a value. We first evaluate the declaration, then the assignment of the initial values for the entire array, returning the additions and updates to the environment and memory. In semantic rule Array Read Entire Array, we are reading every value from an array. We first look up the variable in the environment, asserting that it is an array variable (i.e., $\text{const } \widehat{bty}*$). We look up the array pointer data in memory, use Algorithm 10 (DecodePtr) to get the pointer data structure, then look up the array data in memory. We then use Algorithm 8 (DecodeVal) to obtain the array values from the byte representation, which we return.

In semantic rule Array Write Entire Array, we are overwriting every value for an array. We first evaluate

the expression to be a list of values, and assert that none of these values is the terminal value skip, as that is not a valid value to add into memory. We then look up the variable in the environment, asserting that it is an array variable (i.e., $\text{const } \widehat{bty}*$). We look up the array pointer data in memory, use Algorithm 10 (DecodePtr) to get the pointer data structure, then look up the array data in memory. We assert that the length of the array from the expression $\widehat{n}_e$ is equivalent to the length of the array we are assigning it to, and use Algorithm 13 (UpdateVal) to write the values from the expression into memory for this array. Finally, we return the updated memory.

3.1.4 Basic SMC^2

In this section, we show the Basic SMC^2 semantics with respect to the grammar (Figure 3.1). The semantic judgements in Basic SMC^2 are defined over a four-tuple configuration $C = (\gamma, \sigma, \text{acc}, s)$, where each rule is a reduction from one configuration to a subsequent. We denote the environment as γ; memory as σ; the level of nesting of private-conditioned branches as acc; and a big-step evaluation of a statement s to a value v using $\Downarrow$. We annotate each evaluation with evaluation codes (i.e., $\Downarrow_d$) to facilitate reasoning over evaluation trees, and we annotate evaluations that are not *well-aligned* with a star (i.e., $\Downarrow_d^*$) to identify the rules that we cannot prove correctness over, as they produce unpredictable behavior. The assertions in each semantic rule can be read in sequential order, from left to right and top to bottom.

Figure 3.11 gives the Basic SMC^2 semantics for basic declarations, reading, and writing, sequencing, loops, finding the size of a type, and finding the address of a variable. Figure 3.12 gives the Basic SMC^2 semantics for addition, subtraction, multiplication, and division. Figures 3.13 and 3.14 give the semantics for not equal to and equal to comparison operations and for less than comparison operations, respectively. Figure 3.15 gives the semantics for Basic SMC^2 public and private **`if else`** operations. Figures 3.16 and 3.17 give the semantics for pre-increment operator (`++x`). Figure 3.18 gives the semantics for functions and casting. Figure 3.19 gives the semantics for the input and output of data from files. Figure 3.20 gives the semantics for memory allocation and deallocation. Figure 3.21 gives the semantics for pointer declarations, reads, and writes. Figures 3.22 and 3.23 give the semantics for pointer dereference writes, and Figure 3.24 gives those for pointer dereference reads. Figure 3.25 gives the semantics for array declarations and writing an entire array. Figures 3.26 and 3.27 give the semantics for reading from an array and writing to an index of an array, respectively. Figures 3.28 and 3.29 give the semantics for reading and writing out of the bounds of an array, respectively.

Public Declaration

$$\frac{(ty = \text{public } bty) \quad \text{acc} = 0 \quad l = \phi() \quad \gamma_1 = \gamma[x \rightarrow (l, ty)] \quad \omega = \text{EncodeVal}(ty, \text{NULL}) \quad \sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]}{(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_d (\gamma_1, \sigma_1, \text{acc}, \text{skip})}$$

Private Declaration

$$\frac{((ty = bty) \vee (ty = \text{private } bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float})) \quad l = \phi() \quad \gamma_1 = \gamma[x \rightarrow (l, \text{private } bty)] \quad \omega = \text{EncodeVal}(ty, \text{NULL}) \quad \sigma_1 = \sigma[l \rightarrow (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))]}{(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{d1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})}$$

Read Public Variable

$$\frac{\gamma(x) = (l, \text{public } bty) \quad \sigma(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1)) \quad \text{DecodeVal}(\text{public } bty, 1, \omega) = v}{(\gamma, \sigma, \text{acc}, x) \Downarrow_r (\gamma, \sigma, \text{acc}, v)}$$

Statement Sequencing

$$\frac{(\gamma, \sigma, \text{acc}, s_1) \Downarrow_s (\gamma_1, \sigma_1, \text{acc}, \text{skip}) \quad (\gamma_1, \sigma_1, \text{acc}, s_2) \Downarrow_s (\gamma_2, \sigma_2, \text{acc}, v)}{(\gamma, \sigma, \text{acc}, s_1; s_2) \Downarrow_{ss} (\gamma_2, \sigma_2, \text{acc}, v)}$$

Read Private Variable

$$\frac{\gamma(x) = (l, \text{private } bty) \quad \sigma(l) = (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) \quad \text{DecodeVal}(\text{private } bty, 1, \omega) = v}{(\gamma, \sigma, \text{acc}, x) \Downarrow_{r1} (\gamma, \sigma, \text{acc}, v)}$$

Declaration Assignment

$$\frac{(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_s (\gamma_1, \sigma_1, \text{acc}, \text{skip}) \quad (\gamma_1, \sigma_1, \text{acc}, x = e) \Downarrow_s (\gamma_1, \sigma_2, \text{acc}, \text{skip})}{(\gamma, \sigma, \text{acc}, ty\ x = e) \Downarrow_{ds} (\gamma_1, \sigma_2, \text{acc}, \text{skip})}$$

Write Public Variable

$$\frac{\text{Label}(e, \gamma) = \text{public} \quad (\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, v) \quad v \neq \text{skip} \quad \text{acc} = 0 \quad \gamma(x) = (l, \text{public } bty) \quad \text{UpdateVal}(\sigma_1, l, v, \text{public } bty) = \sigma_2}{(\gamma, \sigma, \text{acc}, x = e) \Downarrow_w (\gamma, \sigma_2, \text{acc}, \text{skip})}$$

Address Of

$$\frac{\gamma(x) = (l,\ ty)}{(\gamma,\ \sigma,\ \text{acc},\ \&x) \Downarrow_{loc} (\gamma,\ \sigma,\ \text{acc},\ (l, 0))}$$

Write Private Variable

$$\frac{\text{Label}(e, \gamma) = \text{private} \quad (\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, v) \quad v \neq \text{skip} \quad \gamma(x) = (l, \text{private } bty) \quad \text{UpdateVal}(\sigma_1, l, v, \text{private } bty) = \sigma_2}{(\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w2} (\gamma, \sigma_2, \text{acc}, \text{skip})}$$

While End

$$\frac{\text{Label}(e, \gamma) = \text{public} \quad (\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, n) \quad n = 0}{(\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow_{wle} (\gamma, \sigma_1, \text{acc}, \text{skip})}$$

Write Private Variable Public Value

$$\frac{\text{Label}(e, \gamma) = \text{public} \quad \gamma(x) = (l, \text{private } bty) \quad (\gamma,\ \sigma,\ \text{acc},\ e) \Downarrow_e (\gamma,\ \sigma_1,\ \text{acc},\ n) \quad \text{UpdateVal}(\sigma_1, l, \text{encrypt}(n), \text{private } bty) = \sigma_2}{(\gamma,\ \sigma,\ \text{acc},\ x = e) \Downarrow_{w1} (\gamma,\ \sigma_2,\ \text{acc},\ \text{skip})}$$

While Continue

$$\frac{\text{Label}(e, \gamma) = \text{public} \quad (\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, n) \quad n \neq 0 \quad (\gamma, \sigma_1, \text{acc}, s) \Downarrow_s (\gamma_1, \sigma_2, \text{acc}, \text{skip}) \quad (\gamma, \sigma_2, \text{acc}, \text{while } (e)\ s) \Downarrow_s (\gamma_2, \sigma_3, \text{acc}, \text{skip})}{(\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow_{wlc} (\gamma, \sigma_2, \text{acc}, \text{while } (e)\ s)}$$

Parentheses

$$\frac{(\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, v)}{(\gamma, \sigma, \text{acc}, (e)) \Downarrow_{ep} (\gamma, \sigma_1, \text{acc}, v)}$$

Statement Block

$$\frac{(\gamma, \sigma, \text{acc}, s) \Downarrow_s (\gamma_1, \sigma_1, \text{acc}, \text{skip})}{(\gamma, \sigma, \text{acc}, \{s\}) \Downarrow_{sb} (\gamma, \sigma_1, \text{acc}, \text{skip})}$$

Size of type

$$\frac{n = \tau(ty)}{(\gamma, \sigma, \text{acc}, \text{sizeof}(ty)) \Downarrow_{ty} (\gamma, \sigma, \text{acc}, n)}$$

Figure 3.11: Basic SMC^2 semantics for basic declarations, reading, writing, sequencing, loops, finding the size of a type, and finding the address of a variable.

The rules for Public Declaration and Private Declaration are similar - the main differences are in ensuring that the appropriate privacy label is assigned to the byte-wise permissions, and disallowing public declarations within branches on private data (i.e., the condition $\mathrm{acc} = 0$ defines that we are not within the scope of a private branch). We do not allow public declarations within an **`if else`** statement branching on private data (i.e., a private-conditioned branch) because we do not allow modifications to public variables within such branches, as such modifications are public side effects that would be viewable in memory and would lead to the leakage of information about the private data we used to evaluate the branch.

Rules Read Private Variable and Read Public Variable are both standard C reading rules, handling either private or public data. Rules Write Private Variable and Write Public Variable are standard C writing rules for storing a value for a variable, where both the variable and the value have the same privacy label. In rule Write Public Variable, there is the assertion that $\mathrm{acc} = 0$, meaning this rule cannot be evaluated when inside a private-conditioned branch. Rule Write Private Variable Public Value handles writing a public value to a private variable. Note, there is no rule for writing a private value to a public variable, as this is not a valid operation in our semantics (i.e., a program with such an operation would result in a compile time or runtime error depending on the specific implementation) because it would reveal (or 'open') the private value. We currently do not model the behavior for 'opening' private values within our semantics.

Rule Declaration Assignment is used to allow declaring a variable and providing and initial value for the variable (assignment) in one statement. The rules for Statement Sequencing, Statement Block, Parentheses, Address Of, and Size of Type are the same as those defined for our C semantics.

For while loops (and loops in general), we must perform a check to ensure that the expression we are looping on is public, as we can not hide the number of times the loop is executed. Rule While Continue uses the $\Rightarrow$ evaluation as it will be evaluated an arbitrary number of times until the condition becomes false and rule While End is evaluated. These rule mirrors what is present in our definition of the C semantics, modified only to do the appropriate runtime assertion that conditional expressions are public.

For completeness, the semantics of binary operations within the scope of the grammar (Figure 3.1) are shown in Figures 3.12, 3.13, and 3.14. The rule Public Addition operates over two public expressions, performing regular addition. For Private Addition, we have two expressions with private data, and we use $+_{\mathrm{private}}$ to abstract the precise implementation of secure addition over two private values. In rules Public-Private Addition2 and Private-Public Addition, we encrypt the public expression and then perform secure addition over the private value and the newly encrypted value. All cases of rules for Subtraction,

Public Addition

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public}\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ n_1 +_{\text{public}} n_2 = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1+e_2) \Downarrow_{bp} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private - Public Addition

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public})\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ n_1 +_{\text{private}} \text{encrypt}(n_2) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1+e_2) \Downarrow_{bp3} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private Addition

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private}\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ n_1 +_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1+e_2) \Downarrow_{bp1} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public - Private Addition

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private})\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ \text{encrypt}(n_1) +_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1+e_2) \Downarrow_{bp2} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public Subtraction

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public}\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ n_1 -_{\text{public}} n_2 = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1-e_2) \Downarrow_{bs} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private - Public Subtraction

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public})\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ n_1 -_{\text{private}} \text{encrypt}(n_2) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1-e_2) \Downarrow_{bs3} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private Subtraction

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private}\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ n_1 -_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1-e_2) \Downarrow_{bs1} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public - Private Subtraction

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private})\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ \text{encrypt}(n_1) -_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1-e_2) \Downarrow_{bs2} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public Multiplication

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public}\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ n_1 \cdot_{\text{public}} n_2 = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1\cdot e_2) \Downarrow_{bm} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private - Public Multiplication

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public})\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ n_1 \cdot_{\text{private}} \text{encrypt}(n_2) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1\cdot e_2) \Downarrow_{bm3} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private Multiplication

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private}\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ n_1 \cdot_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1\cdot e_2) \Downarrow_{bm1} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public - Private Multiplication

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private})\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ \text{encrypt}(n_1) \cdot_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1\cdot e_2) \Downarrow_{bm2} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public Division

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public}\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ n_1 \div_{\text{public}} n_2 = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1\div e_2) \Downarrow_{bd} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private - Public Division

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public})\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ n_1 \div_{\text{private}} \text{encrypt}(n_2) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1\div e_2) \Downarrow_{bd3} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private Division

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private}\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ n_1 \div_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1\div e_2) \Downarrow_{bd1} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public - Private Division

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private})\\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1)\\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2)\\ \text{encrypt}(n_1) \div_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1\div e_2) \Downarrow_{bd2} (\gamma,\sigma_2,\text{acc},n_3)}$$

Figure 3.12: Basic SMC^2 semantics for addition, subtraction, multiplication, and division.

Multiplication, and Division are similar to their corresponding Addition cases.

Figures 3.13 and 3.14 give the semantics for binary comparison operations. For all comparison semantic rules, we have true versions, returning 1, and false versions, returning 0. Given that C does not have Booleans, 1 represents true and 0 represents false here. When the comparison involves private data, the result must be private as well, and we return the encryption of the result of the comparison operation. Similarly to the rules for other binary operations, there are four rules each for the true versions and false versions of the comparison operations to properly handle public and private data.

Figure 3.15 gives the semantics for public and private if else branches. The public if else rules, shown in subfigures 3.15d and 3.15e are the same as those defined for regular C, with the additional check that the expression we are branching on is public ($\mathrm{Label}(e, \gamma) = \mathrm{public}$). The Private If Else rule, shown in Figure 3.15c, is more interesting. Our strategy for dealing with private-conditioned branches involves executing both branches as a sequence of statements (with some additional helper algorithms to aid in storing changes, restoration between branches, and resolution of true values). We chose to use big-step semantics to facilitate the comparison of the Basic SMC^2 semantics with the Vanilla C semantics, and for its proof of correctness that we will discuss in Section 3.2. We give also an example of SMC^2 code in Figure 3.15a, and of its execution in Figure 3.15b. We use coloring throughout Figure 3.15 to highlight the corresponding code and rule execution.

The starting and ending states of the SMC^2 Private If Else rule are essentially the same as the starting and ending states of the corresponding Vanilla C If Else rule; however, there are several additional assertions that guarantee that both of the private-conditioned branches are executed. We will proceed to describe now the different assertions.

We first evaluate expression e over environment γ, memory σ and accumulator acc to obtain some number n; the same environment, and a potentially updated memory (e.g. in the case $e = x++$). We proceed to store n as the value of a temporary variable $res_{\mathrm{acc}+1}$, using $\mathrm{acc} + 1$ to denote the current level of nesting in the upcoming **`then`** and **`else`** statements. The variable $res_{\mathrm{acc}+1}$ is used in the resolution phase, to select the result according to the branching condition. We then extract the non-local variables that are assigned some value within either branch. This is achieved using Algorithm 32 ($\mathrm{ExtractVariables}$), which iterates through both statement s_1 and s_2 and stores the variable names in array $list_{\mathrm{acc}+1}$. Next we call Algorithm 33 ($\mathrm{InitializeVariables}$) with arguments $list_{\mathrm{acc}+1}$, γ_1, σ_2 and acc. This iterates through the list of variables in $list_{\mathrm{acc}+1}$, declaring two temporary versions of each variable, named $x_then[\mathrm{acc}]$ and $x_else[\mathrm{acc}]$ and

Private Not Equal To True

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ \text{encrypt}(1) = n_3 \qquad n_1 \neq_{\text{private}} n_2\end{array}}{(\gamma,\sigma,\text{acc},e_1 != e_2) \Downarrow_{net1} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public - Private Not Equal To True

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ \text{encrypt}(1) = n_3 \qquad \text{encrypt}(n_1) \neq_{\text{private}} n_2\end{array}}{(\gamma,\sigma,\text{acc},e_1 != e_2) \Downarrow_{net2} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public Not Equal To True

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ n_1 \neq_{\text{public}} n_2\end{array}}{(\gamma,\sigma,\text{acc},e_1 != e_2) \Downarrow_{net} (\gamma,\sigma_2,\text{acc},1)}$$

Private - Public Not Equal To True

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ \text{encrypt}(1) = n_3 \qquad n_1 \neq_{\text{private}} \text{encrypt}(n_2)\end{array}}{(\gamma,\sigma,\text{acc},e_1 != e_2) \Downarrow_{net3} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public Not Equal To False

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ n_1 =_{\text{public}} n_2\end{array}}{(\gamma,\sigma,\text{acc},e_1 != e_2) \Downarrow_{nef} (\gamma,\sigma_2,\text{acc},0)}$$

Public - Private Not Equal To False

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ \text{encrypt}(0) = n_3 \qquad \text{encrypt}(n_1) =_{\text{private}} n_2\end{array}}{(\gamma,\sigma,\text{acc},e_1 != e_2) \Downarrow_{nef2} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private Not Equal To False

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ \text{encrypt}(0) = n_3 \qquad n_1 =_{\text{private}} n_2\end{array}}{(\gamma,\sigma,\text{acc},e_1 != e_2) \Downarrow_{nef1} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private - Public Not Equal To False

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ \text{encrypt}(0) = n_3 \qquad n_1 =_{\text{private}} \text{encrypt}(n_2)\end{array}}{(\gamma,\sigma,\text{acc},e_1 != e_2) \Downarrow_{nef3} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public Equal To True

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ n_1 =_{\text{public}} n_2\end{array}}{(\gamma,\sigma,\text{acc},e_1 == e_2) \Downarrow_{eqt} (\gamma,\sigma_2,\text{acc},1)}$$

Public - Private Equal To True

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ \text{encrypt}(n_1) =_{\text{private}} n_2 \qquad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1 == e_2) \Downarrow_{eqt2} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private Equal To True

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ n_1 =_{\text{private}} n_2 \qquad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1 == e_2) \Downarrow_{eqt1} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private - Public Equal To True

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ n_1 =_{\text{private}} \text{encrypt}(n_2) \qquad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1 == e_2) \Downarrow_{eqt3} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public Equal To False

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ n_1 \neq_{\text{public}} n_2\end{array}}{(\gamma,\sigma,\text{acc},e_1 == e_2) \Downarrow_{eqf} (\gamma,\sigma_2,\text{acc},0)}$$

Public - Private Equal To False

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ \text{encrypt}(n_1) \neq_{\text{private}} n_2 \qquad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1 == e_2) \Downarrow_{eqf2} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private Equal To False

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ n_1 \neq_{\text{private}} n_2 \qquad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1 == e_2) \Downarrow_{eqf1} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private - Public Equal To False

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ n_1 \neq_{\text{private}} \text{encrypt}(n_2) \qquad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1 == e_2) \Downarrow_{eqf3} (\gamma,\sigma_2,\text{acc},n_3)}$$

Figure 3.13: Basic SMC^2 semantics for not equal to and equal to comparison operations.

Public Less Than True

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ n_1 <_{\text{public}} n_2\end{array}}{(\gamma,\sigma,\text{acc},e_1 < e_2) \Downarrow_{ltt} (\gamma,\sigma_2,\text{acc},1)}$$

Public - Private Less Than True

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ \text{encrypt}(n_1) <_{\text{private}} n_2 \qquad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1 < e_2) \Downarrow_{ltt2} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private Less Than True

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) == \text{Label}(e_2,\gamma) == \text{private} \\ (\gamma,\ \sigma,\ \text{acc},\ e_1) \Downarrow_e (\gamma,\ \sigma_1,\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \text{acc},\ e_2) \Downarrow_e (\gamma,\ \sigma_2,\ \text{acc},\ n_2) \\ n_1 <_{\text{private}} n_2 \qquad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\ \sigma,\ \text{acc},\ e_1 < e_2) \Downarrow_{ltt1} (\gamma,\ \sigma_2,\ \text{acc},\ n_3)}$$

Private - Public Less Than True

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ \text{encrypt}(n_2) = n_2' \qquad n_1 <_{\text{private}} n_2' \qquad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1 < e_2) \Downarrow_{ltt3} (\gamma,\sigma_2,\text{acc},n_3)}$$

Public Less Than False

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ n_1 >=_{\text{public}} n_2\end{array}}{(\gamma,\sigma,\text{acc},e_1 < e_2) \Downarrow_{ltf} (\gamma,\sigma_2,\text{acc},0)}$$

Public - Private Less Than False

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ \text{encrypt}(n_1) >=_{\text{private}} n_2 \qquad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1 < e_2) \Downarrow_{ltf2} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private Less Than False

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ n_1 >=_{\text{private}} n_2 \qquad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1 < e_2) \Downarrow_{ltf1} (\gamma,\sigma_2,\text{acc},n_3)}$$

Private - Public Less Than False

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \\ (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},n_1) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n_2) \\ n_1 >=_{\text{private}} \text{encrypt}(n_2) \qquad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\sigma,\text{acc},e_1 < e_2) \Downarrow_{ltf3} (\gamma,\sigma_2,\text{acc},n_3)}$$

Figure 3.14: Basic SMC^2 semantics for less than comparison operations.

```
1 private int a=3, b=7, c=0;
2 if (a < b) {c = a;}
3 else {c = b;}
```

(a) SMC^2 code

```
1 private int a=3, b=7, c=0;
2 private int res1 = a < b;
3 private int c_t = c, c_e = c;
4 c = a;
5 c_t = c; c = c_e;
6 c = b;
7 c = (res1 ∧ c_t) ∨ (¬res1 ∧ c);
```

(b) Basic SMC^2 code execution

$$\frac{\begin{array}{l}\text{Label}(e,\gamma) = \text{private} \qquad (\gamma,\ \sigma,\ \text{acc},\ e) \Downarrow_e (\gamma,\ \sigma_1,\ \text{acc},\ n) \\ (\gamma,\ \sigma_1,\ \text{acc},\ \text{private int } res_{\text{acc}+1} = n) \Downarrow_s (\gamma_1,\ \sigma_2,\ \text{acc},\ \text{skip}) \\ \text{Extract_variables}(s_1,\ s_2) = x_{list} \\ \text{InitializeVariables}(x_{list},\ \gamma_1,\ \sigma_2,\ \text{acc}+1) = (\gamma_2,\sigma_3) \\ (\gamma_2,\ \sigma_3,\ \text{acc}+1,\ s_1) \Downarrow_s (\gamma_3,\ \sigma_4,\ \text{acc}+1,\ \text{skip}) \\ \text{RestoreVariables}(x_{list},\ \gamma_3,\ \sigma_4,\ \text{acc}+1) = \sigma_5 \\ (\gamma_3,\ \sigma_5,\ \text{acc}+1,\ s_2) \Downarrow_s (\gamma_4,\ \sigma_6,\ \text{acc}+1,\ \text{skip}) \\ \text{ResolveVariables}(x_{list},\ \gamma_4,\ \sigma_6,\ \text{acc}+1,\ res_{\text{acc}+1}) = \sigma_7\end{array}}{(\gamma,\ \sigma,\ \text{acc},\ \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iep} (\gamma,\ \sigma_7,\ \text{acc},\ \text{skip})}$$

(c) Basic SMC^2 rule Private If Else

$$\frac{\begin{array}{l}\text{Label}(e,\gamma) = \text{public} \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},n) \\ n \neq 0 \qquad (\gamma,\sigma_1,\text{acc},s_1) \Downarrow_s (\gamma_1,\sigma_2,\text{acc},\text{skip})\end{array}}{(\gamma,\sigma,\text{acc},\text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iet} (\gamma,\sigma_2,\text{acc},\text{skip})}$$

(d) Basic SMC^2 Public If Else True

$$\frac{\begin{array}{l}\text{Label}(e,\gamma) = \text{public} \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},n) \\ n = 0 \qquad (\gamma,\sigma_1,\text{acc},s_2) \Downarrow_s (\gamma_1,\sigma_2,\text{acc},\text{skip})\end{array}}{(\gamma,\sigma,\text{acc},\text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{ief} (\gamma,\sigma_2,\text{acc},\text{skip})}$$

(e) Basic SMC^2 Public If Else False

Figure 3.15: **`if else`** branching on private data example (3.15a, 3.15b) matching to the public Basic SMC^2 (3.15d, 3.15e) and private Basic SMC^2 (3.15c) if else rules. Coloring in the rules highlight the corresponding code and rule execution.

initialized with the value that x has in the memory σ_2, and returns the updated environment γ_2 and memory σ_3. This is followed by the evaluation of the **`then`** statement. Next, we must restore the original memory. To do this, we call Algorithm 34 ($\mathrm{RestoreVariables}$) to iterate through each of the variables x contained within $list_{\mathrm{acc+1}}$, saving their current value in σ_1 into their **`then`** temporary (i.e., $x_then_{\mathrm{acc+1}} = x$) and restoring their original value from their **`else`** temporary (i.e., $x = x_else_{\mathrm{acc+1}}$). Now the evaluation of the **`else`** statement can occur.

Finally, we need to perform the resolution of all changes made to variables in either branch. To do this, we call Algorithm 35 ($\mathrm{ResolveVariables}$), again iterating through each of the variables x contained within $list_{\mathrm{acc+1}}$ and resolving their values accordingly for the private condition (whose value we stored in $res_{\mathrm{acc+1}}$). For variables that are not array or pointer variables (e.g., those in the example to the left), we perform a series of logical operations over the byte values of the private variables, shown at the bottom left as `c = (res ∧ c_t)∨(¬res ∧ c_e)`. The process is similar for arrays, with some addition bookkeeping due to their structure as a const pointer referring to the location with the array data. For pointers, we must handle the different locations referred to from each branch, merging the two location lists and finding what the true location is. Notice that, in the conclusion, we revert to the original environment γ. In this way, all the temporary variables we used become out of scope and are discarded - in particular, this prevents reusing the same temporary variable name if we have multiple (not nested) private if else statements. It is worthwhile to stress again the role of the accumulator here with respect to other statements. We increment it when we evaluate the **`then`** and **`else`** statements, so that if we attempt to evaluate a (sub)statement with public side effects or restricted operations, we have an (oblivious) runtime failure.

Figure 3.16 shows the semantic rules for the pre-increment operator on regular (non-array, non-pointer) variables and pointers with a single location at the first level of indirection. Figure 3.17 shows the remaining semantic rules for the pre-increment operator on private pointers with multiple locations and public and private pointers with a higher level of indirection. All of these rules have the condition that we are not within an **`if else`** statement branching on private data, as Basic SMC^2 semantics and tracking only supports direct variable assignments, not indirect variable modifications such as this. This operator behaves as expected for C, incrementing the data of regular variables by 1 or moving the location that the pointer refers to forward in memory by the number of bytes of the expected type of data referred to by the pointer. For private pointers with multiple locations, the behavior follows the same concept (i.e., all locations are moved forward by the expected amount).

Pre-Increment Public Variable

$$\frac{\begin{array}{l}\text{acc} = 0 \quad \gamma(x) = (l, \text{public } bty) \quad \sigma(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1)) \\ \text{DecodeVal}(\text{public } bty, 1, \omega) = v \quad v_1 =_{\text{public}} v +_{\text{public}} 1 \quad \text{UpdateVal}(\sigma, l, v_1, \text{public } bty) = \sigma_1\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin} (\gamma, \sigma_1, \text{acc}, v_1)}$$

Pre-Increment Private Variable

$$\frac{\begin{array}{l}\gamma(x) = (l, \text{private } bty) \quad \sigma(l) = (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) \\ \text{acc} = 0 \quad (bty = \text{int}) \vee (bty = \text{float}) \quad \text{DecodeVal}(\text{private } bty, 1, \omega) = v \\ v_1 =_{\text{private}} v +_{\text{private}} \text{encrypt}(1) \quad \text{UpdateVal}(\sigma, l, v_1, \text{private } bty) = \sigma_1\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin1} (\gamma, \sigma_1, \text{acc}, v_1)}$$

Pre-Increment Public Pointer Single Location

$$\frac{\begin{array}{l}\text{acc} = 0 \quad \gamma(x) = (l, \text{public } bty*) \quad \sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1] \quad ((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty), \sigma) \\ \text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \text{public } bty*) = (\sigma_1, 1)\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))}$$

Pre-Increment Public Pointer Single Location (Not Aligned)

$$\frac{\begin{array}{l}\text{acc} = 0 \quad \gamma(x) = (l, \text{public } bty*) \quad \sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1] \quad ((l_2, \mu_2), 0) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty), \sigma) \\ \text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \text{public } bty*) = (\sigma_1, 1)\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow^*_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))}$$

Pre-Increment Private Pointer Single Location

$$\frac{\begin{array}{l}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1)) \quad \text{acc} = 0 \\ \text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1] \quad ((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{private } bty), \sigma) \\ \text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \text{private } bty*) = (\sigma_1, 1)\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin6} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))}$$

Pre-Increment Private Pointer Single Location (Not Aligned)

$$\frac{\begin{array}{l}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1)) \quad \text{acc} = 0 \\ \text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1] \quad ((l_2, \mu_2), 0) = \text{GetLocation}((l_1, \mu_1), \tau(\text{private } bty), \sigma) \\ \text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \text{private } bty*) = (\sigma_1, 1)\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow^*_{pin6} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))}$$

Figure 3.16: Basic SMC^2 semantics for the pre-increment operator (`++x`).

Figure 3.18 shows the rules for functions and casting. We restrict function declarations and definitions to be outside of an **`if else`** statement branching on private data, as these declarations cause public side effects in memory. When a function is defined, we evaluate whether or not the function will cause public side effects, adding a tag to the data stored for the function to indicate this (i.e., 0 for no public side effects, 1 for public side effects). When we execute a function call, we look up the function, then store the argument values for the function as their corresponding names within the function, and execute the function. As expected with scoping in C, we discard the variables local to the function from the environment, but we keep all new memory locations created within the function. When executing a function call that has public side effects, we ensure we are not inside an **`if else`** branching on private data. In our subset of semantics, we chose not to model functions with return statements as we feel they do not provide additional complexity beyond the

Pre-Increment Public Pointer Higher Level Indirection Single Location

$$\frac{\begin{array}{l}\text{acc} = 0 \quad \gamma(x) = (l, \text{public } bty*) \quad \sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i] \quad ((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty*), \sigma) \\ i > 1 \quad \text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], i], \text{public } bty*) = (\sigma_1, 1)\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin3} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))}$$

Pre-Increment Public Pointer Higher Level Indirection Single Location (Not Aligned)

$$\frac{\begin{array}{l}\text{acc} = 0 \quad \gamma(x) = (l, \text{public } bty*) \quad \sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i] \quad ((l_2, \mu_2), 0) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty*), \sigma) \\ i > 1 \quad \text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], i], \text{public } bty*) = (\sigma_1, 1)\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow^*_{pin3} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))}$$

Pre-Increment Private Pointer Higher Level Indirection Single Location

$$\frac{\begin{array}{l}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1)) \quad \text{acc} = 0 \\ \text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i] \quad ((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{private } bty*), \sigma) \\ i > 1 \quad \text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], i], \text{private } bty*) = (\sigma_1, 1)\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin7} (\gamma, \sigma_1, \text{acc}, (l_2, 0))}$$

Pre-Increment Private Pointer Higher Level Indirection Single Location (Not Aligned)

$$\frac{\begin{array}{l}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1)) \quad \text{acc} = 0 \\ \text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i] \quad ((l_2, \mu_2), 0) = \text{GetLocation}((l_1, \mu_1), \tau(\text{private } bty*), \sigma) \\ i > 1 \quad \text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], i], \text{private } bty*) = (\sigma_1, 1)\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow^*_{pin7} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))}$$

Pre-Increment Private Pointer Multiple Locations

$$\frac{\begin{array}{l}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{acc} = 0 \quad \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1] \\ \text{IncrementList}(\bar{l}, \tau(\text{private } bty), \sigma) = (\bar{l}', 1) \quad \text{UpdatePtr}(\sigma, (l, 0), [\alpha, \bar{l}', \bar{j}, 1], \text{private } bty*) = (\sigma_1, 1)\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin4} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1])}$$

Pre-Increment Private Pointer Multiple Locations (Not Aligned)

$$\frac{\begin{array}{l}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{acc} = 0 \quad \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1] \\ \text{IncrementList}(\bar{l}, \tau(\text{private } bty), \sigma) = (\bar{l}', 0) \quad \text{UpdatePtr}(\sigma, (l, 0), [\alpha, \bar{l}', \bar{j}, 1], \text{private } bty*) = (\sigma_1, 1)\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow^*_{pin4} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1])}$$

Pre-Increment Pointer Higher Level Indirection Multiple Locations

$$\frac{\begin{array}{l}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{acc} = 0 \quad \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i] \quad i > 1 \\ \text{IncrementList}(\bar{l}, \tau(\text{private } bty*), \sigma) = (\bar{l}', 1) \quad \text{UpdatePtr}(\sigma, (l, 0), [\alpha, \bar{l}', \bar{j}, i], \text{private } bty*) = (\sigma_1, 1)\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin5} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}', \bar{j}, i])}$$

Pre-Increment Pointer Higher Level Indirection Multiple Locations (Not Aligned)

$$\frac{\begin{array}{l}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{acc} = 0 \quad \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i] \quad i > 1 \\ \text{IncrementList}(\bar{l}, \tau(\text{private } bty*), \sigma) = (\bar{l}', 0) \quad \text{UpdatePtr}(\sigma, (l, 0), [\alpha, \bar{l}', \bar{j}, i], \text{private } bty*) = (\sigma_1, 1)\end{array}}{(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow^*_{pin5} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}', \bar{j}, i])}$$

Figure 3.17: Additional Basic SMC^2 semantics for the pre-increment operator (++x) on private pointers.

Function Declaration

$$\frac{\begin{array}{l}\text{acc} = 0 \quad \text{GetFunTypeList}(\overline{p}) = \overline{ty} \quad l = \phi() \quad \gamma_1 = \gamma[x \to (l, \overline{ty} \to ty)] \\ \sigma_1 = \sigma[l \to (\text{NULL}, \overline{ty} \to ty, 1, \text{PermL_Fun(public)})]\end{array}}{(\gamma, \sigma, \text{acc}, ty\ x(\overline{p})) \Downarrow_{df} (\gamma_1, \sigma_1, \text{acc}, \text{skip})}$$

Pre-Declared Function Definition

$$\frac{\begin{array}{l}\text{acc} = 0 \quad x \in \gamma \quad \gamma(x) = (l, \overline{ty} \to ty) \quad \text{CheckPublicEffects}(s, x, \gamma, \sigma) = n \quad \text{EncodeFun}(s, n, \overline{p}) = \omega \\ \sigma = \sigma_1[l \to (\text{NULL}, \overline{ty} \to ty, 1, \text{PermL_Fun(public)})] \quad \sigma_2 = \sigma_1[l \to (\omega, \overline{ty} \to ty, 1, \text{PermL_Fun(public)})]\end{array}}{(\gamma, \sigma, \text{acc}, ty\ x(\overline{p})\{s\}) \Downarrow_{fpd} (\gamma, \sigma_2, \text{acc}, \text{skip})}$$

Function Definition

$$\frac{\begin{array}{l}l = \phi() \quad \text{GetFunTypeList}(\overline{p}) = \overline{ty} \quad \gamma_1 = \gamma[x \to (l, \overline{ty} \to ty)] \quad \text{CheckPublicEffects}(s, x, \gamma, \sigma) = n \\ x \notin \gamma \quad \text{acc} = 0 \quad \text{EncodeFun}(s, n, \overline{p}) = \omega \quad \sigma_1 = \sigma[l \to (\omega, \overline{ty} \to ty, 1, \text{PermL_Fun(public)})]\end{array}}{(\gamma, \sigma, \text{acc}, ty\ x(\overline{p})\{s\}) \Downarrow_{fd} (\gamma_1, \sigma_1, \text{acc}, \text{skip})}$$

Function Call With Public Side Effects

$$\frac{\begin{array}{l}\gamma(x) = (l, \overline{ty} \to ty) \quad \sigma(l) = (\omega, \overline{ty} \to ty, 1, \text{PermL_Fun(public)}) \quad \text{DecodeFun}(\omega) = (s, 1, \overline{p}) \quad \text{acc} = 0 \\ \text{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1 \quad (\gamma, \sigma, \text{acc}, s_1) \Downarrow_s (\gamma_1, \sigma_1, \text{acc}, \text{skip}) \quad (\gamma_1, \sigma_1, \text{acc}, s) \Downarrow_s (\gamma_2, \sigma_2, \text{acc}, \text{skip})\end{array}}{(\gamma,\ \sigma,\ \text{acc},\ x(\overline{e})) \Downarrow_{fc} (\gamma,\ \sigma_2,\ \text{acc},\ v)}$$

Function Call Without Public Side Effects

$$\frac{\begin{array}{l}\gamma(x) = (l, \overline{ty} \to ty) \quad \sigma(l) = (\omega, \overline{ty} \to ty, 1, \text{PermL_Fun(public)}) \quad \text{DecodeFun}(\omega) = (s, 0, \overline{p}) \\ \text{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1 \quad (\gamma, \sigma, \text{acc}, s_1) \Downarrow_s (\gamma_1, \sigma_1, \text{acc}, \text{skip}) \quad (\gamma_1, \sigma_1, \text{acc}, s) \Downarrow_s (\gamma_2, \sigma_2, \text{acc}, \text{skip})\end{array}}{(\gamma, \sigma, \text{acc}, x(\overline{e})) \Downarrow_{fc1} (\gamma, \sigma_2, \text{acc}, \text{NULL})}$$

Cast Private Location

$$\frac{\begin{array}{l}(\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, (l, 0)) \quad (ty = \text{private int}*) \vee (ty = \text{private float}*) \vee (ty = \text{int}*) \vee (ty = \text{float}*) \\ \sigma_1 = \sigma_2[l \to (\omega, \text{void}, n, \text{PermL(Freeable}, ty, \text{private}, n))] \\ \sigma_3 = \sigma_2\Big[l \to \Big(\omega, ty, \frac{n}{\tau(ty)}, \text{PermL}\Big(\text{Freeable}, ty, \text{private}, \frac{n}{\tau(ty)}\Big)\Big)\Big]\end{array}}{(\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl1} (\gamma, \sigma_3, \text{acc}, (l, 0))}$$

Cast Public Location

$$\frac{\begin{array}{l}(\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, (l, 0)) \quad \sigma_1 = \sigma_2[l \to (\omega, \text{void}, n, \text{PermL(Freeable}, \text{void}, \text{public}, n))] \\ (ty = \text{public}\ bty*) \vee (ty = \text{char}*) \quad \sigma_3 = \sigma_2\Big[l \to \Big(\omega, ty, \frac{n}{\tau(ty)}, \text{PermL(Freeable}, ty, \text{public}, \frac{n}{\tau(ty)}\Big)\Big)\Big]\end{array}}{(\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl} (\gamma, \sigma_3, \text{acc}, (l, 0))}$$

Cast Public Value

$$\frac{\begin{array}{l}\text{Label}(e, \gamma) = \text{public} \quad (\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, n) \quad n_1 = \text{Cast}(\text{public}, ty, n) \\ (ty = \text{public int}) \vee (ty = \text{public float})\end{array}}{(\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv} (\gamma, \sigma_1, \text{acc}, n_1)}$$

Cast Private Value

$$\frac{\begin{array}{l}\text{Label}(e, \gamma) = \text{private} \quad (\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, n) \quad n_1 = \text{Cast}(\text{private}, ty, n) \\ (ty = \text{private int}) \vee (ty = \text{private float}) \vee (ty = \text{int}) \vee (ty = \text{float})\end{array}}{(\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv1} (\gamma, \sigma_1, \text{acc}, n_1)}$$

Figure 3.18: Basic SMC^2 semantics for functions and casting.

ability to pass arguments by reference.

Rules Cast Private Location and Cast Public Location are the same as those for our C semantics rules for handling the casting of untyped locations (i.e., casting after allocation with `malloc`), one of each privacy label to ensure only appropriate casting occurs. When casting a location, the byte-wise permissions and privacy labels cannot be modified. The rules Cast Public Value and Cast Private Value handle interpreting an expression as the given type. We have two separate rules, one for each privacy label, as we do not allow casting a public expression to be private or a private expression to be public. This restriction is to prevent unintended declassification of private data.

The semantics rules for multiparty input and output helper functions are shown in Figure 3.19. For regular variables, the input and output functions have two parameters, one for the variable name and one for the id of the party supplying the input or receiving the output, respectively. For arrays, these functions have a third parameter for the length of the array. For the input functions, we look up the type of the variable, then we read in data from the given party and assign it to that variable. For the output functions, we obtain the data from memory, then output the data to the appropriate party. We do not encrypt or decrypt data within these functions, as private input data is expected to be already encrypted, and output data will be decrypted by the corresponding output party, not the computational party that is running the program.

Figure 3.20 shows additional semantics for allocation and deallocation. Rules Public Malloc and Public Free work as expected in C, with the additional assertion that we are not inside an **`if else`** statement branching on private data. These functions both cause public side effects in memory, and so we do not allow them inside that type of statement to prevent potential leakage of information about the private data.

In rule Private Malloc, we assert that the given type is either private int or private float, as this function only handles those types, and that the accumulator acc is 0 (i.e., we are not inside an **`if else`** statement branching on private data, as this function causes public side effects). Then we evaluate e to n and obtain the next open memory location l from ϕ. We add to σ_1 the new mapping from location l to the tuple of a NULL set of bytes; the type ty; the size n; and a list of $\mathrm{private}$, $\mathrm{Freeable}$ permissions. As with public `malloc`, we return $(l, 0)$.

In rule Private Free, we evaluate e to x, assert that x is a private pointer of type int or float and acc is 0 (as this rule causes public side effects). We then use Algorithm 75 (PFree), which performs identically to the function called from public free when there is only one location being pointed to. However, when there are multiple locations, we must free one location based on publicly available information, regardless of the true

SMC Input Private Value

$$\frac{\begin{array}{l}\text{Label}(e_2,\gamma) = \text{private} \quad (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},x) \quad (\gamma\ \sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n) \\ \gamma(x) = (l,\text{private } bty) \quad \text{InputValue}(x,n) = n_1 \quad (\gamma,\sigma_2,\text{acc},x = n_1) \Downarrow_s (\gamma,\sigma_3,\text{acc},\text{skip})\end{array}}{(\gamma,\sigma,\text{acc},\text{smcinput}(e_1,e_2)) \Downarrow_{inp3} (\gamma,\sigma_3,\text{acc},\text{skip})}$$

SMC Input Public Value

$$\frac{\begin{array}{l}\text{Label}(e_2,\gamma) = \text{public} \quad (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},x) \quad \text{acc} = 0 \quad (\gamma\ \sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n) \\ \gamma(x) = (l,\text{public } bty) \quad \text{InputValue}(x,n) = n_1 \quad (\gamma,\sigma_2,\text{acc},x = n_1) \Downarrow_s (\gamma,\sigma_3,\text{acc},\text{skip})\end{array}}{(\gamma,\sigma,\text{acc},\text{smcinput}(e_1,e_2)) \Downarrow_{inp} (\gamma,\sigma_3,\text{acc},\text{skip})}$$

SMC Output Public Value

$$\frac{\begin{array}{l}\text{Label}(e_2,\gamma) = \text{public} \quad (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},x) \quad (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n) \\ \gamma(x) = (l,\text{public } bty) \quad \sigma_2(l) = (\omega,\text{public } bty,1,\text{PermL}(\text{Freeable},\text{public } bty,\text{public},1)) \\ \text{DecodeVal}(\text{public } bty,1,\omega) = n_1 \quad \text{OutputValue}(x,n,n_1)\end{array}}{(\gamma,\sigma,\text{acc},\text{smcoutput}(e_1,e_2)) \Downarrow_{out} (\gamma,\sigma_2,\text{acc},\text{skip})}$$

SMC Output Private Value

$$\frac{\begin{array}{l}\text{Label}(e_2,\gamma) = \text{public} \quad (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},x) \quad (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n) \\ \gamma(x) = (l,\text{private } bty) \quad \sigma_2(l) = (\omega,\text{private } bty,1,\text{PermL}(\text{Freeable},\text{private } bty,\text{private},1)) \\ \text{DecodeVal}(\text{private } bty,1,\omega) = n_1 \quad \text{OutputValue}(x,n,n_1)\end{array}}{(\gamma,\sigma,\text{acc},\text{smcoutput}(e_1,e_2)) \Downarrow_{out3} (\gamma,\sigma_2,\text{acc},\text{skip})}$$

SMC Input Public Array

$$\frac{\begin{array}{l}\text{Label}(e_2,\gamma) = \text{Label}(e_3,\gamma) = \text{public} \quad \text{acc} = 0 \quad (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},x) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n) \quad (\gamma,\sigma_2,\text{acc},e_3) \Downarrow_e (\gamma,\sigma_3,\text{acc},n_1) \quad \gamma(x) = (l,\text{public const } bty*) \\ \text{InputArray}(x,n,n_1) = [m_0,...,m_{n_1}] \quad (\gamma,\sigma_3,\text{acc},x = [m_0,...,m_{n_1}]) \Downarrow_s (\gamma,\sigma_4,\text{acc},\text{skip})\end{array}}{(\gamma,\sigma,\text{acc},\text{smcinput}(e_1,e_2,e_3)) \Downarrow_{inp1} (\gamma,\sigma_4,\text{acc},\text{skip})}$$

SMC Output Public Array

$$\frac{\begin{array}{l}\text{Label}(e_2,\gamma) = \text{Label}(e_3,\gamma) = \text{public} \quad (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},x) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n) \quad (\gamma,\sigma_2,\text{acc},e_3) \Downarrow_e (\gamma,\sigma_3,\text{acc},n_1) \quad \gamma(x) = (l,\text{public const } bty*) \\ \sigma_3(l) = (\omega,\text{public const } bty*,1,\text{PermL}(\text{Freeable},\text{public const } bty*,\text{public},1)) \\ \text{DecodePtr}(\text{public const } bty*,1,\omega) = [1,[(l_1,0)],[1],1] \\ \sigma_3(l_1) = (\omega_1,\text{public } bty,n_1,\text{PermL}(\text{Freeable},\text{public } bty,\text{public},n_1)) \\ \text{DecodeVal}(\text{public } bty,n_1,\omega_1) = [m_0,...,m_{n_1}] \quad \text{OutputArray}(x,n,[m_0,...,m_{n_1}])\end{array}}{(\gamma,\sigma,\text{acc},\text{smcoutput}(e_1,e_2,e_3)) \Downarrow_{out1} (\gamma,\sigma_3,\text{acc},\text{skip})}$$

SMC Input Private Array

$$\frac{\begin{array}{l}\text{Label}(e_2,\gamma) = \text{Label}(e_3,\gamma) = \text{public} \quad (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},x) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n) \quad (\gamma,\sigma_2,\text{acc},e_3) \Downarrow_e (\gamma,\sigma_3,\text{acc},n_1) \quad \gamma(x) = (l,\text{private const } bty*) \\ \text{InputArray}(x,n,n_1) = [m_0,...,m_{n_1}] \quad (\gamma,\sigma_3,\text{acc},x = [m_0,...,m_{n_1}]) \Downarrow_s (\gamma,\sigma_4,\text{acc},\text{skip})\end{array}}{(\gamma,\sigma,\text{acc},\text{smcinput}(e_1,e_2,e_3)) \Downarrow_{inp4} (\gamma,\sigma_4,\text{acc},\text{skip})}$$

SMC Output Private Array

$$\frac{\begin{array}{l}\text{Label}(e_2,\gamma) = \text{Label}(e_3,\gamma) = \text{public} \quad (\gamma,\sigma,\text{acc},e_1) \Downarrow_e (\gamma,\sigma_1,\text{acc},x) \\ (\gamma,\sigma_1,\text{acc},e_2) \Downarrow_e (\gamma,\sigma_2,\text{acc},n) \quad (\gamma,\sigma_2,\text{acc},e_3) \Downarrow_e (\gamma,\sigma_3,\text{acc},n_1) \quad \gamma(x) = (l,\text{private const } bty*) \\ \sigma_3(l) = (\omega,\text{private const } bty*,1,\text{PermL}(\text{Freeable},\text{private const } bty*,\text{private},1)) \\ \text{DecodePtr}(\text{private const } bty*,1,\omega) = [1,[(l_1,0)],[1],1] \\ \sigma_3(l_1) = (\omega_1,\text{private } bty,n_1,\text{PermL}(\text{Freeable},\text{private } bty,\text{private},n_1)) \\ \text{DecodeVal}(\text{private } bty,n_1,\omega_1) = [m_0,...,m_{n_1}] \quad \text{OutputArray}(x,n,[m_0,...,m_{n_1}])\end{array}}{(\gamma,\sigma,\text{acc},\text{smcoutput}(e_1,e_2,e_3)) \Downarrow_{out4} (\gamma,\sigma_3,\text{acc},\text{skip})}$$

Figure 3.19: Basic SMC^2 semantics for input / output

Public Malloc

$$\frac{\begin{array}{lll} \text{acc} = 0 & \text{Label}(e,\gamma) = \text{public} & (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},n) \\ l = \phi() & \sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, n)\big)\big] & \end{array}}{(\gamma,\sigma,\text{acc},\text{malloc}(e)) \Downarrow_{mal} (\gamma,\sigma_2,\text{acc},(l,0))}$$

Private Malloc

$$\frac{\begin{array}{l} (ty = \text{private int}) \vee (ty = \text{private float}) \quad \text{acc} = 0 \quad (\gamma,\ \sigma,\ \text{acc},\ e) \Downarrow_e (\gamma,\ \sigma_1,\ \text{acc},\ n) \quad l = \phi() \\ \text{Label}(e,\gamma) = \text{public} \quad \sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n \cdot \tau(ty), \text{PermL}(\text{Freeable}, ty, \text{private}, n \cdot \tau(ty))\big)\big] \end{array}}{(\gamma,\ \sigma,\ \text{acc},\ \text{pmalloc}(e,\ ty)) \Downarrow_{malp} (\gamma,\ \sigma_2,\ \text{acc},\ (l,0))}$$

Public Free

$$\frac{\begin{array}{l} \text{acc} = 0 \quad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},x) \\ \gamma(x) = (l, \text{public } bty*) \\ \text{Free}(\sigma_1, l, \gamma) = \sigma_2 \end{array}}{(\gamma,\sigma,\text{acc},\text{free}(e)) \Downarrow_{fre} (\gamma,\sigma_2,\text{acc},\text{skip})}$$

Private Free

$$\frac{\begin{array}{l} (bty = \text{int}) \vee (bty = \text{float}) \quad \text{acc} = 0 \\ (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},x) \\ \gamma(x) = (l, \text{private } bty*) \quad \text{PFree}(\gamma,\sigma_1,l) = (\sigma_2,\bar{l},\bar{j}) \end{array}}{(\gamma,\ \sigma,\ \text{acc},\ \text{pfree}(e)) \Downarrow_{frep} (\gamma,\ \sigma_2,\ \text{acc},\ \text{skip})}$$

Figure 3.20: Basic SMC^2 memory management.

location of the pointer. For that reason, and without loss of generality, we free the first location, l_0. Since l_0 may not be the true location and may be in use by other pointers, we need to do additional computation to maintain correctness without dislosing whether or not this was the true location. In particular, if l_0 is not the true location, we preserve the content of l_0 by obliviously copying it to the pointer's true location prior to freeing. The remaining step is to update other pointers that stored l_0 on their lists to point to the updated location instead of l_0. This behavior is defined in Algorithm 75, and follows the strategy suggested in [22].

Figure 3.21 shows pointer declarations, reading a location from, and writing a location to a pointer. The rules for pointer declarations work as expected in C, with additional check on types to ensure the proper labeling of the byte-wise permissions. The rules for writing to public pointers behave as expected in standard C, but also have the assertion that we cannot be inside an **`if else`** statement branching on private data, as this is a public side effect that would leak information about the private data in the conditional expression. When writing a single location to a private pointer, as in rule Private Pointer Write Single Location, the behavior is as expected in standard C. There is an additional rule (Private Pointer Write Multiple Locations) for writing to private pointers so as to handle the case where the private pointer is being assigned multiple locations; however, the functionality remains similar. When reading from a public pointer or a private pointer that stores a single location, these rules behave as expected in C, returning the location that is stored. When reading from a private pointer with multiple locations, as in rule Private Pointer Read Multiple Locations, the entire pointer data structure is returned to ensure that the true location being referred to is not leaked.

The semantics for pointer dereference writes at the first level of indirection are shown in Figure 3.22 and at higher levels of indirection in Figure 3.23. The rules for pointer dereference write have the assertion

Public Pointer Declaration

$$\frac{\begin{array}{c}(ty = \text{public } bty*) \vee ((ty = bty*) \wedge ((bty = \text{char}) \vee (bty = \text{void}))) \qquad \text{GetIndirection}(*) = i \\ l = \phi() \qquad \gamma_1 = \gamma[x \rightarrow (l, ty)] \qquad \omega = \text{EncodePtr}(ty, [1, [(l_{default}, 0)], [1], i]) \\ \text{acc} = 0 \qquad \sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))]\end{array}}{(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{dp} (\gamma_1, \sigma_1, \text{acc}, \text{skip})}$$

Private Pointer Declaration

$$\frac{\begin{array}{c}((ty = bty*) \vee (ty = \text{private } bty*)) \wedge ((bty = \text{int}) \vee (bty = \text{float})) \qquad \text{GetIndirection}(*) = i \qquad l = \phi() \\ \gamma_1 = \gamma[x \rightarrow (l, \text{ private } bty*)] \qquad \omega = \text{EncodePtr}(\text{private } bty*, [1, [(l_{default}, 0)], [1], i]) \\ \sigma_1 = \sigma[l \rightarrow (\omega, \text{ private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))]\end{array}}{(\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_{dp1} (\gamma_1,\ \sigma_1,\ \text{acc},\ \text{skip})}$$

Public Pointer Write Single Location

$$\frac{\begin{array}{c}\text{acc} = 0 \qquad \text{Label}(e, \gamma) = \text{public} \qquad (\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, (l_e, \mu_e)) \\ \gamma(x) = (l, \text{public } bty*) \qquad \sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i] \\ \text{UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \text{public } bty*) = (\sigma_2, 1)\end{array}}{(\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp} (\gamma, \sigma_2, \text{acc}, \text{skip})}$$

Private Pointer Write Single Location

$$\frac{\begin{array}{c}\gamma(x) = (l,\ \text{private } bty*) \qquad \sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{Label}(e, \gamma) = \text{public} \qquad (bty = \text{int}) \vee (bty = \text{float}) \qquad \text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l}, \bar{j},\ i] \\ (\gamma,\ \sigma,\ \text{acc},\ e) \Downarrow_e (\gamma,\ \sigma_1,\ \text{acc},\ (l_e, \mu_e)) \qquad \text{UpdatePtr}(\sigma_1,\ (l, 0),\ [1,\ [(l_e, \mu_e)],\ [1],\ i], \text{private } bty*) = (\sigma_2, 1)\end{array}}{(\gamma,\ \sigma,\ \text{acc},\ x = e) \Downarrow_{wp1} (\gamma,\ \sigma_2,\ \text{acc},\ \text{skip})}$$

Private Pointer Write Multiple Locations

$$\frac{\begin{array}{c}(\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}, \bar{j}, i]) \qquad \gamma(x) = (l, \text{private } bty*) \qquad (bty = \text{int}) \vee (bty = \text{float}) \\ \text{UpdatePtr}(\sigma_1, (l, 0), [\alpha, \bar{l}, \bar{j}, i], \text{private } bty*) = (\sigma_2, 1)\end{array}}{(\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp2} (\gamma, \sigma_2, \text{acc}, \text{skip})}$$

Private Pointer Read Single Location

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{private } bty*) \qquad \sigma(l) = (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]\end{array}}{(\gamma, \sigma, \text{acc}, x) \Downarrow_{rp2} (\gamma, \sigma, \text{acc}, (l_1, \mu_1))}$$

Private Pointer Read Multiple Locations

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{private } bty*) \qquad \sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i] \qquad (bty = \text{int}) \vee (bty = \text{float})\end{array}}{(\gamma, \sigma, \text{acc}, x) \Downarrow_{rp1} (\gamma, \sigma, \text{acc}, [\alpha, \bar{l}, \bar{j}, i])}$$

Public Pointer Read Single Location

$$\frac{\begin{array}{c}\gamma(x) = (l,\ \text{public } bty*) \qquad \sigma(l) = (\omega,\ \text{public } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i]\end{array}}{(\gamma,\ \sigma,\ \text{acc},\ x) \Downarrow_{rp} (\gamma,\ \sigma,\ \text{acc},\ (l_1, \mu_1))}$$

Figure 3.21: Basic SMC^2 pointer declaration, read, and write rules.

Public Pointer Dereference Write Public Value

$$\frac{\begin{array}{c}(\gamma,\sigma,\text{acc},e)\Downarrow_e(\gamma,\sigma_1,\text{acc},v)\quad \gamma(x)=(l,\text{public } bty*)\quad \text{Label}(e,\gamma)=\text{public}\quad v\neq\text{skip}\\ \sigma_1(l)=(\omega,\text{public } bty*,1,\text{PermL}(\text{Freeable},\text{public } bty*,\text{public},1))\quad \text{acc}=0\\ \text{DecodePtr}(\text{public } bty*,1,\omega)=[1,[(l_1,\mu_1)],[1],1]\quad \text{UpdateOffset}(\sigma_1,(l_1,\mu_1),v,\text{public } bty)=(\sigma_2,1)\end{array}}{(\gamma,\sigma,\text{acc},*x=\ e)\Downarrow_{wdp}(\gamma,\sigma_2,\text{acc},\text{skip})}$$

Public Pointer Dereference Write Public Value (Not Aligned)

$$\frac{\begin{array}{c}\gamma(x)=(l,\text{public } bty*)\quad \sigma_1(l)=(\omega,\text{public } bty*,1,\text{PermL}(\text{Freeable},\text{public } bty*,\text{public},1))\\ (\gamma,\sigma,\text{acc},e)\Downarrow_e(\gamma,\sigma_1,\text{acc},v)\quad v\neq\text{skip}\quad \text{DecodePtr}(\text{public } bty*,1,\omega)=[1,[(l_1,\mu_1)],[1],1]\\ \text{UpdateOffset}(\sigma_1,(l_1,\mu_1),v,\text{public } bty)=(\sigma_2,0)\quad \text{Label}(e,\gamma)=\text{public}\quad \text{acc}=0\end{array}}{(\gamma,\sigma,\text{acc},*x=\ e)\Downarrow^*_{wdp}(\gamma,\sigma_2,\text{acc},\text{skip})}$$

Private Pointer Dereference Write Private Value

$$\frac{\begin{array}{c}(\gamma,\sigma,\text{acc},e)\Downarrow_e(\gamma,\sigma_1,\text{acc},v)\quad \gamma(x)=(l,\text{private } bty*)\quad \text{acc}=0\quad \text{Label}(e,\gamma)=\text{private}\\ (bty=\text{int})\vee(bty=\text{float})\quad \sigma_1(l)=(\omega,\text{private } bty*,\alpha,\text{PermL}(\text{Freeable},\text{private } bty*,\text{private},\alpha))\\ \text{DecodePtr}(\text{private } bty*,\alpha,\omega)=[\alpha,\bar{l},\bar{j},1]\quad v\neq\text{skip}\quad \text{UpdatePriv}(\sigma_1,\alpha,\bar{l},\bar{j},\text{private } bty,v)=(\sigma_2,1)\end{array}}{(\gamma,\sigma,\text{acc},*x=e)\Downarrow_{wdp3}(\gamma,\sigma_2,\text{acc},\text{skip})}$$

Private Pointer Dereference Write Private Value (Not Aligned)

$$\frac{\begin{array}{c}(\gamma,\sigma,\text{acc},e)\Downarrow_e(\gamma,\sigma_1,\text{acc},v)\quad v\neq\text{skip}\quad \gamma(x)=(l,\text{private } bty*)\quad \text{Label}(e,\gamma)=\text{private}\\ (bty=\text{int})\vee(bty=\text{float})\quad \sigma_1(l)=(\omega,\text{private } bty*,\alpha,\text{PermL}(\text{Freeable},\text{private } bty*,\text{private},\alpha))\\ \text{DecodePtr}(\text{private } bty*,\alpha,\omega)=[\alpha,\bar{l},\bar{j},1]\quad \text{acc}=0\quad \text{UpdatePriv}(\sigma_1,\alpha,\bar{l},\bar{j},\text{private } bty,v)=(\sigma_2,0)\end{array}}{(\gamma,\sigma,\text{acc},*x=e)\Downarrow^*_{wdp3}(\gamma,\sigma_2,\text{acc},\text{skip})}$$

Private Pointer Dereference Write Public Value

$$\frac{\begin{array}{c}\text{Label}(e,\gamma)=\text{public}\quad (\gamma,\sigma,\text{acc},e)\Downarrow_e(\gamma,\sigma_1,\text{acc},v)\quad v\neq\text{skip}\quad \text{acc}=0\quad \gamma(x)=(l,\text{private } bty*)\\ (bty=\text{int})\vee(bty=\text{float})\quad \sigma_1(l)=(\omega,\text{private } bty*,\alpha,\text{PermL}(\text{Freeable},\text{private } bty*,\text{private},\alpha))\\ \text{DecodePtr}(\text{private } bty*,\alpha,\omega)=[\alpha,\bar{l},\bar{j},1]\quad \text{UpdatePriv}(\sigma_1,\alpha,\bar{l},\bar{j},\text{private } bty,\text{encrypt}(v))=(\sigma_2,1)\end{array}}{(\gamma,\sigma,\text{acc},*x=e)\Downarrow_{wdp4}(\gamma,\sigma_2,\text{acc},\text{skip})}$$

Private Pointer Dereference Write Public Value (Not Aligned)

$$\frac{\begin{array}{c}\text{Label}(e,\gamma)=\text{public}\quad (\gamma,\sigma,\text{acc},e)\Downarrow_e(\gamma,\sigma_1,\text{acc},v)\quad v\neq\text{skip}\quad \text{acc}=0\quad \gamma(x)=(l,\text{private } bty*)\\ (bty=\text{int})\vee(bty=\text{float})\quad \sigma_1(l)=(\omega,\text{private } bty*,\alpha,\text{PermL}(\text{Freeable},\text{private } bty*,\text{private},\alpha))\\ \text{DecodePtr}(\text{private } bty*,\alpha,\omega)=[\alpha,\bar{l},\bar{j},1]\quad \text{UpdatePriv}(\sigma_1,\alpha,\bar{l},\bar{j},\text{private } bty,\text{encrypt}(v))=(\sigma_2,0)\end{array}}{(\gamma,\sigma,\text{acc},*x=e)\Downarrow^*_{wdp4}(\gamma,\sigma_2,\text{acc},\text{skip})}$$

Figure 3.22: Basic SMC^2 pointer dereference write rules for the first level of indirection.

that we must be outside an **`if else`** branching on private data (i.e., $\mathrm{acc} = 0$) in order to prevent leakage of information about the private data with public pointers and the possibility of incorrect resolution when a private pointer is used at multiple levels within such a statement, as shown in the example in Figure 4.1a. The rules shown in this figure are mostly standard, with the exceptions described below. The rules with $\Downarrow^*$ indicate that the incremented location is not aligned or not of the same type as the expected type of the pointer.

In rule Private Pointer Dereference Write Private Value, we use Algorithm 50 ($\mathrm{UpdatePriv}$) to appropriately update the value at private location. To do this, we must touch each of the α locations that the pointer refers to, and at each location number n, assign $(j_n \wedge v_{new}) \vee (\neg j_n \wedge v_{old})$. The behavior here is similar to that of resolving a value after executing a private-conditioned **`if else`**, as discussed earlier in this section. Rule Private Pointer Dereference Write Public Value is nearly identical, with the added call to encrypt the public value before it is stored.

In rule Private Pointer Dereference Write Higher Level Indirection, we handle writing a new location to the lower level private pointer that a higher level private pointer refers to when the higher level private pointer has multiple locations. We use Algorithm 49 ($\mathrm{UpdatePrivPtr}$) to do this, and it's behavior is similar to that of resolving a private pointer after executing a private-conditioned **`if else`**, as discussed earlier in this section. To do this, we must go to each of the α locations that the higher level pointer stores, then use the Algorithm 78 ($\mathrm{CondAssign}$) on the locations stored for the lower level pointer (and their corresponding tags) and the new location we are trying to write from within this rule (and the corresponding tag that the higher level pointer has for that lower level pointer location).

In rule Private Pointer Dereference, we use Algorithm 73 ($\mathrm{Retrieve_vals}$) to securely iterate through the locations and obtain the value for the true location that the pointer refers to. To do this, we perform $j \wedge v$ for each of the α locations that the pointer refers to, and perform bitwise or operations between all such values.

In rule Private Pointer Dereference Higher Level Indirection, we use Algorithm 72 ($\mathrm{DerefPrivPtr}$) to dereference multiple locations of a private pointer at a higher level of indirection. For each location dereference of the higher level private pointer, we may obtain multiple locations for the lower level private pointer. To resolve these to a condensed list and find the true location for the dereference, we use an algorithm similar to Algorithm 78 ($\mathrm{CondAssign}$) to handle merging the location lists and resolving down to the final set of tags, such that there are no duplicates of a location within the location list and the private tag set to 1 correctly identifies the true location.

Private Pointer Dereference Write Higher Level Indirection

$$\frac{\begin{array}{c}\text{Label}(e,\gamma) = \text{private} \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},(l_e,\mu_e)) \qquad i > 1 \qquad \text{acc} = 0 \\ \gamma(x) = (l, \text{private } bty*) \qquad \sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i] \\ \text{UpdatePrivPtr}(\sigma_1, [\alpha, \bar{l}, \bar{j}, i], [1, [(l_e, \mu_e)], [1], i-1], \text{private } bty*) = (\sigma_2, 1)\end{array}}{(\gamma,\sigma,\text{acc},*x = e) \Downarrow_{wdp2} (\gamma,\sigma_2,\text{acc},\text{skip})}$$

Private Pointer Dereference Write Higher Level Indirection (Not Aligned)

$$\frac{\begin{array}{c}\text{Label}(e,\gamma) = \text{private} \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},(l_e,\mu_e)) \qquad i > 1 \qquad \text{acc} = 0 \\ \gamma(x) = (l, \text{private } bty*) \qquad \sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i] \\ \text{UpdatePrivPtr}(\sigma_1, [\alpha, \bar{l}, \bar{j}, i], [1, [(l_e, \mu_e)], [1], i-1], \text{private } bty*) = (\sigma_2, 0)\end{array}}{(\gamma,\sigma,\text{acc},*x = e) \Downarrow^*_{wdp2} (\gamma,\sigma_2,\text{acc},\text{skip})}$$

Private Pointer Dereference Write Higher Level Indirection Multiple Locations

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{private } bty*) \qquad \sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{Label}(e,\gamma) = \text{private} \qquad \text{acc} = 0 \qquad \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i] \qquad i > 1 \\ (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},[\alpha_e, \bar{l}_e, \bar{j}_e, i-1]) \\ \text{UpdatePrivPtr}(\sigma_1, [\alpha, \bar{l}, \bar{j}, i-1], [\alpha_e, \bar{l}_e, \bar{j}_e, i-1], \text{private } bty*) = (\sigma_2, 1)\end{array}}{(\gamma,\sigma,\text{acc},*x = e) \Downarrow_{wdp5} (\gamma,\sigma_2,\text{acc},\text{skip})}$$

Private Pointer Dereference Write Higher Level Indirection Multiple Locations (Not Aligned)

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{private } bty*) \qquad \sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{Label}(e,\gamma) = \text{private} \qquad \text{acc} = 0 \qquad \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i] \qquad i > 1 \\ (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},[\alpha_e, \bar{l}_e, \bar{j}_e, i-1]) \\ \text{UpdatePrivPtr}(\sigma_1, [\alpha, \bar{l}, \bar{j}, i-1], [\alpha_e, \bar{l}_e, \bar{j}_e, i-1], \text{private } bty*) = (\sigma_2, 1)\end{array}}{(\gamma,\sigma,\text{acc},*x = e) \Downarrow^*_{wdp5} (\gamma,\sigma_2,\text{acc},\text{skip})}$$

Public Pointer Dereference Write Higher Level Indirection

$$\frac{\begin{array}{c}\text{acc} = 0 \qquad \gamma(x) = (l, \text{public } bty*) \qquad \sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i] \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},(l_e,\mu_e)) \\ i > 1 \qquad \text{Label}(e,\gamma) = \text{public} \qquad \text{UpdatePtr}(\sigma_1, (l_1, \mu_1), [1, [(l_e, \mu_e)], [1], i-1], \text{public } bty*) = (\sigma_2, 1)\end{array}}{(\gamma,\sigma,\text{acc},*x = e) \Downarrow_{wdp1} (\gamma,\sigma_2,\text{acc},\text{skip})}$$

Public Pointer Dereference Write Higher Level Indirection (Not Aligned)

$$\frac{\begin{array}{c}\text{acc} = 0 \qquad \gamma(x) = (l, \text{public } bty*) \qquad \sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i] \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},(l_e,\mu_e)) \\ i > 1 \qquad \text{Label}(e,\gamma) = \text{public} \qquad \text{UpdatePtr}(\sigma_1, (l_1, \mu_1), [1, [(l_e, \mu_e)], [1], i-1], \text{public } bty*) = (\sigma_2, 0)\end{array}}{(\gamma,\sigma,\text{acc},*x = e) \Downarrow^*_{wdp1} (\gamma,\sigma_2,\text{acc},\text{skip})}$$

Figure 3.23: Basic SMC^2 semantic rules for pointer dereference writes at higher levels of indirection.

Public Pointer Dereference Single Location

$$\frac{\gamma(x) = (l, \text{public } bty*) \quad \sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \quad \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1] \quad \text{DerefPtr}(\sigma, \text{public } bty, (l_1, \mu_1)) = (v, 1)}{(\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp} (\gamma, \sigma, \text{acc}, v)}$$

Public Pointer Dereference Single Location (Not Aligned)

$$\frac{\gamma(x) = (l, \text{public } bty*) \quad \sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \quad \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1] \quad \text{DerefPtr}(\sigma, \text{public } bty, (l_1, \mu_1)) = (v, 0)}{(\gamma, \sigma, \text{acc}, *x) \Downarrow^*_{rdp} (\gamma, \sigma, \text{acc}, v)}$$

Public Pointer Dereference Single Location Higher Level Indirection

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{public } bty*) \quad \sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i] \quad i > 1 \\ \text{DerefPtrHLI}(\sigma, \text{public } bty*, (l_1, \mu_1)) = ([1, [(l_2, \mu_2)], [1], i-1], 1)\end{array}}{(\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp1} (\gamma, \sigma, \text{acc}, (l_2, \mu_2))}$$

Public Pointer Dereference Single Location Higher Level Indirection (Not Aligned)

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{public } bty*) \quad \sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i] \quad i > 1 \\ \text{DerefPtrHLI}(\sigma, \text{public } bty*, (l_1, \mu_1)) = ([1, [(l_2, \mu_2)], [1], i-1], 0)\end{array}}{(\gamma, \sigma, \text{acc}, *x) \Downarrow^*_{rdp1} (\gamma, \sigma, \text{acc}, (l_2, \mu_2))}$$

Private Pointer Dereference

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1] \quad (bty = \text{int}) \vee (bty = \text{float}) \\ \text{Retrieve_vals}(\alpha, \bar{l}, \bar{j}, \text{private } bty, \sigma) = (v, 1)\end{array}}{(\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp2} (\gamma, \sigma, \text{acc}, v)}$$

Private Pointer Dereference (Not Aligned)

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1] \quad (bty = \text{int}) \vee (bty = \text{float}) \\ \text{Retrieve_vals}(\alpha, \bar{l}, \bar{j}, \text{private } bty, \sigma) = (v, 0)\end{array}}{(\gamma, \sigma, \text{acc}, *x) \Downarrow^*_{rdp2} (\gamma, \sigma, \text{acc}, v)}$$

Private Pointer Dereference Higher Level Indirection

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i] \quad (bty = \text{int}) \vee (bty = \text{float}) \quad i > 1 \\ \text{DerefPrivPtr}(\alpha, \bar{l}, \bar{j}, \text{private } bty*, \sigma) = ((\alpha', \bar{l}', \bar{j}'), 1)\end{array}}{(\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp3} (\gamma, \sigma, \text{acc}, [\alpha', \bar{l}', \bar{j}', i-1])}$$

Private Pointer Dereference Higher Level Indirection (Not Aligned)

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i] \quad (bty = \text{int}) \vee (bty = \text{float}) \quad i > 1 \\ \text{DerefPrivPtr}(\alpha, \bar{l}, \bar{j}, \text{private } bty*, \sigma) = ((\alpha', \bar{l}', \bar{j}'), 0)\end{array}}{(\gamma, \sigma, \text{acc}, *x) \Downarrow^*_{rdp3} (\gamma, \sigma, \text{acc}, [\alpha', \bar{l}', \bar{j}', i-1])}$$

Figure 3.24: Basic SMC^2 semantic rules for pointer dereference read.

Figure 3.25 shows semantics for array declarations and writing to an entire array. For array declarations, we require that the size of the array must be public and that public arrays cannot be declared inside of an **`if else`** statement branching on private data. As is standard for C, we model arrays as a const pointer to the array data, and this structure is set up in memory when an array is declared.

Figure 3.26 shows the rules for reading from an array, and Figure 3.27 shows the rules for writing to an index in an array. When reading from or writing to a public index within an array, the behavior is as expected of C. Reading from a private index in a public array is allowed, as the value can be securely evaluated and returned as an encrypted value; this is shown in rule Public Array Read Private Index. The rule for reading from a private index within a private array is similar to this rule, comparing the private index to the encrypted index at each point in the array, then performing a bitwise and operation over the result of the comparison and the value stored at that index. Finally, bitwise or operations are performed between all resulting values for each index of the array, obtaining the true (encrypted) value stored at the private index.

To prevent leakage of information about private data, we do not allow writing a private value to a public array, nor do we allow writing to a private index within a public array. Otherwise, writing to arrays follows similarly to reading from arrays, with writes to public indexes occurring as expected, and writes to private indexes writing an updated value to each index of the array. This updated value is computed for each array index m and the private index i as $((i = \mathrm{encrypt}(m)) \wedge v_{new}) \vee (\neg(i = \mathrm{encrypt}(m)) \wedge v_m)$.

Additional semantics for overshooting reads and writes with arrays (i.e., using an index less than 0 or greater than the size of the array) are shown in Figures 3.28 and 3.29. There are two rules for each combination of privacy labels for reads and writes. The rules with $\Downarrow^*$ indicate that the read or write goes beyond the bounds of the array and is not aligned or not of the same type as elements of the array. This indication is important, as the correctness of the evaluation of such rules is no longer guaranteed.

We use Algorithm 57 ($\mathrm{ReadOOB}$) to find the appropriate location where that index would be expected to be in memory, grabbing the correct amount of bytes for the size of an element of the type of the array, and interpreting these bytes of data as a value of the type of the array (i.e., we ignore the type and privacy labels of the location itself). It returns a two-tuple of this value and an indicator for whether the read was aligned and of the same type as the array or not. This indicator follows the standard C idea of 0 being false and 1 being true. The idea of Algorithm 58 ($\mathrm{WriteOOB}$) is similar, but instead of reading from the location we iterate to, we write the given value to that location in memory as the byte-representation of the value corresponding to the type of the array (i.e., we ignore the type and privacy labels of the location itself). A two-tuple of the

Public Array Declaration

$$\frac{\begin{array}{l}((ty = \text{public } bty) \wedge ((bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{int}))) \vee (ty = \text{char}) \qquad l = \phi() \\ \text{Label}(e, \gamma) = \text{public} \qquad (\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, n) \qquad \text{acc} = 0 \qquad n > 0 \\ \gamma_1 = \gamma[x \rightarrow (l, \text{public const } bty*)] \qquad l_1 = \phi() \qquad \omega = \text{EncodePtr}(\text{public const } bty*, [1, [(l_1, 0)], [1], 1]) \\ \sigma_2 = \sigma_1[l \rightarrow (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))] \\ \omega_1 = \text{EncodeVal}(\text{private } bty, \text{NULL}) \\ \sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))]\end{array}}{(\gamma, \sigma, \text{acc}, ty\ x[e]) \Downarrow_{da} (\gamma_1, \sigma_3, \text{acc}, \text{skip})}$$

Private Array Declaration

$$\frac{\begin{array}{l}\text{Label}(e, \gamma) = \text{public} \qquad ((ty = \text{private } bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float})) \\ (\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, n) \qquad n > 0 \qquad l = \phi() \qquad l_1 = \phi() \qquad \gamma_1 = \gamma[x \rightarrow (l, \text{private const } bty*)] \\ \omega = \text{EncodePtr}(\text{private const } bty*, [1, [(l_1, 0)], [1], 1]) \qquad \omega_1 = \text{EncodeVal}(\text{private } bty, \text{NULL}) \\ \sigma_2 = \sigma_1[l \rightarrow (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))] \\ \sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))]\end{array}}{(\gamma, \sigma, \text{acc}, ty\ x[e]) \Downarrow_{da1} (\gamma_1, \sigma_3, \text{acc}, \text{skip})}$$

Array Declaration Assignment

$$\frac{(\gamma, \sigma, \text{acc}, ty\ x[e]) \Downarrow_s (\gamma_1, \sigma_1, \text{acc}, \text{skip}) \qquad (\gamma_1, \sigma_1, \text{acc}, x = e) \Downarrow_s (\gamma_1, \sigma_2, \text{acc}, \text{skip})}{(\gamma, \sigma, \text{acc}, ty\ x[e] = e) \Downarrow_{das} (\gamma_1, \sigma_2, \text{acc}, \text{skip})}$$

Public Array Write Entire Array

$$\frac{\begin{array}{l}\text{Label}(e, \gamma) = \text{public} \qquad (\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}]) \qquad \forall v_m \in [v_0, ..., v_{n_e-1}].v_m \neq \text{skip} \\ \gamma(x) = (l, \text{public const } bty*) \\ \sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1] \\ \sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n)) \\ n_e = n \qquad \text{UpdateVal}(\sigma_1, l_1, [v_0, ..., v_{n_e-1}], \text{public } bty) = \sigma_2\end{array}}{(\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wa5} (\gamma, \sigma_2, \text{acc}, \text{skip})}$$

Private Array Write Entire Private Array

$$\frac{\begin{array}{l}(\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}]) \qquad \gamma(x) = (l, \text{private const } bty*) \\ \sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ \forall v_m \in [v_0, ..., v_{n_e-1}].v_m \neq \text{skip} \qquad \text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1] \\ \sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) \qquad (bty = \text{int}) \vee (bty = \text{float}) \\ \text{Label}(e, \gamma) = \text{private} \qquad n_e = n \qquad \text{UpdateVal}(\sigma_1, l_1, [v_0, ..., v_{n-1}], \text{private } bty) = \sigma_2\end{array}}{(\gamma, \sigma, \text{acc}, x = e_1) \Downarrow_{wa6} (\gamma, \sigma_2, \text{acc}, \text{skip})}$$

Private Array Write Entire Public Array

$$\frac{\begin{array}{l}(\gamma, \sigma, \text{acc}, e) \Downarrow_e (\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}]) \qquad \gamma(x) = (l, \text{private const } bty*) \qquad \text{Label}(e, \gamma) = \text{public} \\ \sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ (bty = \text{int}) \vee (bty = \text{float}) \qquad \text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1] \\ \forall v_m \in [v_0, ..., v_{n_e-1}].v_m \neq \text{skip} \qquad \sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) \\ n_e = n \qquad \forall v_m \in [v_0, ..., v_{n_e-1}].\ v'_m = \text{encrypt}(v_m) \qquad \text{UpdateVal}(\sigma_1, l_1, [v'_0, ..., v'_{n_e-1}], \text{private } bty) = \sigma_2\end{array}}{(\gamma, \sigma, \text{acc}, x = e_1) \Downarrow_{wa7} (\gamma, \sigma_2, \text{acc}, \text{skip})}$$

Figure 3.25: Basic SMC^2 semantic rules for array declarations and writing an entire array.

Public Array Read Public Index

$$\frac{\begin{array}{l}\text{Label}(e,\gamma) = \text{public} \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \qquad \gamma(x) = (l, \text{public const } bty*) \\ \sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1] \\ \sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n)) \\ \text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}] \qquad 0 \le i \le n-1\end{array}}{(\gamma,\sigma,\text{acc},x[e]) \Downarrow_{ra} (\gamma,\sigma_1,\text{acc},v_i)}$$

Public Array Read Private Index

$$\frac{\begin{array}{l}(\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \qquad \gamma(x) = (l, \text{public const } bty*) \qquad (bty = \text{int}) \vee (bty = \text{float}) \\ \sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)) \qquad \text{Label}(e,\gamma) = \text{private} \\ \text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \sigma_1(l_1) = (\omega_1,\ \text{public } bty,\ n,\ \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n)) \qquad v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge \text{encrypt}(v_m) \\ \text{DecodeVal}(\text{public } bty,\ n,\ \omega_1) = [v_0,\ ...,\ v_{n-1}]\end{array}}{(\gamma,\ \sigma,\ \text{acc},\ x[e]) \Downarrow_{ra2} (\gamma,\ \sigma_1,\ \text{acc},\ v)}$$

Private Array Read Private Index

$$\frac{\begin{array}{l}\gamma(x) = (l, \text{private const } bty*) \qquad (bty = \text{int}) \vee (bty = \text{float}) \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \\ \sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \qquad \text{Label}(e,\gamma) = \text{private} \\ \text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1] \\ \sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) \qquad v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge v_m \\ \text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]\end{array}}{(\gamma,\sigma,\text{acc},x[e]) \Downarrow_{ra1} (\gamma,\sigma_1,\text{acc},v)}$$

Private Array Read Public Index

$$\frac{\begin{array}{l}\text{Label}(e,\gamma) = \text{public} \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \qquad \gamma(x) = (l, \text{private const } bty*) \\ \sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1] \\ \sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) \\ \text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}] \qquad 0 \le i \le n-1\end{array}}{(\gamma,\sigma,\text{acc},x[e]) \Downarrow_{ra3} (\gamma,\sigma_1,\text{acc},v_i)}$$

Private Array Read Entire Array

$$\frac{\begin{array}{l}\gamma(x) = (l, \text{private const } bty*) \qquad (bty = \text{int}) \vee (bty = \text{float}) \\ \sigma(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1] \\ \sigma(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) \\ \text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]\end{array}}{(\gamma,\sigma,\text{acc},x) \Downarrow_{ra5} (\gamma,\sigma,\text{acc},[v_0, ..., v_{n-1}])}$$

Public Array Read Entire Array

$$\frac{\begin{array}{l}\gamma(x) = (l, \text{public const } bty*) \\ \sigma(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1] \\ \sigma(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n)) \\ \text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]\end{array}}{(\gamma,\sigma,\text{acc},x) \Downarrow_{ra4} (\gamma,\sigma,\text{acc},[v_0, ..., v_{n-1}])}$$

Figure 3.26: Basic SMC^2 semantic rules for reading from arrays.

Public Array Write Public Value Public Index

$$\frac{\begin{array}{l}(\gamma,\sigma,\text{acc},e_1)\ \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \quad (\gamma,\sigma_1,\text{acc},e_2)\ \Downarrow_e (\gamma,\sigma_2,\text{acc},v) \quad \gamma(x)=(l,\text{public const } bty*)\\ \sigma_2(l)=(\omega,\text{public const } bty*,1,\text{PermL}(\text{Freeable},\text{public const } bty*,\text{public},1))\\ \text{DecodePtr}(\text{public const } bty*,1,\omega)=[1,[(l_1,0)],[1],1] \quad \text{Label}(e_1,\gamma)=\text{Label}(e_2,\gamma)=\text{public}\\ \sigma_2(l_1)=(\omega_1,\text{public } bty,n,\text{PermL}(\text{Freeable},\text{public } bty,\text{public},n))\\ \text{DecodeVal}(\text{public } bty,n,\omega_1)=[v_0,...,v_{n-1}] \quad v\neq\text{skip} \quad \text{acc}=0\\ 0\leq i\leq n-1 \quad [v'_0,...,v'_{n-1}]=[v_0,...,v_{n-1}]\Big(\frac{v}{v_i}\Big) \quad \text{UpdateVal}(\sigma_2,l_1,[v'_0,...,v'_{n-1}],\text{public } bty)=\sigma_3\end{array}}{(\gamma,\sigma,\text{acc},x[e_1]\ =e_2)\Downarrow_{wa}(\gamma,\sigma_3,\text{acc},\text{skip})}$$

Private Array Write Private Value Public Index

$$\frac{\begin{array}{l}(\gamma,\sigma,\text{acc},e_1)\ \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \quad (\gamma,\sigma_1,\text{acc},e_2)\ \Downarrow_e (\gamma,\sigma_2,\text{acc},v) \quad \gamma(x)=(l,\text{private const } bty*)\\ v\neq\text{skip} \quad \sigma_2(l)=(\omega,\text{private const } bty*,1,\text{PermL}(\text{Freeable},\text{private const } bty*,\text{private},1))\\ \text{Label}(e_2,\gamma)=\text{private} \quad \text{DecodePtr}(\text{private const } bty*,1,\omega)=[1,[(l_1,0)],[1],1]\\ \sigma_2(l_1)=(\omega_1,\text{private } bty,n,\text{PermL}(\text{Freeable},\text{private } bty,\text{private},n)) \quad 0\leq i\leq n-1\\ \text{DecodeVal}(\text{private } bty,n,\omega_1)=[v_0,...,v_{n-1}] \quad \text{Label}(e_1,\gamma)=\text{public}\\ [v'_0,...,v'_{n-1}]=[v_0,...,v_{n-1}]\Big(\frac{v}{v_i}\Big) \quad \text{UpdateVal}(\sigma_2,l_1,[v'_0,...,v'_{n-1}],\text{private } bty)=\sigma_3\end{array}}{(\gamma,\sigma,\text{acc},x[e_1]\ =e_2)\Downarrow_{wa4}(\gamma,\sigma_3,\text{acc},\text{skip})}$$

Private Array Write Public Value Public Index

$$\frac{\begin{array}{l}(\gamma,\sigma,\text{acc},e_1)\ \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \quad (\gamma,\sigma_1,\text{acc},e_2)\ \Downarrow_e (\gamma,\sigma_2,\text{acc},v) \quad \gamma(x)=(l,\text{private const } bty*)\\ v\neq\text{skip} \quad \sigma_2(l)=(\omega,\text{private const } bty*,1,\text{PermL}(\text{Freeable},\text{private const } bty*,\text{private},1))\\ \text{Label}(e_1,\gamma)=\text{Label}(e_2,\gamma)=\text{public} \quad \text{DecodePtr}(\text{private const } bty*,1,\omega)=[1,[(l_1,0)],[1],1]\\ (bty=\text{int})\vee(bty=\text{float}) \quad \sigma_2(l_1)=(\omega_1,\text{private } bty,n,\text{PermL}(\text{Freeable},\text{private } bty,\text{private},n))\\ \text{DecodeVal}(\text{private } bty,n,\omega_1)=[v_0,...,v_{n-1}] \quad 0\leq i\leq n-1\\ [v'_0,...,v'_{n-1}]=[v_0,...,v_{n-1}]\Big(\frac{\text{encrypt}(v)}{v_i}\Big) \quad \text{UpdateVal}(\sigma_2,l_1,[v'_0,...,v'_{n-1}],\text{private } bty)=\sigma_3\end{array}}{(\gamma,\sigma,\text{acc},x[e_1]\ =e_2)\Downarrow_{wa1}(\gamma,\sigma_3,\text{acc},\text{skip})}$$

Private Array Write Public Value Private Index

$$\frac{\begin{array}{l}(\gamma,\sigma,\text{acc},e_1)\ \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \quad (\gamma,\sigma_1,\text{acc},e_2)\ \Downarrow_e (\gamma,\sigma_2,\text{acc},v) \quad \gamma(x)=(l,\text{private const } bty*)\\ v\neq\text{skip} \quad \sigma_2(l)=(\omega,\text{private const } bty*,1,\text{PermL}(\text{Freeable},\text{private const } bty*,\text{private},1))\\ \text{DecodePtr}(\text{private const } bty*,1,\omega)=[1,[(l_1,0)],[1],1]\\ \sigma_2(l_1)=(\omega_1,\text{private } bty,n,\text{PermL}(\text{Freeable},\text{private } bty,\text{private},n))\\ \text{DecodeVal}(\text{private } bty,n,\omega_1)=[v_0,...,v_{n-1}] \quad (bty=\text{int})\vee(bty=\text{float})\\ \forall v_m\in[v_0,...,v_{n-1}].\ v'_m=((i=\text{encrypt}(m))\wedge\text{encrypt}(v))\vee(\neg(i=\text{encrypt}(m))\wedge v_m)\\ \text{Label}(e_1,\gamma)=\text{private} \quad \text{Label}(e_2,\gamma)=\text{public} \quad \text{UpdateVal}(\sigma_2,l_1,[v'_0,...,v'_{n-1}],\text{private } bty)=\sigma_3\end{array}}{(\gamma,\sigma,\text{acc},x[e_1]\ =e_2)\Downarrow_{wa2}(\gamma,\sigma_3,\text{acc},\text{skip})}$$

Private Array Write Private Value Private Index

$$\frac{\begin{array}{l}(\gamma,\sigma,\text{acc},e_1)\ \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \quad (\gamma,\sigma_1,\text{acc},e_2)\ \Downarrow_e (\gamma,\sigma_2,\text{acc},v) \quad \gamma(x)=(l,\text{private const } bty*)\\ v\neq\text{skip} \quad \sigma_2(l)=(\omega,\text{private const } bty*,1,\text{PermL}(\text{Freeable},\text{private const } bty*,\text{private},1))\\ \text{DecodePtr}(\text{private const } bty*,1,\omega)=[1,[(l_1,0)],[1],1]\\ \sigma_2(l_1)=(\omega_1,\text{private } bty,n,\text{PermL}(\text{Freeable},\text{private } bty,\text{private},n))\\ \text{DecodeVal}(\text{private } bty,n,\omega_1)=[v_0,...,v_{n-1}] \quad (bty=\text{int})\vee(bty=\text{float})\\ \forall v_m\in[v_0,...,v_{n-1}].\ v'_m=((i=\text{encrypt}(m))\wedge v)\vee(\neg(i=\text{encrypt}(m))\wedge v_m)\\ \text{Label}(e_1,\gamma)=\text{Label}(e_2,\gamma)=\text{private} \quad \text{UpdateVal}(\sigma_2,l_1,[v'_0,...,v'_{n-1}],\text{private } bty)=\sigma_3\end{array}}{(\gamma,\sigma,\text{acc},x[e_1]\ =e_2)\Downarrow_{wa3}(\gamma,\sigma_3,\text{acc},\text{skip})}$$

Figure 3.27: Basic SMC^2 semantic rules for writing to an array.

Public Array Read Out-of-Bounds Public Index

$$\frac{\begin{array}{l}\text{Label}(e,\gamma) = \text{public} \qquad \gamma(x) = (l,\ \text{public const } bty*) \qquad (\gamma,\ \sigma,\ \text{acc},\ e) \Downarrow_e (\gamma,\ \sigma_1,\ \text{acc},\ i) \\ \sigma_1(l) = (\omega,\ \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1] \\ \sigma_1(l_1) = (\omega_1,\ \text{public } bty,\ n,\ \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n)) \\ (i < 0) \vee (i \geq n) \qquad \text{ReadOOB}(i,\ n,\ l_1,\ \text{public } bty,\ \sigma_1) = (v,\ 1)\end{array}}{(\gamma,\ \sigma,\ \text{acc},\ x[e]) \Downarrow_{rao} (\gamma,\ \sigma_1,\ \text{acc},\ v)}$$

Public Array Read Out of Bounds Public Index (Not Aligned)

$$\frac{\begin{array}{l}\text{Label}(e,\gamma) = \text{public} \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \qquad \gamma(x) = (l, \text{public const } bty*) \\ \sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1] \\ \sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n)) \\ (i < 0) \vee (i \geq n) \qquad \text{ReadOOB}(i, n, l_1, \text{public } bty, \sigma_1) = (v, 0)\end{array}}{(\gamma,\sigma,\text{acc},x[e]) \Downarrow^*_{rao} (\gamma,\sigma_1,\text{acc},v)}$$

Private Array Read Out of Bounds Public Index

$$\frac{\begin{array}{l}\text{Label}(e,\gamma) = \text{public} \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \qquad \gamma(x) = (l, \text{private const } bty*) \\ \sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1] \\ \sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) \\ (i < 0) \vee (i \geq n) \qquad \text{ReadOOB}(i, n, l_1, \text{private } bty, \sigma_1) = (v, 1)\end{array}}{(\gamma,\sigma,\text{acc},x[e]) \Downarrow_{rao1} (\gamma,\sigma_1,\text{acc},v)}$$

Private Array Read Out of Bounds Public Index (Not Aligned)

$$\frac{\begin{array}{l}\text{Label}(e,\gamma) = \text{public} \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \qquad \gamma(x) = (l, \text{private const } bty*) \\ \sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1] \\ \sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) \\ (i < 0) \vee (i \geq n) \qquad \text{ReadOOB}(i, n, l_1, \text{private } bty, \sigma_1) = (v, 0)\end{array}}{(\gamma,\sigma,\text{acc},x[e]) \Downarrow^*_{rao1} (\gamma,\sigma_1,\text{acc},v)}$$

Figure 3.28: Basic SMC^2 array reading out of bounds rules

updated memory and an indicator for whether the write was aligned and of the same type as the array or not.

Public Array Write Out of Bounds Public Index Public Value

$$\frac{\begin{array}{l}(\gamma,\sigma,\text{acc},e_1)\ \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \quad (\gamma,\sigma_1,\text{acc},e_2)\ \Downarrow_e (\gamma,\sigma_2,\text{acc},v) \quad \gamma(x)=(l,\text{public const } bty*)\\ \text{acc}=0 \quad \sigma_2(l)=(\omega,\text{public const } bty*,1,\text{PermL}(\text{Freeable},\text{public const } bty*,\text{public},1))\\ \text{Label}(e_1,\gamma)=\text{Label}(e_2,\gamma)=\text{public} \quad \text{DecodePtr}(\text{public const } bty*,1,\omega)=[1,[(l_1,0)],[1],1]\\ \sigma_2(l_1)=(\omega_1,\text{public } bty,n,\text{PermL}(\text{Freeable},\text{public } bty,\text{public},n))\\ v\neq\text{skip} \quad (i<0)\vee(i\geq n) \quad \text{WriteOOB}(v,i,n,l_1,\text{public } bty,\sigma_2)=(\sigma_3,1)\end{array}}{(\gamma,\sigma,\text{acc},x[e_1]\ =e_2)\Downarrow_{wao}(\gamma,\sigma_3,\text{acc},\text{skip})}$$

Public Array Write Out-of-Bounds Public Index Public Value (Not Aligned)

$$\frac{\begin{array}{l}(\gamma,\ \sigma,\ \text{acc},\ e_1)\Downarrow_e(\gamma,\ \sigma_1,\ \text{acc},\ i) \quad (\gamma,\ \sigma_1,\ \text{acc},\ e_2)\Downarrow_e(\gamma,\ \sigma_2,\ \text{acc},\ v) \quad \gamma(x)=(l,\text{public const } bty*)\\ \sigma_2(l)=(\omega,\text{public const } bty*,1,\text{PermL}(\text{Freeable},\text{public const } bty*,\text{public},1))\\ \text{Label}(e_1,\gamma)=\text{Label}(e_2,\gamma)=\text{public} \quad \text{DecodePtr}(\text{public const } bty*,1,\omega)=[1,\ [(l_1,0)],\ [1],\ 1]\\ \sigma_2(l_1)=(\omega_1,\ \text{public } bty,\ n,\ \text{PermL}(\text{Freeable},\text{public } bty,\text{public},n))\\ v\neq\text{skip} \quad (i<0)\vee(i\geq n) \quad \text{WriteOOB}(v,\ i,\ n,\ l_1,\ \text{public } bty,\ \sigma_2)=(\sigma_3,\ 0)\end{array}}{(\gamma,\ \sigma,\ \text{acc},\ x[e_1]\ =e_2)\ \Downarrow^*_{wao}(\gamma,\ \sigma_3,\ \text{acc},\ \text{skip})}$$

Private Array Write Public Value Out of Bounds Public Index

$$\frac{\begin{array}{l}(\gamma,\sigma,\text{acc},e_1)\ \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \quad (\gamma,\sigma_1,\text{acc},e_2)\ \Downarrow_e (\gamma,\sigma_2,\text{acc},v) \quad \gamma(x)=(l,\text{private const } bty*)\\ \sigma_2(l)=(\omega,\text{private const } bty*,1,\text{PermL}(\text{Freeable},\text{private const } bty*,\text{private},1))\\ \text{Label}(e_1,\gamma)=\text{Label}(e_2,\gamma)=\text{public} \quad \text{DecodePtr}(\text{private const } bty*,1,\omega)=[1,[(l_1,0)],[1],1]\\ \sigma_2(l_1)=(\omega_1,\text{private } bty,n,\text{PermL}(\text{Freeable},\text{private } bty,\text{private},n))\\ v\neq\text{skip} \quad (i<0)\vee(i\geq n) \quad \text{WriteOOB}(\text{encrypt}(v),i,n,l_1,\text{private } bty,\sigma_2)=(\sigma_3,1)\end{array}}{(\gamma,\sigma,\text{acc},x[e_1]\ =e_2)\Downarrow_{wao1}(\gamma,\sigma_3,\text{acc},\text{skip})}$$

Private Array Write Public Value Out of Bounds Public Index (Not Aligned)

$$\frac{\begin{array}{l}(\gamma,\sigma,\text{acc},e_1)\ \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \quad (\gamma,\sigma_1,\text{acc},e_2)\ \Downarrow_e (\gamma,\sigma_2,\text{acc},v) \quad \gamma(x)=(l,\text{private const } bty*)\\ \sigma_2(l)=(\omega,\text{private const } bty*,1,\text{PermL}(\text{Freeable},\text{private const } bty*,\text{private},1))\\ \text{Label}(e_1,\gamma)=\text{Label}(e_2,\gamma)=\text{public} \quad \text{DecodePtr}(\text{private const } bty*,1,\omega)=[1,[(l_1,0)],[1],1]\\ \sigma_2(l_1)=(\omega_1,\text{private } bty,n,\text{PermL}(\text{Freeable},\text{private } bty,\text{private},n))\\ v\neq\text{skip} \quad (i<0)\vee(i\geq n) \quad \text{WriteOOB}(\text{encrypt}(v),i,n,l_1,\text{private } bty,\sigma_2)=(\sigma_3,0)\end{array}}{(\gamma,\sigma,\text{acc},x[e_1]\ =e_2)\Downarrow^*_{wao1}(\gamma,\sigma_3,\text{acc},\text{skip})}$$

Private Array Write Out of Bounds Public Index Private Value

$$\frac{\begin{array}{l}(\gamma,\sigma,\text{acc},e_1)\ \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \quad (\gamma,\sigma_1,\text{acc},e_2)\ \Downarrow_e (\gamma,\sigma_2,\text{acc},v) \quad \gamma(x)=(l,\text{private const } bty*)\\ \sigma_2(l)=(\omega,\text{private const } bty*,1,\text{PermL}(\text{Freeable},\text{private const } bty*,\text{private},1))\\ \text{Label}(e_2,\gamma)=\text{private} \quad \text{DecodePtr}(\text{private const } bty*,1,\omega)=[1,[(l_1,0)],[1],1]\\ \text{Label}(e_1,\gamma)=\text{public} \quad \sigma_2(l_1)=(\omega_1,\text{private } bty,n,\text{PermL}(\text{Freeable},\text{private } bty,\text{private},n))\\ v\neq\text{skip} \quad (i<0)\vee(i\geq n) \quad \text{WriteOOB}(v,i,n,l_1,\text{private } bty,\sigma_2)=(\sigma_3,1)\end{array}}{(\gamma,\sigma,\text{acc},x[e_1]\ =e_2)\Downarrow_{wao2}(\gamma,\sigma_3,\text{acc},\text{skip})}$$

Private Array Write Out of Bounds Public Index Private Value (Not Aligned)

$$\frac{\begin{array}{l}(\gamma,\sigma,\text{acc},e_1)\ \Downarrow_e (\gamma,\sigma_1,\text{acc},i) \quad (\gamma,\sigma_1,\text{acc},e_2)\ \Downarrow_e (\gamma,\sigma_2,\text{acc},v) \quad \gamma(x)=(l,\text{private const } bty*)\\ \sigma_2(l)=(\omega,\text{private const } bty*,1,\text{PermL}(\text{Freeable},\text{private const } bty*,\text{private},1))\\ \text{Label}(e_1,\gamma)=\text{public} \quad \text{DecodePtr}(\text{private const } bty*,1,\omega)=[1,[(l_1,0)],[1],1]\\ \text{Label}(e_2,\gamma)=\text{private} \quad \sigma_2(l_1)=(\omega_1,\text{private } bty,n,\text{PermL}(\text{Freeable},\text{private } bty,\text{private},n))\\ v\neq\text{skip} \quad (i<0)\vee(i\geq n) \quad \text{WriteOOB}(v,i,n,l_1,\text{private } bty,\sigma_2)=(\sigma_3,0)\end{array}}{(\gamma,\sigma,\text{acc},x[e_1]\ =e_2)\ \Downarrow^*_{wao2}(\gamma,\sigma_3,\text{acc},\text{skip})}$$

Figure 3.29: Basic SMC^2 array writing out of bounds rules

3.1.5 Vanilla C Algorithms

1	PermL	$::= f : (perm,\ \widehat{ty},\ \text{public},\ \widehat{n}) \rightarrow \widehat{\overline{perm}}$
2	PermL_Fun	$::= f : (\text{public}) \rightarrow \widehat{\overline{perm}}$
3	$\phi()$	$::= f : () \rightarrow \widehat{l}$
4	τ	$::= f : (\widehat{ty}) \rightarrow \widehat{n}$
5	Cast	$::= f : (\text{public},\ \widehat{ty},\ \widehat{n}) \rightarrow \widehat{n}$
6	GetIndirection	$::= f : (*) \rightarrow \widehat{n}$
7	EncodeVal	$::= f : (\widehat{bty}, \widehat{v}) \rightarrow \widehat{\omega}$
8	DecodeVal	$::= f : (\widehat{bty},\ \widehat{n},\ \widehat{\omega}) \rightarrow \widehat{v}$
9	EncodePtr	$::= f : (\widehat{ty},\ [1,\ [(\widehat{l}, \widehat{\mu})],\ [1],\ \widehat{i}]) \rightarrow \widehat{\omega}$
10	DecodePtr	$::= f : (\widehat{ty}, 1, \widehat{\omega}) \rightarrow [1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}]$
11	EncodeFun	$::= f : (\widehat{s}, \square, \widehat{\overline{p}}) \rightarrow \widehat{\omega}$
12	DecodeFun	$::= f : (\widehat{s},\ \square,\ \widehat{\overline{p}}) \rightarrow (\widehat{\omega})$
13	UpdateVal	$::= f : (\widehat{\sigma},\ \widehat{l},\ \widehat{v}, \widehat{bty}) \rightarrow \widehat{\sigma}$
14	UpdatePtr	$::= f : (\widehat{\sigma}, (\widehat{l}, \widehat{\mu}), [1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}], \widehat{bty}*) = (\widehat{\sigma}, \widehat{j})$
15	UpdateOffset	$::= f : (\widehat{\sigma},\ (\widehat{l}, \widehat{\mu}),\ \widehat{v}, \widehat{bty}) \rightarrow (\widehat{\sigma}, \widehat{j})$
16	DerefPtr	$::= f : (\widehat{\sigma}, \widehat{bty}, (\widehat{l}, \widehat{\mu})) \rightarrow (\widehat{v}, \widehat{j})$
17	DerefPtrHLI	$::= f : (\widehat{\sigma}, \widehat{bty}*, (\widehat{l}, \widehat{\mu})) \rightarrow ([1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}], \widehat{j})$
18	SetBytes	$::= f : ((\widehat{l}, \widehat{\mu}), \widehat{ty}, \widehat{v}, \widehat{\sigma}) \rightarrow \widehat{\sigma}$
19	GetBytes	$::= f : ((\widehat{l}, \widehat{\mu}), \widehat{ty}, \widehat{\sigma}) \rightarrow \widehat{v}$
20	GetBlock	$::= f : (\widehat{l}) \rightarrow \widehat{l}$
21	GetLocation	$::= f : ((\widehat{l}, \widehat{\mu}), n, \widehat{\sigma}) \rightarrow ((\widehat{l}, \widehat{\mu}), \widehat{j})$
22	GetFunTypeList	$::= f : (\widehat{\overline{p}}) \rightarrow \widehat{\overline{ty}}$
23	GetFunParamAssign	$::= f : (\widehat{\overline{p}}, \widehat{\overline{e}}) \rightarrow \widehat{s}$
26	Free	$::= f : (\widehat{\gamma}, \widehat{l}, \widehat{\sigma}) \rightarrow (\widehat{\sigma})$
27	CheckFreeable	$::= f : (\widehat{\gamma}, [(\widehat{l}, \widehat{\mu})], [1], \widehat{\sigma}) \rightarrow \widehat{j}$
24	ReadOOB	$::= f : (\widehat{i},\ \widehat{n},\ \widehat{l},\ \widehat{bty},\ \widehat{\sigma}) \rightarrow (\widehat{v},\ \widehat{j})$
25	WriteOOB	$::= f : (\widehat{v},\ \widehat{i},\ \widehat{n},\ \widehat{l},\ \widehat{bty},\ \widehat{\sigma}) \rightarrow (\widehat{\sigma},\ \widehat{j})$

28	InputValue	$::= f : (\widehat{x},\ \widehat{n}) \rightarrow \widehat{n}$
29	InputArray	$::= f : (\widehat{x}, \widehat{n}, \widehat{n}) \rightarrow \widehat{\overline{n}}$
30	OutputValue	$::= f : (\widehat{x},\ \widehat{n},\ \widehat{v}) \rightarrow \text{NULL}$
31	OutputArray	$::= f : (\widehat{x},\ \widehat{n},\ \widehat{\overline{n}}) \rightarrow \text{NULL}$

Algorithm 1 $\widehat{\overline{perm}} \leftarrow \text{PermL}(perm, \widehat{ty}, \text{public}, \widehat{n})$

```
1: perm̂ = [ ]
2: if tŷ = btŷ then
3:     for all î ∈ {0...τ(btŷ) · n̂ − 1} do
4:         perm̂ = (î, public, perm) :: perm̂
5:     end for
6: else if (tŷ = btŷ∗) ∨ (tŷ = const btŷ∗) then
7:     m̂ = τ(int) + τ(int) · 2 + τ(int) + τ(int)
8:     for all î ∈ {0...m̂ − 1} do
9:         perm̂ = (î, public, perm) :: perm̂
10:    end for
11: end if
12: return perm̂
```

Algorithm 1 (PermL) creates a byte-wise permission list based on the given arguments. In Vanilla C, the privacy label is always public, so this algorithm just creates a permission list of the appropriate length for the given type and number of elements. For basic types, this is as simple as getting the byte-length of the type and multiplying it by the number of elements. For pointer types, it needs to equate to the size of the pointer data structure, which only even has one possible location in Vanilla C. Therefore, we have the size of an int for the number of locations, then two ints for the memory block identifier and offset, then an int for the tag, and another int for the level of indirection. We then add the public permission tuple for each byte into the permission list, and return it once complete.

Algorithm 2 $\overline{perm} \leftarrow \text{PermL_Fun}(\text{public})$

```
1: perm = [(0, public, Freeable)]
2: return perm
```

Algorithm 2 (PermL_Fun) creates a permission list for a function memory. We leave this separate, and only create a single tuple for this block, which if overshot into will be read as being for all bytes.

Algorithm 3 $\widehat{l}_{next} \leftarrow \phi()$

```
1: next = global_location_counter ++
2: return l̂_next
```

Algorithm 3 (ϕ) defines how new memory block identifiers are obtained, with a global counter that is monotonically increasing after each time ϕ is called.

Algorithm 4 $\widehat{n} \leftarrow \tau(\widehat{ty})$

Algorithm 4 (τ) returns the expected byte-length for the given type. This is implementation and machine-specific, so we leave the lengths to be determined by the implementor.

Algorithm 5 $(\widehat{n}_1) \leftarrow \mathrm{Cast}(\mathrm{public},\ \widehat{ty},\ \widehat{n})$

1: $\widehat{\omega} = \mathrm{EncodeVal}(\widehat{ty}, \widehat{n})$
2: $\widehat{n}_1 = \mathrm{DecodeVal}(\widehat{ty}, 1, \widehat{\omega})$
3: **return** $\widehat{n}_1$

Algorithm 5 (Cast) is designed to take a privacy label, type, and value, and cast the given value as the into the appropriate size for the new type. Here, we model this as simply encoding the given value into it's byte representation for the new type, then decoding it back into a value of the new type. This assumes that there is a standard handling of casting a value of one type to another type defined in the system or implementation itself, but leaves it to the implementation to define the precise behavior, which can vary.

Algorithm 6 $(\widehat{n}) \leftarrow \mathrm{GetIndirection}(*)$

1: $\widehat{n} = |*|$
2: **return** $\widehat{n}$

Algorithm 6 (GetIndirection) takes the $*$ and returns the number of them. This is used in pointer declarations.

Next, we present the algorithms types used for encoding and decoding bytes in memory in our semantics. It is important to note that we leave the specifics of encoding to bytes and decoding from bytes up to the implementation, as this low-level function may vary based on the system and underlying architecture.

Algorithm 7 $\widehat{\omega} \leftarrow \mathrm{EncodeVal}(\widehat{ty}, \widehat{v})$

Algorithm 8 $\widehat{v} \leftarrow \mathrm{DecodeVal}(\widehat{ty}, \widehat{\omega})$

Algorithm 7 (EncodeVal) takes as input a type and a value. It encodes the given value of the given type as bytes of data, and returns those bytes.

Algorithm 8 (DecodeVal) takes as input a type and bytes of data. It interprets the given bytes of data as a value of the given type, and returns that value.

Algorithm 9 $\widehat{\omega} \leftarrow \mathrm{EncodePtr}(\widehat{ty}, [1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}])$

Algorithm 9 (EncodePtr) takes a pointer data structure and encodes it into byte data. It takes a pointer

type, number, and byte data as input. It then encodes the pointer data structure containing the number 1 indicating there is one location, a list containing the location $(\widehat{l}, \widehat{\mu})$, a list with the single tag 1, and a number indicating the level of indirection of the pointer into byte data. This byte data is then returned.

Algorithm 10 $[1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}] \leftarrow \mathrm{DecodePtr}(\widehat{ty}, 1, \widehat{\omega})$

Algorithm 10 ($\mathrm{DecodePtr}$) does the opposite of $\mathrm{EncodePtr}$, taking byte data and retrieving the pointer data structure from it. It takes a pointer type, number, and byte data as input. It then interprets the given set of bytes as a pointer data structure containing the number number 1 indicating there is one location, a list containing the location $(\widehat{l}, \widehat{\mu})$, a list with the single tag 1, and a number indicating the level of indirection of the pointer into byte data. This pointer data structure is then returned.

Algorithm 11 $\widehat{\omega} \leftarrow \mathrm{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}})$

Algorithm 12 $(\widehat{s}, \square, \widehat{\overline{p}}) \leftarrow \mathrm{DecodeFun}(\widehat{\omega})$

Algorithm 11 ($\mathrm{EncodeFun}$) takes the function data and encodes it into its byte representation. It takes as input a statement (body of the function), a placeholder for the SMC^2 tag for whether the function contains public side effects, and the function's parameter list. $\mathrm{EncodeFun}$ then encodes this information into byte data and returns the byte data.

Algorithm 12 ($\mathrm{DecodeFun}$) takes the byte representation of a function and decodes it into the function's information: the statement (body of the function), a placeholder for the SMC^2 tag for whether the function contains public side effects, and the parameter list. It takes as input the byte data and then returns the function's information.

Next, we present the algorithms used to update memory within the semantics. The following algorithms are for regular (int or float) values, array values, and pointer values, respectively, when updating these values in memory.

Algorithm 13 $\widehat{\sigma}_2 \leftarrow \mathrm{UpdateVal}(\widehat{\sigma},\ \widehat{l},\ \widehat{v},\ \widehat{bty})$

1: $\widehat{\omega}_2 = \mathrm{EncodeVal}(\widehat{bty},\ \widehat{v})$
2: $\widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega}_1,\ \widehat{ty},\ \widehat{n},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}, \mathrm{public}, \widehat{n}))] = \sigma$
3: $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega}_2,\ \widehat{ty},\ \widehat{n},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}, \mathrm{public}, \widehat{n}))]$
4: **return** $\widehat{\sigma}_2$

Algorithm 13 ($\mathrm{UpdateVal}$) is used to update regular (int or float) values in memory. It takes as input memory σ, the memory block identifier of the location we will be updating, the value to store into memory,

and the type to store it as. We first encode the value as the specified type, then removes the original mapping from memory and inserts the new mapping with the updated byte data. It returns the updated memory.

Algorithm 14 $(\widehat{\sigma}_2,\ \widehat{j}) \leftarrow \mathrm{UpdatePtr}(\widehat{\sigma},\ (\widehat{l},\widehat{\mu}),\ [1,\ [(\widehat{l}_1,\widehat{\mu}_1)],\ [1],\ \widehat{i}],\ \widehat{bty}*)$

1: $\widehat{j} = 0$
2: $\widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega}_1,\ \widehat{ty}_1,\ \widehat{\alpha}_1,\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}, \mathrm{public}, \widehat{\alpha}_1))] = \widehat{\sigma}$
3: **if** $(\widehat{\mu} = 0) \wedge (\widehat{bty}* = \widehat{ty}_1)$ **then**
4: $\quad \widehat{\omega} = \mathrm{EncodePtr}(\widehat{bty}*,\ [1,\ [(\widehat{l}_1,\widehat{\mu}_1)],\ [1],\ \widehat{i}])$
5: $\quad \widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{ty}_1,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}_1, \mathrm{public}, 1))]$
6: $\quad \widehat{j} = 1$
7: **else**
8: $\quad \widehat{\sigma}_2 = \mathrm{SetBytes}((\widehat{l},\widehat{\mu}), \widehat{bty}*, [1,\ [(\widehat{l}_1,\widehat{\mu}_1)],\ [1],\ \widehat{i}], \widehat{\sigma})$
9: **end if**
10: **return** $(\widehat{\sigma}_2, \widehat{j})$

Algorithm 14 ($\mathrm{UpdatePtr}$) is used to update the pointer data structure for a pointer. It takes as input memory $\widehat{\sigma}$, the location (memory block identifier and offset) we will be updating, the value to store into memory, and the type to store the value as. First, we extract the given memory block identifier's mapping in memory. If the given offset is 0 and the given pointer type matches the pointer type in that mapping, we encode the pointer data structure into its byte representation and add a new mapping to memory with the new byte data, and set the tag to 1, indicating that we performed a *well-aligned* update to memory. Otherwise, we call $\mathrm{SetBytes}$ to perform the update to memory at this location, as it is not *well-aligned*. Finally, it returns the updated memory.

Algorithm 15 $(\widehat{\sigma}_f, \widehat{j}) \leftarrow \mathrm{UpdateOffset}(\widehat{\sigma},\ (\widehat{l},\widehat{\mu}),\ \widehat{v}, \widehat{bty})$

1: $\widehat{j} = 0$
2: $\widehat{\sigma}_f[\widehat{l} \rightarrow (\widehat{\omega}_1,\ \widehat{ty}_1,\ \widehat{\alpha},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}_1, \mathrm{public}, \widehat{\alpha}))] = \widehat{\sigma}$
3: **if** $(\widehat{bty} = \widehat{ty}_1) \wedge (\widehat{\mu} = 0) \wedge (\widehat{\alpha} = 1)$ **then**
4: $\quad \widehat{\omega}_2 = \mathrm{EncodeVal}(\widehat{bty},\ \widehat{v})$
5: $\quad \widehat{\sigma}_f = \widehat{\sigma}_f[\widehat{l} \rightarrow (\widehat{\omega}_2,\ \widehat{bty},\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}, \mathrm{public}, 1))]$
6: $\quad \widehat{j} = 1$
7: **else**
8: $\quad$ **if** $(\widehat{bty} = \widehat{ty}_1) \wedge (\widehat{\mu}\ \%\ \tau(\widehat{bty}) = 0) \wedge (\frac{\widehat{\mu}}{\tau(\widehat{bty})} < \widehat{\alpha})$ **then**
9: $\quad\quad \widehat{j} = 1$
10: $\quad$ **end if**
11: $\quad \widehat{\sigma}_f = \mathrm{SetBytes}((\widehat{l},\widehat{\mu}), \widehat{bty}, \widehat{v}, \widehat{\sigma})$
12: **end if**
13: **return** $(\widehat{\sigma}_f, \widehat{j})$

Algorithm 15 ($\mathrm{UpdateOffset}$) is designed to update a value at an offset within a memory block, and is used by semantic rules Pointer Dereference Write Value and Pointer Dereference Write Value (Not Aligned). First, we extract the memory block we are looking going to be updating. Next, we check if the memory block

is of the expected type, the offset is 0, and the number of locations is 1 to see if this is a simple update. In this case, we encode the value into its byte representation and add the updated mapping into the final memory and set the tag to be 1, as we have made a *well-aligned* update to memory. If not, and the memory block happens to be a block of array data, we can check if our update will be aligned by checking if the type is the same, then if the byte-offset of the pointer aligns with a value of the given type within the block by using the modulo operation, and also that it is within the range of the current block based on the given type and the number of locations in the block. If all of these elements are true, we will have a *well-aligned* update to memory. We use Algorithm 18 ($\mathrm{SetBytes}$) to perform the update here, as that algorithm facilitates proper insertion of the byte representation for a value into a larger block or across blocks. Finally, we return the updated memory and tag.

Algorithm 16 $(\widehat{v}, \widehat{j}) \leftarrow \mathrm{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}, \widehat{\mu}))$

1: $\widehat{j} = 1$
2: $\widehat{v} = \mathrm{NULL}$
3: $(\widehat{\omega},\ \widehat{ty}_1,\ \widehat{\alpha},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}_1, \mathrm{public}, \widehat{\alpha})) = \widehat{\sigma}(\widehat{l})$
4: **if** $(\widehat{bty} = \widehat{ty}_1) \wedge (\widehat{\mu} = 0) \wedge (\widehat{\alpha} = 1)$ **then**
5: $\quad \widehat{v} = \mathrm{DecodeVal}(\widehat{bty},\ 1, \widehat{\omega})$
6: **else if** $(\widehat{bty} = \widehat{ty}_1) \wedge (\widehat{\mu}\ \%\ \tau(\widehat{bty}) = 0) \wedge (\frac{\widehat{\mu}}{\tau(\widehat{bty})} < \widehat{\alpha})$ **then**
7: $\quad [\widehat{v}_0, ..., \widehat{v}_{\widehat{\alpha}-1}] = \mathrm{DecodeVal}(\widehat{bty},\ \widehat{\alpha}, \widehat{\omega})$
8: $\quad \widehat{n} = \frac{\widehat{\mu}}{\tau(\widehat{bty})}$
9: $\quad \widehat{v} = \widehat{v}_{\widehat{n}}$
10: **else**
11: $\quad \widehat{j} = 0$
12: $\quad \widehat{v} = \mathrm{GetBytes}((\widehat{l}, \widehat{\mu}), \widehat{bty}, \widehat{\sigma})$
13: **end if**
14: **return** $(\widehat{v}, \widehat{j})$

Algorithm 16 ($\mathrm{DerefPtr}$) is designed to dereference a value at a location in memory. It takes the memory, a type, and a location as input, and interprets the byte data at that location as a value of the given type. First, we look up the memory block associated with the location. In line 4, we check to see if this is a simple lookup by seeing if the location is of the correct type, the offset it 0, and there is only one element in the block. In line 6, we check to see if this is a simple lookup in an array block, where we have the correct type and the offset corresponds to an element of the array. For both of these cases, we can use Algorithm 8 to get the value(s) stored in the block and return the correct value. In line 10, we have a dereference that is not *well-aligned*, and so we set the tag to 0 and use Algorithm 19 to obtain a value of the anticipated type from the byte representation, although the value is likely garbage. Finally, we return the value and the tag.

Algorithm 17 ($\mathrm{DerefPtrHLI}$) is designed to dereference a pointer at a location in memory. It takes the

Algorithm 17 $([1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}], \widehat{j}) \leftarrow \mathrm{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty}*, (\widehat{l}, \widehat{\mu}))$

1: $(\widehat{\omega},\ \widehat{ty}_1,\ \widehat{\alpha},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}_1, \mathrm{public}, \widehat{\alpha})) = \widehat{\sigma}(\widehat{l})$
2: **if** $(\widehat{bty}* = \widehat{ty}_1) \wedge (\widehat{\mu} = 0) \wedge (\widehat{\alpha} = 1)$ **then**
3: $\quad [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}] = \mathrm{DecodePtr}(\widehat{bty}*,\ 1, \widehat{\omega})$
4: $\quad$ **return** $([1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}], 1)$
5: **else**
6: $\quad [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}] = \mathrm{GetBytes}((\widehat{l}, \widehat{\mu}), \widehat{bty}*, \widehat{\sigma})$
7: $\quad$ **return** $([1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}], 0)$
8: **end if**

memory, a type, and a location as input, and interprets the byte representation from that location as a pointer data structure. For pointers, we currently assert that the location we are grabbing the pointer from must be at the beginning of the block, otherwise the pointer data is not aligned, as we do not currently support arrays of pointers. The behavior of this algorithm is similar to the previous algorithm, just obtaining a pointer data structure instead of a value. It returns the pointer data structure and a tag indicating whether the access was *well-aligned* or not.

Algorithm 18 ($\mathrm{SetBytes}$) is designed to store a value into a location in memory that may not be *well-aligned*. It takes as input the location, the type to encode the byte representation of the value, the value, and the memory, and returns the updated memory. We first remove the current block referred to by the location from memory, then obtain the byte representation of the value based on the type of the value - line 4 handling non-pointer values, and line 6 handling pointer values (i.e., pointer data structures). In line 8, we are finding the size of the location (in bytes) that is left after the offset given by the location we received as input. We then check this size against the size of the value we are trying to store into memory. In line 9, we are checking if the value's byte representation will fit somewhere in the middle of the block, with bytes at the beginning and at the end. We then take the first portion of what was in the block up until the offset and add the byte representation we are storing to it, and at the end add on whatever bytes were left beyond the chunk that we replaced. We then store this final byte representation back into this location.

In line 12, we have the the byte representation of the value we obtained should fit within this memory block, and will take up the space from the offset until the end of the block. Thus, we take the first part of the bytes currently stored in the block up until the offset, and replace the rest with our byte representation of the given value, storing it into memory. In line 15, we are entering the portion where we are overflowing from this block into block(s) that come after it. We first store here whatever portion of our byte representation of the value that can be stored within this block, then obtain the remaining portion and length that we still

Algorithm 18 $(\widehat{\sigma}_f) \leftarrow \mathrm{SetBytes}((\widehat{l}, \widehat{\mu}), \widehat{ty}, \widehat{v}, \widehat{\sigma})$

1: $\widehat{\sigma}_f[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{ty}_1,\ \widehat{\alpha},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}_1, \mathrm{public}, \widehat{\alpha}))] = \widehat{\sigma}$
2: $\widehat{\omega}_v = \mathrm{NULL}$
3: **if** $\widehat{ty} = \widehat{bty}$ **then**
4: $\quad \widehat{\omega}_v = \mathrm{EncodeVal}(\widehat{ty}, \widehat{v})$
5: **else**
6: $\quad \widehat{\omega}_v = \mathrm{EncodePtr}(\widehat{ty}, \widehat{v})$
7: **end if**
8: $\widehat{n}_l = \tau(\widehat{ty}_1) \cdot \widehat{\alpha} - \widehat{\mu}$
9: **if** $\tau(\widehat{ty}) < \widehat{n}_l - 1$ **then**
10: $\quad \widehat{\omega}_f = \widehat{\omega}[0 : \widehat{\mu} - 1] + \widehat{\omega}_v + \widehat{\omega}[\widehat{\mu} + \tau(\widehat{ty}) :]$
11: $\quad \widehat{\sigma}_f = \widehat{\sigma}_f[\widehat{l} \rightarrow (\widehat{\omega}_f,\ \widehat{ty}_1,\ \widehat{\alpha},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}_1, \mathrm{public}, \widehat{\alpha}))]$
12: **else if** $\tau(\widehat{ty}) = \widehat{n}_l - 1$ **then**
13: $\quad \widehat{\omega}_f = \widehat{\omega}[0 : \widehat{\mu} - 1] + \widehat{\omega}_v$
14: $\quad \widehat{\sigma}_f = \widehat{\sigma}_f[\widehat{l} \rightarrow (\widehat{\omega}_f,\ \widehat{ty}_1,\ \widehat{\alpha},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}_1, \mathrm{public}, \widehat{\alpha}))]$
15: **else**
16: $\quad \widehat{\omega}_f = \widehat{\omega}[0 : \widehat{\mu} - 1] + \widehat{\omega}_v[0 : \widehat{n}_l - 1]$
17: $\quad \widehat{\sigma}_f = \widehat{\sigma}_f[\widehat{l} \rightarrow (\widehat{\omega}_f,\ \widehat{ty}_1,\ \widehat{\alpha},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}_1, \mathrm{public}, \widehat{\alpha}))]$
18: $\quad \widehat{\omega}_v = \widehat{\omega}_v[\widehat{n}_l :]$
19: $\quad \widehat{n}_v = \tau(\widehat{ty}) - \widehat{n}_l$
20: $\quad$ **while** $\widehat{n}_v > 0$ **do**
21: $\quad\quad \widehat{l} = \mathrm{GetBlock}(\widehat{l})$
22: $\quad\quad \widehat{\sigma}_f[\widehat{l} \rightarrow (\widehat{\omega}_c,\ \widehat{ty}_c,\ \widehat{\alpha}_c,\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}_c, \mathrm{public}, \widehat{\alpha}_c))] = \widehat{\sigma}_f$
23: $\quad\quad \widehat{n}_c = \tau(\widehat{ty}_c) \cdot \widehat{\alpha}_c$
24: $\quad\quad$ **if** $\widehat{n}_v < \widehat{n}_c$ **then**
25: $\quad\quad\quad \widehat{\omega}_f = \widehat{\omega}_v + \widehat{\omega}_c[\widehat{n}_v :]$
26: $\quad\quad$ **else if** $\widehat{n}_v = \widehat{n}_c$ **then**
27: $\quad\quad\quad \widehat{\omega}_f = \widehat{\omega}_v$
28: $\quad\quad$ **else**
29: $\quad\quad\quad \widehat{\omega}_f = \widehat{\omega}_v[0 : \widehat{n}_c - 1]$
30: $\quad\quad\quad \widehat{\omega}_v = \widehat{\omega}_v[\widehat{n}_c :]$
31: $\quad\quad$ **end if**
32: $\quad\quad \widehat{n}_v = \widehat{n}_v - \widehat{n}_c$
33: $\quad\quad \widehat{\sigma}_f = \widehat{\sigma}_f[\widehat{l} \rightarrow (\widehat{\omega}_f,\ \widehat{ty}_c,\ \widehat{\alpha}_c,\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}_c, \mathrm{public}, \widehat{\alpha}_c))]$
34: $\quad$ **end while**
35: **end if**
36: **return** $\widehat{\sigma}_f$

need to store. In line 20, we enter a loop that will iteratively store the rest into memory. It first obtains the next sequential block, then stores whatever it can into that block, appropriately keeping any bytes of data that were there if what we are storing will not take up the entire block. It continues this process until the entire byte representation for the value has been stored into memory.

Algorithm 19 (GetBytes) is designed to take the byte representation from a specific location and interpret it as the given type. It takes as input the location, the type, and the memory, and returns a value of the expected type. It is important to note here that when we are trying to read a pointer data structure from a location that is not aligned, this function will automatically ignore the bytes for the number of locations and the tag and assume they are both 1, as we can only have a single location for any pointer in Vanilla C.

Algorithm 19 $(\widehat{v}) \leftarrow \mathrm{GetBytes}((\widehat{l}, \widehat{\mu}), \widehat{ty}, \widehat{\sigma})$

```
1: (ω̂, tŷ_1, α̂, PermL(Freeable, tŷ_1, public, α̂)) = σ̂(l̂)
2: v̂_f = NULL
3: n̂_c = τ(tŷ_1) · α̂ − μ̂
4: if tŷ = b̂ty then
5:     if (τ(tŷ) ≤ n̂_c − 1) then
6:         ω̂_v = ω̂[μ̂ : μ̂ + τ(tŷ)]
7:     else
8:         ω̂_v = ω̂[μ̂ :]
9:         n̂_v = τ(tŷ) − n̂_c
10:        while (n̂_v > 0) do
11:            l̂ = GetBlock(l̂)
12:            (ω̂_c, tŷ_c, α̂_c, PermL(Freeable, tŷ_c, public, α̂_c)) = σ̂(l̂)
13:            n̂_c = τ(tŷ_c) · α̂_c
14:            ω̂_v = ω̂_v + ω̂_c[0 : min(n̂_v, n̂_c) − 1]
15:            n̂_v = n̂_v − n̂_c
16:        end while
17:    end if
18:    v̂_f = DecodeVal(tŷ, 1, ω̂_v)
19: else if (tŷ = b̂ty∗) then
20:    if (τ(int) · 5 ≤ n_c − 1) then
21:        ω̂_v = ω̂[μ̂ : μ̂ + n̂_v]
22:    else
23:        ω̂_v = ω̂[μ̂ :]
24:        n̂_v = τ(int) · 5 − n̂_c
25:        while (n̂_v > 0) do
26:            l̂ = GetBlock(l̂)
27:            (ω̂_c, tŷ_c, α̂_c, PermL(Freeable, tŷ_c, public, α̂_c)) = σ̂(l̂)
28:            n̂_c = τ(tŷ_c) · α̂_c
29:            ω̂_v = ω̂_v + ω̂_c[0 : min(n̂_v, n̂_c) − 1]
30:            n̂_v = n̂_v − n̂_c
31:        end while
32:    end if
33:    [α̂_1, [(l̂_1, μ̂_1)], [ĵ_1], î_1] = DecodePtr(tŷ, 1, ω̂_v)
34:    v̂_f = [1, [(l̂_1, μ̂_1)], [1], î_1]
35: end if
36: return v̂_f
```

Algorithm 20 $((\widehat{l_{\widehat{n}+1}}) \leftarrow \mathrm{GetBlock}(\widehat{l_{\widehat{n}}})$

```
1: return l̂_{n̂+1}
```

Algorithm 20 (GetBlock) is designed to select the identifier for the next block in memory after the current one. It takes a memory block identifier as input, and returns the next sequential memory block. We chose to formalize this as simply grabbing the next higher block, however, our formalization will work with any implementation of this (e.g., selecting a random block, looping through only allocated blocks, etc.).

Algorithm 21 (GetLocation) is designed to obtain the next location that is $\widehat{n}$ bytes ahead of the current location $(\widehat{l}, \widehat{\mu})$. It takes the current location, the number of bytes to get to the next location, and memory as

Algorithm 21 $((\widehat{l}_f, \widehat{\mu}_f), \widehat{j}) \leftarrow \mathrm{GetLocation}((\widehat{l}, \widehat{\mu}), \widehat{n}, \widehat{\sigma})$

```
1: (ω̂_1, t̂y_1, α̂_1, PermL(perm, t̂y_1, public, α̂_1)) = σ̂(l̂)
2: ĵ = 1
3: if (τ(t̂y_1) ≠ n̂) ∨ (μ̂ % τ(t̂y_1) ≠ 0) then
4:     ĵ = 0
5: end if
6: if (n̂ < τ(t̂y_1) · α̂ − μ̂) then
7:     (l̂_f, μ̂_f) = (l̂, μ̂ + n̂)
8: else
9:     n̂_2 = n̂ − τ(t̂y_1) · α̂ − μ̂
10:    l̂_1 = l̂
11:    while n̂_2 ≥ 0 do
12:        (l̂_2) = GetBlock(l̂_1)
13:        if n̂_2 = 0 then
14:            (l̂_f, μ̂_f) = (l̂_2, 0)
15:            n̂_2 = −1
16:            if (ĵ = 1) ∧ (τ(t̂y_2) ≠ n̂) then
17:                ĵ = 0
18:            end if
19:        else
20:            ĵ = 0
21:            (ω̂_2, t̂y_2, α̂_2, PermL(perm, t̂y_2, public, α̂_2)) = σ̂(l̂_2)
22:            if (n̂_2 < τ(t̂y_2) · α̂_2) then
23:                (l̂_f, μ̂_f) = (l̂_2, n̂_2)
24:                n̂_2 = −1
25:            else
26:                n̂_2 = n̂_2 − τ(t̂y_2) · α̂_2
27:            end if
28:            l̂_1 = l̂_2
29:        end if
30:    end while
31: end if
32: return ((l̂_f, μ̂_f), ĵ)
```

input and return the next location and a tag indicating whether the next location is *well-aligned* or not. We first look up the current memory block and initialize the tag as 1. Then, we check whether the next location is not *well-aligned* with the current block. If it is not, then it cannot be *well-aligned* with the location we return, either. Next, we check whether the next location is still within this block. If it is, we update the offset and return. If it is not, we must obtain the next memory block and keep checking. To do this, we first calculate the number of bytes that we need to move forward beyond the current block, and we initialize the memory block identifier we are looking beyond as the original memory block identifier. Then we loop until we have finally found the next location to return. In each loop iteration, we grab the next block using Algorithm 20, and look up the location in memory. If the number of bytes we have left is 0, then our next location is at the start of the block. We perform one last check to see if the current location is *well-aligned* with the new location (which is possible if this is the start of the next block immediately after the one we were in). Otherwise, we set the

tag as 0 and reset the block we are looking beyond to be the new block. Then, we check whether our new location is within this block. If it is not, we decrease the number of bytes we need to move forward by the number of bytes within this block, and repeat the loop.

Algorithm 22 $\widehat{\overline{ty}} \leftarrow \mathrm{GetFunTypeList}(\widehat{\overline{p}})$

1: $\widehat{\overline{ty}} = [\,]$
2: **while** $\widehat{\overline{p}} \neq \mathrm{void}$ **do**
3: **if** $\widehat{\overline{p}} = \widehat{ty}$ **then**
4: $\widehat{\overline{ty}} = \widehat{ty} :: \widehat{\overline{ty}}$
5: $\widehat{\overline{p}} = \mathrm{void}$
6: **else if** $\widehat{\overline{p}} = \widehat{\overline{p}}', \widehat{ty}$ **then**
7: $\widehat{\overline{ty}} = \widehat{ty} :: \widehat{\overline{ty}}$
8: $\widehat{\overline{p}} = \widehat{\overline{p}}'$
9: **end if**
10: **end while**
11: **return** $\widehat{\overline{ty}}$

Algorithm 22 ($\mathrm{GetFunTypeList}$) is designed to obtain the function input type from its parameter list, taking as input a parameter list and returning the corresponding type list.

Algorithm 23 $\widehat{s} \leftarrow \mathrm{GetFunParamAssign}(\widehat{\overline{p}}, \widehat{\overline{e}})$

Require: $\mathrm{length}(\widehat{\overline{p}}) = \mathrm{length}(\widehat{\overline{e}})$
1: $\widehat{s} = \mathrm{skip}$
2: **while** $\widehat{\overline{p}} \neq \mathrm{void}$ **do**
3: **if** $(\widehat{\overline{p}} = \widehat{ty}\ \widehat{var}) \wedge (\widehat{\overline{e}} = \widehat{e})$ **then**
4: $\widehat{s} = \widehat{ty}\ \widehat{var} = \widehat{e}; \widehat{s}$
5: $\widehat{\overline{p}} = \mathrm{void}$
6: $\widehat{\overline{e}} = \mathrm{void}$
7: **else if** $(\widehat{\overline{p}} = \widehat{\overline{p}}', \widehat{ty}) \wedge (\widehat{\overline{e}} = \widehat{\overline{e}}', \widehat{e})$ **then**
8: $\widehat{s} = \widehat{ty}\ \widehat{var} = \widehat{e}; \widehat{s}$
9: $\widehat{\overline{p}} = \widehat{\overline{p}}'$
10: $\widehat{\overline{e}} = \widehat{\overline{e}}'$
11: **end if**
12: **end while**
13: **return** $\widehat{s}$

Algorithm 23 ($\mathrm{GetFunParamAssign}$) is designed to create assignment statements for the parameters of a function, taking as input a parameter list and expression list and returning a set of assignment statements to assign the expressions to the corresponding parameters.

Algorithm 24 ($\mathrm{ReadOOB}$) is designed to read a value of the given type from memory as though it was at index $\widehat{i}$ of the array in memory block $\widehat{l}$. It takes as input the out of bounds index $\widehat{i}$, the number of values in the array $\widehat{n}$, the memory block of the array data $\widehat{l}$, the type of elements in the array $\widehat{bty}$, and memory $\widehat{\sigma}$. It then iterates through memory until it finds the bytes that would be at index $\widehat{i}$ and decodes them as the expected type $\widehat{bty}$ to obtain value $\widehat{v}$. As the algorithm iterates through memory, if all locations we iterate over are of the

Algorithm 24 $(\widehat{v},\ \widehat{j}) \leftarrow \mathrm{ReadOOB}(\widehat{i},\ \widehat{n},\ \widehat{l},\ \widehat{bty},\ \widehat{\sigma})$

```
1: n̂_v = τ(b̂ty)
2: n̂_b = (î − n̂) · n̂_v
3: ĵ = 1
4: ω̂_v = [ ]
5: while (n̂_b > 0) ∨ (n̂_v > 0) do
6:     l̂ = GetBlock(l̂)
7:     (ω̂, tŷ_1, α̂, PermL(Freeable, tŷ_1, public, α̂)) = σ̂(l̂)
8:     if (tŷ_1 ≠ b̂ty) then
9:         ĵ = 0
10:    end if
11:    if (n̂_b < τ(tŷ_1) · α̂) then
12:        ω̂_v = ω̂_v :: ω̂[n̂_b : min(n̂_b + n̂_v, τ(tŷ_1) · α̂ − 1)]
13:        n̂_v = n̂_v − τ(tŷ_1) · α̂ + n̂_b
14:    end if
15:    n̂_b = max(0, n̂_b − τ(tŷ_1) · α̂)
16: end while
17: v̂ = DecodeVal(b̂ty, 1, ω̂_v)
18: return (v̂, ĵ)
```

same type as the expected type, and the location we are reading the value from is also the expected type, then it will return tag 1, indicating that our read was *well-aligned*. Otherwise, tag 0 will be returned. We currently only show the algorithm handling overshooting in the positive direction, however, it can trivially extended to grab the previous blocks and iterate backwards through memory to handle a negative index as well.

Algorithm 25 (WriteOOB) is designed to store a value of the given type from memory as though it was at index $\widehat{i}$ of the array in memory block $\widehat{l}$. It takes as input the value to write in memory $\widehat{v}$, the out of bounds index $\widehat{i}$, the number of values in the array $\widehat{n}$, the memory block of the array data $\widehat{l}$, the type of elements in the array $\widehat{bty}$, and memory $\widehat{\sigma}$. It then iterates through memory until it finds the position that would be for index $\widehat{i}$, encodes value $\widehat{v}$ as the expected type, and places its byte representation into memory starting at that position. As the algorithm iterates through memory, if all locations we iterate over are of the same type as the expected type, and the location we are writing the value to is also the expected type, then it will return tag 1, indicating that our read was *well-aligned*. Otherwise, tag 0 will be returned. We currently only show the algorithm handling overshooting in the positive direction, however, it can trivially extended to grab the previous blocks and iterate backwards through memory to handle a negative index as well.

Algorithm 26 (Free) corresponds to conventional memory deallocation when we call free to deallocate memory associated with some pointer. In particular, on input location $\widehat{l}$, we first check whether the location corresponds to memory that can be deallocated using Algorithm 27 (CheckFreeable). If CheckFreeable returns 1, we will mark location $\widehat{l}$ as unavailable. Otherwise, calling Free has no effect on the state of memory.

Algorithm 25 $(\widehat{\sigma},\ \widehat{j}) \leftarrow \mathrm{WriteOOB}(\widehat{v},\ \widehat{i},\ \widehat{n},\ \widehat{l},\ \widehat{bty},\ \widehat{\sigma})$

$\widehat{\omega}_v = \mathrm{EncodeVal}(\widehat{bty}, \widehat{v})$
$\widehat{n}_b = (\widehat{i} - \widehat{n}) \cdot \tau(\widehat{bty})$
$\widehat{j} = 1$
while $(\widehat{n}_b > 0) \vee (|\widehat{\omega}_v| > 0)$ **do**
  $\widehat{l} = \mathrm{GetBlock}(\widehat{l})$
  $\widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{ty}_1,\ \widehat{\alpha},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}_1, \mathrm{public}, \widehat{\alpha}))] = \widehat{\sigma}$
  if $(\widehat{ty}_1 \neq \widehat{bty})$ **then**
    $\widehat{j} = 0$
  end if
  if $(\widehat{n}_b < \tau(\widehat{ty}_1) \cdot \widehat{\alpha})$ **then**
    if $(|\widehat{\omega}_v| > \tau(\widehat{ty}_1) \cdot \widehat{\alpha} - \widehat{n}_b)$ **then**
      $\widehat{\omega}_1 = \widehat{\omega}[0 : \widehat{n}_b] + \widehat{\omega}_v + \widehat{\omega}[|\widehat{\omega}_v| + \widehat{n}_b :]$
      $\widehat{\omega}_v = [\,]$
    else if $(|\widehat{\omega}_v| = \tau(\widehat{ty}_1) \cdot \widehat{\alpha} - \widehat{n}_b)$ **then**
      $\widehat{\omega}_1 = \widehat{\omega}[0 : \widehat{n}_b] + \widehat{\omega}_v$
      $\widehat{\omega}_v = [\,]$
    else
      $\widehat{\omega}_1 = \widehat{\omega}[0 : \widehat{n}_b] + \widehat{\omega}_v[0 : \tau(\widehat{ty}_1) \cdot \widehat{\alpha} - \widehat{n}_b - 1]$
      $\widehat{\omega}_v = \widehat{\omega}_v[\tau(\widehat{ty}_1) \cdot \widehat{\alpha} - \widehat{n}_b :]$
    end if
    $\widehat{\sigma} = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega}_1,\ \widehat{ty}_1,\ \widehat{\alpha},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}_1, \mathrm{public}, \widehat{\alpha}))]$
  end if
  $\widehat{n}_b = max(0, \widehat{n}_b - \tau(\widehat{ty}_1) \cdot \widehat{\alpha})$
end while
return $(\widehat{\sigma},\ \widehat{j})$

Algorithm 27 ($\mathrm{CheckFreeable}$) follows the behavior expected of free: if the location was properly allocated via a call to malloc, it is de-allocatable for the purpose of this function. In particular, the default location $l_{default}$ that corresponds to uninitialized pointers is not de-allocatable (and freeing such a pointer has no effect); similarly memory associated with statically declared variables is not de-allocatable via this mechanism (and freeing it here also has no effect). Thus, if $\mathrm{CheckFreeable}$ returns 1, we will proceed to deallocate a location, otherwise, there will be no effect on the state of memory as we cannot safely perform the deallocation operation.

Algorithm 28 ($\mathrm{InputValue}$) is designed to obtain a single value from a specific input party's input file. We first open the input file for that party in read mode, then iterate through the file to find the desired variable. We then check if there is more than one value - if there is, we take only the first value; otherwise, we return the value as we read it in.

Algorithm 29 ($\mathrm{InputArray}$) is designed to obtain a list of values from a specific input party's input file. We first open the input file for that party in read mode, then iterate through the file to find the desired variable. We then check if the length of the array given is longer than the desired length, and only take the desired length if it is. Otherwise, we return the array as we read it in.

Algorithm 26 $(\widehat{\sigma}_f) \leftarrow \text{Free}(\widehat{\gamma}, \widehat{l}, \widehat{\sigma})$

$(\widehat{\omega},\ \widehat{bty*},\ 1,\ \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) = \widehat{\sigma}(\widehat{l})$
$[1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ i] = \text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega})$
if $\text{CheckFreeable}(\widehat{\gamma}, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{\sigma})$ **then**
    $\widehat{\sigma}_1[\widehat{l}_1 \rightarrow (\widehat{\omega},\ \widehat{ty},\ 1,\ \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, 1))] = \widehat{\sigma}$
    $\widehat{\sigma}_f = \widehat{\sigma}_1[\widehat{l}_1 \rightarrow (\widehat{\omega},\ \widehat{ty},\ 1,\ \text{PermL}(\text{None}, \widehat{ty}, \text{public}, 1))]$
else
    $\widehat{\sigma}_f = \widehat{\sigma}$
end if
return $(\widehat{\sigma}_f)$

Algorithm 27 $\widehat{j} \leftarrow \text{CheckFreeable}(\widehat{\gamma}, [(\widehat{l}, \widehat{\mu})], [1], \widehat{\sigma})$

if $(\widehat{l}_{default} = \widehat{l}) \vee (\widehat{\mu} \neq 0)$ **then**
    return 0
end if
for all $x \in \gamma$ **do**
    $(l_x, ty_x) = \gamma(x)$
    if $(l_x = \widehat{l})$ **then**
        return 0
    else if $(ty_x = a \text{ const } bty*)$ **then**
        $(\omega, ty_x, 1, \text{PermL}(\text{Freeable}, ty_x, a, 1)) = \sigma(l_x)$
        $[1, [(l_1, 0)], [1], 1] = \text{DecodePtr}(ty_x, 1, \omega)$
        if $(l_1 = \widehat{l})$ **then**
            return 0
        end if
    end if
end for
return 1

Algorithm 30 (OutputValue) is designed to write a value out to an output file for the output party $\widehat{n}_p$. We first open the output file for that party in append mode, as we want to allow multiple uses of the output function throughout the program. Next, we write the variable name and the value out to the file, then close the file and return from this algorithm.

Algorithm 31 (OutputArray) is designed to write the given array to an output file for the output party $\widehat{n}_p$. We first open the output file for that party in append mode, as we want to allow multiple uses of the output function throughout the program. Next, we write the variable name and its list of values out to the file, then close the file and return from this algorithm.

Algorithm 28 $(\widehat{n}) \leftarrow \text{InputValue}(\widehat{x},\ \widehat{n}_p)$

```
1: n̂ = NULL
2: file = open(inputFiles[n̂_p − 1], r)
3: for all line ∈ file do
4:     (x̂_f, v̂) = line.split(=)
5:     if x̂_f = x̂ then
6:         if v̂ = n̂̂ then
7:             n̂ = n̂̂[0]
8:         else
9:             n̂ = v̂
10:        end if
11:        break
12:    end if
13: end for
14: file.close()
15: return n̂
```

Algorithm 29 $(\widehat{\widehat{n}}) \leftarrow \text{InputArray}(\widehat{x}, \widehat{n}_p, \widehat{n})$

```
1: n̂̂ = [ ]
2: file = open(inputFiles[n̂_p − 1], r)
3: for all line ∈ file do
4:     (x̂_f, n̂̂_f) = line.split(=)
5:     if x̂_f = x̂ then
6:         if n̂̂_f.length() > n̂_p then
7:             n̂̂ = n̂̂_f[0 : n̂_p − 1]
8:         else
9:             n̂̂ = n̂̂_f
10:        end if
11:        break
12:    end if
13: end for
14: file.close()
15: return n̂̂
```

Algorithm 30 $\text{NULL} \leftarrow \text{OutputValue}(\widehat{x},\ \widehat{n}_p,\ \widehat{n})$

```
1: file = open(outputFiles[n̂_p − 1], a)
2: file.write(x̂ = n̂)
3: file.close()
4: return NULL
```

Algorithm 31 $\text{NULL} \leftarrow \text{OutputArray}(\widehat{x},\ \widehat{n}_p,\ \widehat{\widehat{n}})$

```
1: file = open(outputFiles[n̂_p − 1], a)
2: file.write(x̂ = n̂̂)
3: file.close()
4: return NULL
```

3.1.6 Basic SMC^2 Algorithms

32	Extract_variables	$::= f : (s, s) \rightarrow x_{list}$
33	InitializeVariables	$::= f : (\overline{x},\ \gamma,\ \sigma,\ \text{acc}) \rightarrow (\gamma, \sigma)$
34	RestoreVariables	$::= f : (\overline{x},\ \gamma,\ \sigma,\ \text{acc}) \rightarrow \sigma$
35	ResolveVariables	$::= f : (\overline{x},\ \gamma,\ \sigma,\ \text{acc},\ res) \rightarrow \sigma$
36	PermL	$::= f : (perm,\ ty,\ a,\ n) \rightarrow \overline{perm}$
37	PermL_Fun	$::= f : (\text{public}) \rightarrow \overline{perm}$
38	$\phi()$	$::= f : () \rightarrow l$
39	τ	$::= f : (ty) \rightarrow n$
40	Label	$::= f : (e, \gamma) \rightarrow a$
41	EncodeVal	$::= f : (ty, v) \rightarrow \omega$
42	DecodeVal	$::= f : (a\ bty, n, \omega) \rightarrow v$
43	EncodePtr	$::= f : (ty, [\alpha, \overline{l}, \overline{j}, i]) \rightarrow \omega$
44	DecodePtr	$::= f : (ty, \alpha, \omega) \rightarrow [\alpha, \overline{l}, \overline{j}, i]$
45	EncodeFun	$::= f : (s, j, \overline{p}) \rightarrow \omega$
46	DecodeFun	$::= f : (s, j, \overline{p}) \rightarrow \omega$
47	UpdateVal	$::= f : (\sigma, l, v, a\ bty) \rightarrow \sigma$
48	UpdatePtr	$::= f : (\sigma, (l, \mu), [\alpha, \overline{l}, \overline{j}, i], ty) \rightarrow (\sigma, j)$
49	UpdatePrivPtr	$::= f : (\sigma, [\alpha, \overline{l}, \overline{j}, i], [\alpha, \overline{l}, \overline{j}, i], ty) \rightarrow (\sigma, j)$
50	UpdatePriv	$::= f : (\sigma, \alpha, \overline{l}, \overline{j}, ty, v) \rightarrow (\sigma, j)$
51	UpdateOffset	$::= f : (\sigma, (l, \mu), v, ty) \rightarrow (\sigma, j)$
52	GetFunTypeList	$::= f : (\overline{p}) \rightarrow \overline{ty}$
53	GetFunParamAssign	$::= f : (\overline{p}, \overline{e}) \rightarrow s$
54	CheckPublicEffects	$::= f : (s, x, \gamma, \sigma) \rightarrow j$
55	Cast	$::= f : (a, ty, n) \rightarrow n$
56	encrypt	$::= f : (n) \rightarrow n$
57	ReadOOB	$::= f : (i,\ n,\ l,\ ty,\ \sigma) \rightarrow (v,\ j)$
58	WriteOOB	$::= f : (v, i, n, l, ty, \sigma) \rightarrow (\sigma, j)$

59	InputValue	$::= f : (x, n) \rightarrow n$
60	InputArray	$::= f : (x, n, n) \rightarrow \overline{n}$
61	OutputValue	$::= f : (x, n, n) \rightarrow \text{NULL}$
62	OutputArray	$::= f : (x, n, \overline{n}) \rightarrow \text{NULL}$
63	IncrementList	$::= f : (\bar{l}, n, \sigma) \rightarrow (\bar{l}, j)$
64	GetLocation	$::= f : ((l, \mu), n, \sigma) \rightarrow ((l, \mu), j)$
65	SetBytes	$::= f : ((l, \mu), ty, v, \sigma) \rightarrow \sigma$
66	GetBytes	$::= f : ((l, \mu), ty, \sigma) \rightarrow v$
68	GetBlock	$::= f : (l) \rightarrow l$
69	GetIndirection	$::= f : (*) \rightarrow n$
70	DerefPtr	$::= f : (\sigma, ty, (l, \mu)) \rightarrow (v, j)$
71	DerefPtrHLI	$::= f : (\sigma, ty, (l, \mu)) \rightarrow ([\alpha, \bar{l}, \bar{j}, i], j)$
72	DerefPrivPtr	$::= f : (\alpha, \bar{l}, \bar{j}, ty, \sigma) \rightarrow ((\alpha, \bar{l}, \bar{j}), j)$
73	Retrieve_vals	$::= f : (\alpha, \bar{l}, \bar{j}, ty, \sigma) \rightarrow (v, j)$
74	Free	$::= f : (\sigma,\ l, \gamma) \rightarrow \sigma$
75	PFree	$::= f : (\gamma, \sigma, l) \rightarrow (\sigma, \bar{l}, \bar{j})$
76	CheckFreeable	$::= f : (\gamma, \bar{l}, \bar{j}, \sigma) \rightarrow j$
77	UpdatePointerLocations	$::= f : (\sigma, \bar{l}, \bar{j}, l, j) \rightarrow \sigma$
78	CondAssign	$::= f : ([\alpha,\ \bar{l},\ \bar{j}],\ [\alpha,\ \bar{l},\ \bar{j}],\ n) \rightarrow [\alpha,\ \bar{l},\ \bar{j}]$

Our helper algorithms for use in Private If Else (shown in Figure 3.15c) are defined in Algorithms 32, 33, 34, and 35. These are called from within the Private If Else rule, and are not part of the SMC^2 source code.

Algorithm 32 $(\overline{x}_{mod}) \leftarrow \text{ExtractVariables}(s_1,\ s_2)$

```
1: x̄_local = [ ]
2: x̄_mod = [ ]
3: for all s ∈ {s_1; s_2} do
4:     if ((s = ty x) ∨ (s = ty x[e]) ∨ (s = ty x[e][e])) then
5:         x̄_local.append(x)
6:     else if ((s = x = e) ∧ (¬x̄_local.contains(x))) then
7:         x̄_mod = x̄_mod ∪ [x]
8:     end if
9: end for
10: return x̄_mod
```

Algorithm 32 (Extract_variables) is used to iterate over the **`then`** and **`else`** branch statements, creating a list of variables that are assigned to within either branch (excluding any variables local to either branch).

This list is returned, enabling us to make copies of these variables and restore and resolve the true values appropriately.

Algorithm 33 $(\gamma_f,\ \sigma_f) \leftarrow \text{InitializeVariables}(\overline{x},\ \gamma,\ \sigma,\ \text{acc})$

1: $\gamma_1 = [\,]$
2: $\sigma_1 = [\,]$
3: **for all** $x \in \overline{x}$ **do**
4: $\quad (l, ty) = \gamma(x)$
5: $\quad l_t = \phi(temp)$
6: $\quad l_e = \phi(temp)$
7: $\quad \gamma_1 = \gamma_1[x_{then_\text{acc}} \rightarrow (l_t, ty)][x_{else_\text{acc}} \rightarrow (l_e, ty)]$
8: $\quad$ **if** $(ty = \text{private } bty)$ **then**
9: $\quad\quad (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) = \sigma(l)$
10: $\quad\quad \sigma_2 = \sigma_1[l_t \rightarrow (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)]$
11: $\quad\quad \sigma_3 = \sigma_2[l_e \rightarrow (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)]$
12: $\quad\quad \sigma_1 = \sigma_3$
13: $\quad$ **else if** $(ty = \text{private } bty*)$ **then**
14: $\quad\quad (\omega,\ \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) = \sigma(l)$
15: $\quad\quad \sigma_2 = \sigma_1[l_t \rightarrow (\omega,\ \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))]$
16: $\quad\quad \sigma_3 = \sigma_2[l_e \rightarrow (\omega,\ \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))]$
17: $\quad\quad \sigma_1 = \sigma_3$
18: $\quad$ **else if** $(ty = \text{private const } bty*)$ **then**
19: $\quad\quad (\omega,\ \text{private const} bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) = \sigma(l)$
20: $\quad\quad \sigma_2 = \sigma_1[l_t \rightarrow (\omega,\ \text{private const} bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))]$
21: $\quad\quad \sigma_3 = \sigma_2[l_e \rightarrow (\omega,\ \text{private const} bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))]$
22: $\quad\quad [1,\ [(l', 0)],\ [1],\ 1] = \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega)]$
23: $\quad\quad l'_t = \phi(temp)$
24: $\quad\quad l'_e = \phi(temp)$
25: $\quad\quad (\omega, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) = \sigma_2(l')$
26: $\quad\quad \sigma_4 = \sigma_3[l'_t \rightarrow (\omega, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))]$
27: $\quad\quad \sigma_5 = \sigma_4[l'_e \rightarrow (\omega, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))]$
28: $\quad\quad \sigma_1 = \sigma_5$
29: $\quad$ **end if**
30: **end for**
31: $\gamma_f = \gamma :: \gamma_1$
32: $\sigma_f = \sigma :: \sigma_1$
33: **return** (γ_f, σ_f)

Algorithm 33 (InitializeVariables) ensures copies are made of all variables within the list x_{list}. This algorithm adds the new temporary variable names to the environment, then creates copies of the current state of memory for the original variable. The copies of memory are created by type - this is particularly important for array variables. Due to arrays being stored as a constant pointer to a block of data, which is standard in C, we need to perform multiple lookups and make copies of both the array pointer and data.

Algorithm 34 (RestoreVariables) ensures that all variables that are modified within either branch will have the value they contain at the end of the **`then`** branch stored, and then be reset to the value they had previous to the **`then`** branch.

Algorithm 35 (ResolveVariables) ensures the various variables that are modified within either branch

Algorithm 34 $(\sigma_2) \leftarrow \text{RestoreVariables}(\overline{x}, \gamma, \sigma, \text{acc})$

1: **for all** $x \in \overline{x}$ **do**
2: $(l_x, ty) = \gamma(x)$
3: $(l_t, ty) = \gamma(x_{then_\text{acc}})$
4: $(l_e, ty) = \gamma(x_{else_\text{acc}})$
5: **if** $(ty = \text{private } bty)$ **then**
6: $(\omega_x, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) = \sigma(l_x)$
7: $(\omega_e, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) = \sigma(l_e)$
8: $v_x = \text{DecodeVal}(\text{private } bty, 1, \omega_x)$
9: $v_e = \text{DecodeVal}(\text{private } bty, 1, \omega_e)$
10: $\sigma_1 = \text{UpdateVal}(\sigma, l_t, v_x)$
11: $\sigma_2 = \text{UpdateVal}(\sigma_1, l_x, v_e)$
12: **else if** $(ty = \text{private } bty*)$ **then**
13: $(\omega_x, \text{private } bty*, \alpha_x, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha_x)) = \sigma(l_x)$
14: $(\omega_e, \text{private } bty*, \alpha_e, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha_e)) = \sigma(l_e)$
15: $[\alpha_x,\ \overline{l}_x,\ \overline{j}_x,\ i] = \text{DecodePtr}(\text{private } bty*,\ \alpha_x,\ \omega_x)$
16: $[\alpha_e,\ \overline{l}_e,\ \overline{j}_e,\ i] = \text{DecodePtr}(\text{private } bty*,\ \alpha_e,\ \omega_e)$
17: $\sigma_1 = \text{UpdatePtr}(\sigma,\ (l_t, 0),\ [\alpha_x,\ \overline{l}_x,\ \overline{j}_x,\ i])$
18: $\sigma_2 = \text{UpdatePtr}(\sigma_1,\ (l_x, 0),\ [\alpha_e,\ \overline{l}_e,\ \overline{j}_e,\ i])$
19: **else if** $(ty = \text{private const } bty*)$ **then**
20: $(\omega_x,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) = \sigma(l_x)$
21: $[1,\ [(l'_x, 0)],\ [1],\ 1] = \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega_x)]$
22: $(\omega_t,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) = \sigma(l_t)$
23: $[1,\ [(l'_t, 0)],\ [1],\ 1] = \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega_x)]$
24: $(\omega_e,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) = \sigma(l_e)$
25: $[1,\ [(l'_e, 0)],\ [1],\ 1] = \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega_x)]$
26: $(\omega'_x, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) = \sigma(l'_x)$
27: $(\omega'_e, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) = \sigma(l'_e)$
28: $[v_0, ..., v_{n-1}] = \text{DecodeVal}(\text{private } bty, n, \omega'_x)$
29: $[v'_0, ..., v'_{n-1}] = \text{DecodeVal}(\text{private } bty, n, \omega'_e)$
30: $\sigma_1 = \text{UpdateVal}(\sigma, l'_t, [v_0, ..., v_{n-1}])$
31: $\sigma_2 = \text{UpdateVal}(\sigma_1, l'_x, [v'_0, ..., v'_{n-1}])$
32: **end if**
33: $\sigma = \sigma_2$
34: **end for**
35: **return** σ_2

will be appropriately resolved to their true values based on their type. When handling regular variables, it first looks up both copies of the variable, obtaining the resulting values for each branch, then performs a bit-wise operations over these two values and the result of the condition, x_{res}, to securely obtain the true value. For arrays, we handle the extra lookups needed as well as making sure each value within the array is properly resolved. Lastly, for pointer variables, we use Algorithm 78 (CondAssign) as defined for PICCO [22]) to assist us in creating the resulting list of locations the pointer can refer to and the corresponding set of tags to store the true location.

Algorithm 36 (PermL) creates a byte-wise permission list based on the given arguments for all memory blocks except function memory blocks. For basic types, this is as simple as getting the byte-length of the type and multiplying it by the number of elements. For pointer types, it needs to equate to the size of the pointer

Algorithm 35 $(\sigma_1) \leftarrow \text{ResolveVariables}(\overline{x}, \gamma, \sigma, \text{acc}, res_{\text{acc}})$

1: $(l_{res}, \text{private int}) = \gamma(res_{\text{acc}})$
2: $(\omega_{res}, \text{private int}, 1, \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1)) = \sigma(l_{res})$
3: $n_{res} = \text{DecodeVal}(\text{private } bty, 1, \omega_{res})$
4: **for all** $x \in \overline{x}$ **do**
5: $\quad (l_x, ty) = \gamma(x)$
6: $\quad (l_t, ty) = \gamma(x_{then_\text{acc}})$
7: $\quad$ **if** $(ty = \text{private } bty)$ **then**
8: $\quad\quad (\omega_t, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) = \sigma(l_t)$
9: $\quad\quad (\omega_x, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) = \sigma(l_x)$
10: $\quad\quad n_t = \text{DecodeVal}(\text{private } bty, 1, \omega_t)$
11: $\quad\quad n_x = \text{DecodeVal}(\text{private } bty, 1, \omega_x)$
12: $\quad\quad n_f = (n_{res} \wedge n_t) \vee (\neg\, n_{res} \wedge n_e)$
13: $\quad\quad \sigma_1 = \text{UpdateVal}(\sigma, l_x, n_f)$
14: $\quad$ **else if** $(ty = \text{private } bty*)$ **then**
15: $\quad\quad (\omega_t, \text{private } bty*, \alpha_t, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha_t)) = \sigma(l_t)$
16: $\quad\quad (\omega_x, \text{private } bty*, \alpha_x, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha_x)) = \sigma(l_x)$
17: $\quad\quad [\alpha_t,\ \overline{l}_t,\ \overline{j}_t,\ i] = \text{DecodePtr}(\text{private } bty*,\ \alpha_t,\ \omega_t)$
18: $\quad\quad [\alpha_x,\ \overline{l}_x,\ \overline{j}_x,\ i] = \text{DecodePtr}(\text{private } bty*,\ \alpha_x,\ \omega_x)$
19: $\quad\quad [\alpha_f, \overline{l}_f, \overline{j}_f] = \text{CondAssign}([\alpha_t, \overline{l}_t, \overline{j}_t, [\alpha_e, \overline{l}_e, \overline{j}_e], n_{res})$
20: $\quad\quad \sigma_1 = \text{UpdatePtr}(\sigma,\ (l_x, 0),\ [\alpha_f, \overline{l}_f, \overline{j}_f,\ i])$
21: $\quad$ **else if** $(ty = \text{private const } bty*)$ **then**
22: $\quad\quad (\omega_t,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) = \sigma(l_t)$
23: $\quad\quad [1,\ [(l'_t, 0)],\ [1],\ 1] = \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega_x)]$
24: $\quad\quad (\omega'_t, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) = \sigma(l'_t)$
25: $\quad\quad [v_{t_0}, ..., v_{t_{n-1}}] = \text{DecodeVal}(\text{private } bty, n, \omega'_t)$
26: $\quad\quad (\omega_x,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) = \sigma(l_x)$
27: $\quad\quad [1,\ [(l'_x, 0)],\ [1],\ 1] = \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega_x)]$
28: $\quad\quad (\omega'_x, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) = \sigma(l'_x)$
29: $\quad\quad [v_{x_0}, ..., v_{x_{n-1}}] = \text{DecodeVal}(\text{private } bty, n, \omega'_x)$
30: $\quad\quad$ **for all** $m \in [0, ..., n-1]$ **do**
31: $\quad\quad\quad v_{f_m} = (n_{res} \wedge v_{t_m}) \vee (\neg\, n_{res} \wedge v_{x_m})$
32: $\quad\quad$ **end for**
33: $\quad\quad \sigma_1 = \text{UpdateVal}(\sigma, l_x, [v_{f_0}, ..., v_{f_{n-1}}])$
34: $\quad$ **end if**
35: **end for**
36: $\sigma = \sigma_1$
37: **return** σ_1

data structure. Therefore, we have the size of a public int for the number of locations α and two public ints for each of the α memory block identifier and offset pairs, all of which have public privacy labels. Then we have an int for each of the α tags, which are public if the privacy label of the type is public, otherwise private, and another public int for the level of indirection. We then add these permission tuples for each byte into the permission list as we are calculating the size and ensuring the privacy labels are consistent with that of the type. Once complete, the permission list is returned.

Algorithm 37 (PermL_Fun) creates a permission list for a function memory. We leave this separate, and only create a single tuple for this block, which, if overshot into, will be read as being for all bytes.

Algorithm 38 defines how new memory block identifiers are obtained - each party will have a counter that

Algorithm 36 $\overline{perm} \leftarrow \text{PermL}(perm, ty, a, \alpha)$

```
1: perm̄ = [ ]
2: if ty = a bty then
3:     for all i ∈ {0...τ(ty) · α − 1} do
4:         perm̄ = (i, a, perm) :: perm̄
5:     end for
6: else if (ty = a bty∗) ∨ (ty = const a bty∗) then
7:     m1 = τ(public int) + τ(public int) · 2 · α
8:     for all i ∈ {0...m1 − 1} do
9:         perm̄ = (i, public, perm) :: perm̄
10:    end for
11:    m2 = τ(a int) · α
12:    for all i ∈ {m1...m1 + m2 − 1} do
13:        perm̄ = (i, a, perm) :: perm̄
14:    end for
15:    m3 = τ(public int)
16:    for all i ∈ {m1 + m2...m1 + m2 + m3 − 1} do
17:        perm̄ = (i, public, perm) :: perm̄
18:    end for
19: end if
20: return perm̄
```

Algorithm 37 $\overline{perm} \leftarrow \text{PermL_Fun}(\text{public})$

```
1: perm̄ = [(0, public, Freeable)]
2: return perm̄
```

is monotonically increasing after each time ϕ is called, and a *temp* counter that is monotonically decreasing after each time $\phi(temp)$ is called. The *temp* argument is optional, and it signifies when the *temp* counter is to be used – that is, only during the allocation of temporary variables used within the Private If Else rules. We separate these elements into their own partition of memory in order to easily show correctness of the memory with regards to Vanilla C- it is possible to provide a more robust mapping scheme between locations in Vanilla C and locations in SMC^2, but this extension provides unnecessary complexity for our proofs.

Algorithm 39 (τ) returns the expected bit-length for the given type. This is implementation and machine-specific, so we leave the lengths to be determined by the implementor.

Algorithm 40 (Label) returns the privacy label of the given expression. We set the default case as returning private, and list out cases that would be public.

Algorithm 38 $l_{next} \leftarrow \phi(\{temp\})$

```
1: next = l_default
2: if temp then
3:     next = global_location_temp_counter − −
4: else
5:     next = global_location_counter ++
6: end if
7: return l_next
```

Algorithm 39 $n \leftarrow \tau(ty)$

Algorithm 40 $a \leftarrow \text{Label}(e, \gamma)$

```
if (e = x(ē)) ∧ ((l, t̄y → ty) = γ(x)) then
    if ((ty = a bty*) ∨ (ty = a const bty*) ∨ (ty = a bty)) then
        return a
    end if
else if ((e = uop var) ∨ (e = var)) then
    if (var = x) ∧ ((l, ty) = γ(x)) then
        if ((ty = public bty*) ∨ (ty = public bty)) then
            return public
        end if
    else if (var = x[e1]) ∧ ((l, ty) = γ(x)) then
        if (ty = public const bty*) then
            return Label(e1, γ)
        end if
    end if
else if (e = e1 bop e2) then
    if (Label(e1, γ) = public) ∧ (Label(e2, γ) = public) then
        return public
    end if
else if (e = (e1)) then
    return Label(e1, γ)
else if (e = (ty) e1) then
    if ((ty = public bty) ∨ (ty = public bty*)) ∧ (Label(e1, γ) = public) then
        return public
    end if
else if ((e = n) ∨ (e = prim) ∨ (e = NULL) ∨ (e = char) ∨ (e = skip)) then
    return public
else if (e = v̄) then
    return Label(v[0], γ)
end if
return private
```

Next, we present the algorithms types used for encoding and decoding bytes in memory in our semantics. It is important to note that we leave the specifics of encoding to bytes and decoding from bytes up to the implementation, as this low-level function may vary based on the system and underlying architecture.

Algorithm 41 $\omega \leftarrow \text{EncodeVal}(ty, v)$

Algorithm 42 $v \leftarrow \text{DecodeVal}(ty, \omega)$

Algorithm 41, EncodeVal, takes as input a type and a value. It encodes the given value of the given type as bytes of data, and returns those bytes.

Algorithm 42, DecodeVal, takes as input a type and bytes of data. It interprets the given bytes of data as a value of the given type, and returns that value.

Algorithm 43 $\omega \leftarrow \mathrm{EncodePtr}(ty, [\alpha, \bar{l}, \bar{j}, i])$

Algorithm 44 $[\alpha, \bar{l}, \bar{j}, i] \leftarrow \mathrm{DecodePtr}(ty, \alpha, \omega)$

Algorithm 43 ($\mathrm{EncodePtr}$) takes a pointer data structure and encodes it into byte data. It takes a pointer type, number, and byte data as input. It then encodes the pointer data structure containing the number α indicating the number of locations, a list of α locations $\bar{l}$, a list of α tags, and a number indicating the level of indirection of the pointer into byte data. This byte data is then returned.

Algorithm 44 ($\mathrm{DecodePtr}$) does the opposite of $\mathrm{EncodePtr}$, taking byte data and retrieving the pointer data structure from it. It takes a pointer type, number, and byte data as input. It then interprets the given set of bytes as a pointer data structure containing the number α indicating the number of locations, a list of α locations $\bar{l}$, a list of α tags, and a number indicating the level of indirection of the pointer. This pointer data structure is then returned.

Algorithm 45 $\omega \leftarrow \mathrm{EncodeFun}(s, n, \overline{p})$

Algorithm 46 $(s, n, \overline{p}) \leftarrow \mathrm{DecodeFun}(\omega)$

Algorithm 45 ($\mathrm{EncodeFun}$) takes the function data and encodes it into its byte representation. It takes as input a statement (body of the function), the tag for whether it contains public side effects, and the function's parameter list. $\mathrm{EncodeFun}$ then encodes this information into byte data and returns the byte data.

Algorithm 46 ($\mathrm{DecodeFun}$) takes the byte representation of a function and decodes it into the function's information: the statement (body of the function), the tag for whether it contains public side effects, and the parameter list. It takes as input the byte data and then returns the function's information.

Next, we present the algorithms used to update memory within the semantics. The following algorithms are for regular (int or float) values, array values, and pointer values, respectively, when updating these values in memory.

Algorithm 47 $\sigma_2 \leftarrow \mathrm{UpdateVal}(\sigma,\ l,\ v,\ a\ bty)$

1: $\omega_2 = \mathrm{EncodeVal}(a\ bty,\ v)$
2: $\sigma_1[l \rightarrow (\omega_1,\ ty,\ n,\ \mathrm{PermL}(\mathrm{Freeable}, ty, a, n))] = \sigma$
3: $\sigma_2 = \sigma_1[l \rightarrow (\omega_2,\ ty,\ n,\ \mathrm{PermL}(\mathrm{Freeable}, ty, a, n))]$
4: **return** σ_2

Algorithm 47 ($\mathrm{UpdateVal}$) is used to update regular (int or float) values in memory. It takes as input memory σ, the memory block identifier of the location we will be updating, the value to store into memory, and the type to store it as. $\mathrm{UpdateVal}$ first encodes the value as the specified type, then removes the original

mapping from memory and inserts the new mapping with the updated byte data. It then returns the updated memory.

Algorithm 48 $(\sigma_2,\ j) \leftarrow \mathrm{UpdatePtr}(\sigma,\ (l, \mu),\ [\alpha,\ \bar{l},\ \bar{j},\ i],\ a\ bty*)$

```
1: j = 0
2: σ1[l → (ω1, ty1, α1, PermL(Freeable, ty, a1, α1))] = σ
3: if (μ = 0) ∧ (a bty* = ty1) then
4:     ω = EncodePtr(a bty*, [α, l̄, j̄, i])
5:     σ2 = σ1[l → (ω, ty1, 1, PermL(Freeable, ty1, a, 1))]
6:     j = 1
7: else
8:     σ2 = SetBytes((l, μ), a bty*, [α, l̄, j̄, i], σ)
9: end if
10: return (σ2, j)
```

Algorithm 48 ($\mathrm{UpdatePtr}$) is used to update the pointer data structure for a pointer. It takes as input memory σ, the location (memory block identifier and offset) we will be updating, the value to store into memory, and the type to store the value as. First, we extract the given memory block identifier's mapping in memory. If the given offset is 0 and the given pointer type matches the type in that mapping, we encode the pointer data structure into its byte representation and add a new mapping to memory with the new byte data, and set the tag to 1, indicating that we performed a *well-aligned* update to memory. Otherwise, we call $\mathrm{SetBytes}$ to perform the update to memory at this location, as it is not *well-aligned*. Finally, it returns the updated memory.

Algorithm 49 ($\mathrm{UpdatePrivPtr}$) is used during private pointer dereference writes at a level of indirection greater than 1, as shown in Figure 3.23. It facilitates the proper handling of private pointer data, particularly when there are multiple locations. First, we check if the default location is in the location list. If it is, then one of the possible locations for the pointer is an uninitialized location, and this would cause a segmentation fault at runtime. We return tag -1 if this is the case, and the program will get stuck and be unable to evaluate further. When there is a single location, it calls Algorithm 48 ($\mathrm{UpdatePtr}$) for that location to update the location stored there, and returns the updated memory and alignment tag. When there are multiple locations, it iterates through all the locations and updates each location. If the location is aligned, it will combine the two location lists and privately update the tag list using Algorithm 78, then update the location with the new location and tag lists. If the location is not aligned, then the alignment tag is set to 0 and the bytes in that location are overwritten with the new location list that we are storing in memory. If the update is aligned at all locations, then the tag will be returned as 1; otherwise, 0.

Algorithm 50 ($\mathrm{UpdatePriv}$) is used during private pointer dereference writes at the first level of indirec-

Algorithm 49 $(\sigma_f,\ j) \leftarrow \text{UpdatePrivPtr}(\sigma,\ [\alpha,\ \bar{l},\ \bar{j},\ i],\ [\alpha_e, \bar{l}_e, \bar{j}_e, i-1],\ \text{private } bty*)$

```
1: j = 1
2: if (l_default, 0) ∈ l̄ then
3:     ERROR
4:     j = −1
5: else if α = 1 then
6:     [(l, μ)] = l̄
7:     (σ_f, j) ← UpdatePtr(σ, (l, μ), [α_e, l̄_e, j̄_e, i − 1], private bty*)
8: else
9:     for all (l_m, μ_m) ∈ l̄ do
10:        σ_f[l_m → (ω_m, ty_m, n_m, PermL(Freeable, ty_m, a_m, n_m))] = σ
11:        if (μ_m = 0) ∧ (ty_m = private bty*) then
12:            [α_m, l̄_m, j̄_m, i − 1] = DecodePtr(ty, 1, ω_m)
13:            [α'_m, l̄'_m, j̄'_m] = CondAssign([α_e, l̄_e, j̄_e], [α_m, l̄_m, j̄_m], j_m)
14:            ω = EncodePtr(a bty*, [α'_m, l̄'_m, j̄'_m, i − 1])
15:            σ_f = σ_1[l_m → (ω, ty_m, α, PermL(Freeable, ty_m, a_m, α))]
16:            σ = σ_f
17:        else
18:            σ_f = SetBytes((l_m, μ_m), a bty*, [1, [(l_new, μ_new)], [1], i], σ)
19:            j = 0
20:        end if
21:    end for
22: end if
23: return (σ_f, j)
```

tion, as shown in Figure 3.22. It facilitates the proper handling of private pointer data, particularly when there are multiple locations. First, we check if the default location is in the location list. If it is, then one of the possible locations for the pointer is an uninitialized location, and this would cause a segmentation fault at runtime. We return tag -1 if this is the case, and the program will get stuck and be unable to evaluate further. When there is a single location, it calls Algorithm 51 (UpdateOffset) for that location to update the value stored there and returns the updated memory and alignment tag. When there are multiple locations, it iterates through all the locations and updates each at location. If the location is aligned, it will privately update the value based on the tag for the location, and then update the location with this value. If the location is not aligned, then the alignment tag is set to 0 and the bytes in that location are overwritten with the new value that we are storing in memory. If the update is aligned at all locations, then the tag will be returned as 1; otherwise, 0.

Algorithm 51 (UpdateOffset) is designed to update a value at an offset within a memory block, and is used by semantic rules Pointer Dereference Write Value and Pointer Dereference Write Value (Not Aligned). First, we check that we are not trying to update the default location, which is not valid (i.e., this would cause a segmentation fault). If we are, we return with the tag -1, which will not allow the statement that triggered this to resolve any further. We extract the memory block we are looking going to be updating. Next, we

Algorithm 50 $(\sigma_1,\ j) \leftarrow \text{UpdatePriv}(\sigma,\ \alpha,\ \bar{l},\ \bar{j},\ \text{private } bty,\ v)$

```
1: j = 1
2: if (l_default, 0) ∈ l̄ then
3:     ERROR
4:     return (σ, −1)
5: else if α = 1 then
6:     [(l_1, μ_1)] = l̄
7:     (σ_1, j) = UpdateOffset(σ, (l_1, μ_1), v, private bty)
8: else
9:     for all (l_m, μ_m) ∈ l̄ do
10:        (ω_m, ty_m, n, PermL(Freeable, ty_m, a_m, n)) = σ(l_m)
11:        if (μ_m = 0) ∧ (ty_m = private bty) ∧ (n = 1) then
12:            v_m = DecodeVal(private bty, ω_m)
13:            v'_m = (j̄[m] ∧ v) ∨ (¬ j̄[m] ∧ v_m)
14:            σ_1 = UpdateVal(σ, l_m, v'_m, private bty)
15:            σ = σ_1
16:        else if (μ % τ(private bty) = 0) ∧ (μ / τ(private bty) < n) ∧ (ty_m = private bty) then
17:            [v_0, ..., v_{n−1}] = DecodeVal(private bty, n, ω_m)
18:            v'_{μ_m} = (j̄[m] ∧ v) ∨ (¬ j̄[m] ∧ v_{μ_m})
19:            [v'_0, ..., v'_{n−1}] = [v_0, ..., v_{n−1}](v'_{μ_m} / v_{μ_m})
20:            σ_1 = UpdateVal(σ, l_m, [v'_0, ..., v'_{n−1}], private bty)
21:            σ = σ_1
22:        else
23:            σ_1 = SetBytes((l_m, μ_m), private bty, v, σ)
24:            j = 0
25:            σ = σ_1
26:        end if
27:    end for
28: end if
29: return (σ_1, j)
```

check if the memory block is of the expected type, the offset is 0, and the number of locations is 1 to see if this is a simple update. In this case, we encode the value into its byte representation and add the updated mapping into the final memory and set the tag to be 1, as we have made a *well-aligned* update to memory. If not, and the memory block happens to be a block of array data, we can check if our update will be aligned by checking if the type is the same, then if the byte-offset of the pointer aligns with a value of the given type within the block by using the modulo operation, and also that it is within the range of the current block based on the given type and the number of locations in the block. If all of these elements are true, we will have a *well-aligned* update to memory. We use Algorithm 65 (SetBytes) to perform the update here, as that algorithm facilitates proper insertion of the byte representation for a value into a larger block or across blocks. Finally, we return the updated memory and tag.

Algorithm 52 (GetFunTypeList) is designed to obtain the function input type from its parameter list, taking as input a parameter list and returning the corresponding type list.

Algorithm 53 (GetFunParamAssign) is designed to create assignment statements for the parameters of

Algorithm 51 $(\sigma_f,\ j) \leftarrow \text{UpdateOffset}(\sigma,\ (l, \mu),\ v,\ a\ bty)$

1: **if** $l_{default} = l$ **then**
2: ERROR
3: **return** $(\sigma, -1)$
4: **end if**
5: $j = 0$
6: $\sigma_f[l \rightarrow (\omega_1,\ ty_1,\ \alpha,\ \text{PermL}(\text{Freeable}, ty_1, a_1, \alpha))] = \sigma$
7: **if** $(a\ bty = ty_1) \wedge (\mu = 0) \wedge (\alpha = 1)$ **then**
8: $\omega_2 = \text{EncodeVal}(a\ bty,\ v)$
9: $\sigma_f = \sigma_f[l \rightarrow (\omega_2,\ ty_1,\ 1,\ \text{PermL}(\text{Freeable}, \widehat{ty}_1, a, 1))]$
10: $j = 1$
11: **else**
12: **if** $(a\ bty = ty_1) \wedge (\mu\ \%\ \tau(ty_1) = 0) \wedge (\frac{\mu}{\tau(ty_1)} < \alpha)$ **then**
13: $j = 1$
14: **end if**
15: $\sigma_f = \text{SetBytes}((l, \mu), a\ bty, v, \sigma)$
16: **end if**
17: **return** (σ_f, j)

Algorithm 52 $\overline{ty} \leftarrow \text{GetFunTypeList}(\overline{p})$

1: $\overline{ty} = [\,]$
2: **while** $(\overline{p} \neq \text{void})$ **do**
3: **if** $(\overline{p} = ty\ x)$ **then**
4: $\overline{ty} = ty :: \overline{ty}$
5: $\overline{p} = \text{void}$
6: **else if** $(\overline{p} = \overline{p}', ty\ x)$ **then**
7: $\overline{ty} = ty :: \overline{ty}$
8: $\overline{p} = \overline{p}'$
9: **end if**
10: **end while**
11: **return** $\overline{ty}$

a function, taking as input a parameter list and expression list and returning a set of assignment statements to assign the expressions to the corresponding parameters.

Algorithm 54 (CheckPublicEffects) is designed to decide whether a function has public side effects or not. It takes as input the function body s, the function name x, the environment γ, and the memory σ, and returns a tag j indicating whether there are public side effects found (1) or not (0). It is worth noting that we pass in the name of the function that we are evaluating in order to account for recursive functions, as the name of the function we are evaluating will not yet have an annotation in memory for whether it has public effects or not; we make the explicit annotation within the check for function calls that the function . This algorithm short-circuits on finding the first public side effect.

Algorithm 53 $s \leftarrow \text{GetFunParamAssign}(\overline{p}, \overline{e})$

Require: $\text{length}(\overline{p}) = \text{length}(\overline{e})$
$s = \text{skip}$
while $(\overline{p} \neq \text{void})$ **do**
  if $(\overline{p} = ty\ var) \wedge (\overline{e} = e)$ **then**
    $s = ty\ var = e; s$
    $\overline{p} = \text{void}$
    $\overline{e} = \text{void}$
  else if $(\overline{p} = \overline{p}', ty) \wedge (\overline{e} = \overline{e}', e)$ **then**
    $s = ty\ var = e; s$
    $\overline{p} = \overline{p}'$
    $\overline{e} = \overline{e}'$
  end if
end while
return s

Algorithm 54 $j \leftarrow \text{CheckPublicEffects}(s,\ x,\ \gamma, \sigma)$

$j = 0$
for all $s_1 \in s$ **do**
  if $(s_1 = x_1\ ++)$ **then**
    $(l, ty) = \gamma(x_1)$
    if $(ty = \text{public}\ bty) \vee (ty = \text{public}\ bty*)$ **then**
      return 1
    end if
  else if $(s_1 = x_1 = e)$ **then**
    $(l, ty) = \gamma(x_1)$
    if $(ty = \text{public}\ bty) \vee (ty = \text{public}\ bty*) \vee (\text{CheckPublicEffects}(e,\ x,\ \gamma, \sigma) = 1)$ **then**
      return 1
    end if
  else if $(s_1 = ty\ var)$ **then**
    if $(ty = \text{public}\ bty) \vee (ty = \text{public}\ bty*)$ **then**
      return 1
    end if
  else if $(s_1 = ty\ var = e)$ **then**
    if $(ty = \text{public}\ bty) \vee (ty = \text{public}\ bty*) \vee (\text{CheckPublicEffects}(e,\ x,\ \gamma, \sigma) = 1)$ **then**
      return 1
    end if
  else if $(\text{pmalloc}(e, ty)) \vee (\text{malloc}(e)) \vee (\text{pfree}(x)) \vee (\text{free}(x)) \vee (\text{smcinput}(var, e)) \vee (\text{smcoutput}(var, e))$ **then**
    return 1
  else if $s_1 = ty\ x_1(\overline{p})\{s_2\}$ **then**
    if $\text{CheckPublicEffects}(s_2,\ x_1,\ \gamma, \sigma) = 1$ **then**
      return 1
    end if
  else if $(s_1 = x_1(\overline{e})) \wedge (x \neq x_1)$ **then**
    $(l, ty) = \gamma(x_1)$
    $(\omega, ty, 1, \text{PermL_Fun(public)}) = \sigma(l)$
    if $((s_2, 1, \overline{p}) = \text{DecodeFun}(\omega)) \vee ((\text{NULL}, -1, \overline{p}) = \text{DecodeFun}(\omega))$ **then**
      return 1
    end if
  end if
end for
return j

Algorithm 55 $(n_1) \leftarrow \text{Cast}(a,\ ty,\ n)$

```
1: ω = EncodeVal(ty, n)
2: n_1 = DecodeVal(ty, 1, ω)
3: return n_1
```

Algorithm 55 (Cast) is designed to take a privacy label, type, and value, and cast the given value as the into the appropriate size for the new type. Here, we model this as simply encoding the given value into it's byte representation for the new type, then decoding it back into a value of the new type. This assumes that there is a standard handling of casting a value of one type to another type defined in the system or implementation itself, but leaves it to the implementation to define the precise behavior, which can vary.

Algorithm 56 $(n_1) \leftarrow \text{encrypt}(n)$

Algorithm 56 (encrypt) is designed to take a number and return its encrypted representation. We do not define the specifics of encryption in our model, as it is dependent on the implementation. However, we do assert that the encryption algorithm must maintain the correctness and security properties we define.

Algorithm 57 $(v,\ j) \leftarrow \text{ReadOOB}(i,\ n,\ l,\ a\ bty,\ \sigma)$

```
1: n_v = τ(a bty)
2: n_b = (i − n) · n_v
3: j = 1
4: ω_v = [ ]
5: while (n_b > 0) ∨ (n_v > 0) do
6:     l = GetBlock(l)
7:     (ω, ty_1, α, PermL(Freeable, ty_1, a_1, α)) = σ(l)
8:     if (ty_1 ≠ a bty) then
9:         j = 0
10:    end if
11:    if (n_b < τ(ty_1) · α) then
12:        ω_v = ω_v :: ω[n_b : min(n_b + n_v, τ(ty_1) · α − 1)]
13:        n_v = n_v − τ(ty_1) · α + n_b
14:    end if
15:    n_b = max(0, n_b − τ(ty_1) · α)
16: end while
17: v = DecodeVal(a bty, 1, ω_v)
18: return (v, j)
```

Algorithm 57 (ReadOOB) is designed to read a value of the given type from memory as though it was at index i of the array in memory block l. It takes as input the out of bounds index i, the number of values in the array n, the memory block of the array data l, the type of elements in the array $a\ bty$, and memory σ. It then iterates through memory until it finds the bytes that would be at index i and decodes them as the expected type bty to obtain value v. It is important to note here that index i will be public, as we do not overshoot the

bounds of an array when we have a private index. As the algorithm iterates through memory, if all locations we iterate over are of the same type as the expected type, and the location we are reading the value from is also the expected type, then it will return tag 1, indicating that our read was *well-aligned*. Otherwise, tag 0 will be returned. We currently only show the algorithm handling overshooting in the positive direction, however, it can trivially extended to grab the previous blocks and iterate backwards through memory to handle a negative index as well.

Algorithm 58 $(\sigma,\ j) \leftarrow \mathrm{WriteOOB}(v,\ i,\ n,\ l,\ a\ bty,\ \sigma)$

```
1: ω_v = EncodeVal(a bty, v)
2: n_b = (i − n) · τ(a bty)
3: j = 1
4: while (n_b > 0) ∨ (|ω_v| > 0) do
5:     l = GetBlock(l)
6:     σ_1[l → (ω, ty_1, α, PermL(Freeable, ty_1, a_1, α))] = σ
7:     if (ty_1 ≠ a bty) then
8:         j = 0
9:     end if
10:    if (n_b < τ(ty_1) · α) then
11:        if (|ω_v| > τ(ty_1) · α − n_b) then
12:            ω_1 = ω[0 : n_b] + ω_v + ω[|ω_v| + n_b :]
13:            ω_v = [ ]
14:        else if (|ω_v| = τ(ty_1) · α − n_b) then
15:            ω_1 = ω[0 : n_b] + ω_v
16:            ω_v = [ ]
17:        else
18:            ω_1 = ω[0 : n_b] + ω_v[0 : τ(ty_1) · α − n_b − 1]
19:            ω_v = ω_v[τ(ty_1) · α − n_b :]
20:        end if
21:        σ = σ_1[l → (ω_1, ty_1, α, PermL(Freeable, ty_1, a_1, α))]
22:    end if
23:    n_b = max(0, n_b − τ(ty_1) · α)
24: end while
25: return (σ, j)
```

Algorithm 58 (WriteOOB) is designed to store a value of the given type from memory as though it was at index i of the array in memory block l. It takes as input the value to write in memory v, the out of bounds index i, the number of values in the array n, the memory block of the array data l, the type of elements in the array $a\ bty$, and memory σ. It then iterates through memory until it finds the position that would be for index i, encodes value v as the expected type, and places its byte representation into memory starting at that position. It is important to note here that index i will be public, as we do not overshoot the bounds of an array when we have a private index. As the algorithm iterates through memory, if all locations we iterate over are of the same type as the expected type, and the location we are writing the value to is also the expected type, then it will return tag 1, indicating that our read was *well-aligned*. Otherwise, tag 0 will be returned. We currently

only show the algorithm handling overshooting in the positive direction, however, it can trivially extended to grab the previous blocks and iterate backwards through memory to handle a negative index as well.

Algorithm 59 $(n) \leftarrow \text{InputValue}(x,\ n_p)$

```
n = NULL
file = open(inputFiles[n_p − 1], r)
for all line ∈ file do
    (x_f, v) = line.split(=)
    if x_f = x then
        if v = n̄ then
            n = n̄[0]
        else
            n = v
        end if
        break
    end if
end for
file.close()
return n
```

Algorithm 59 (InputValue) is designed to obtain a single value from a specific input party's input file. We first open the input file for that party in read mode, then iterate through the file to find the desired variable. We then check if there is more than one value - if there is, we take only the first value; otherwise, we return the value as we read it in.

Algorithm 60 $(\overline{n}) \leftarrow \text{InputArray}(x, n_p, n)$

```
n̄ = [ ]
file = open(inputFiles[n_p − 1], r)
for all line ∈ file do
    (x_f, n̄_f) = line.split(=)
    if x_f = x then
        if n̄_f.length() > n_p then
            n̄ = n̄_f[0 : n_p − 1]
        else
            n̄ = n̄_f
        end if
        break
    end if
end for
file.close()
return n̄
```

Algorithm 60 (InputArray) is designed to obtain a list of values from a specific input party's input file. We first open the input file for that party in read mode, then iterate through the file to find the desired variable. We then check if the length of the array given is longer than the desired length, and only take the desired length if it is. Otherwise, we return the array as we read it in.

Algorithm 61 (OutputValue) is designed to write a value out to an output file for the output party n_p.

Algorithm 61 NULL $\leftarrow$ OutputValue$(x,\ n_p,\ n)$

```
1: file = open(outputFiles[n_p − 1], a)
2: file.write(x = n)
3: file.close()
4: return NULL
```

We first open the output file for that party in append mode, as we want to allow multiple uses of the output function throughout the program. Next, we write the variable name and the value out to the file, then close the file and return from this algorithm.

Algorithm 62 NULL $\leftarrow$ OutputArray$(x,\ n_p,\ \overline{n})$

```
1: file = open(outputFiles[n_p − 1], a)
2: file.write(x = n̄)
3: file.close()
4: return NULL
```

Algorithm 62 (OutputArray) is designed to write the given array to an output file for the output party n_p. We first open the output file for that party in append mode, as we want to allow multiple uses of the output function throughout the program. Next, we write the variable name and its list of values out to the file, then close the file and return from this algorithm.

Algorithm 63 $(\bar{l}_2,\ j_f) \leftarrow$ IncrementList$(\bar{l}_1,\ n,\ \sigma)$

```
1: if (l_default, 0) ∈ l̄_1 then
2:     ERROR
3:     return (l̄_1, −1)
4: end if
5: l̄_2 = [ ]
6: j_final = 1
7: for all (l, μ) ∈ l̄_1 do
8:     if l = l_default then
9:         l̄_2.append((l_default, 0))
10:    else
11:        ((l_1, μ_1), j) = GetLocation((l, μ), n, σ)
12:        j_f = j ∧ j_f
13:        l̄_2.append((l_1, μ_1))
14:    end if
15: end for
16: return (l̄_2, j_f)
```

Algorithm 63 (IncrementList) is used to increment every location in the location list of a private pointer by the given size n. It takes a location list $\bar{l}_1$, size n, and memory σ as input, and returns the incremented location list $\bar{l}_2$ and alignment indicator j_{final}. The alignment indicator is used to show whether or not the increment operation over each location is well-aligned or not (i.e., every location was incremented over an element of the expected size and type). This is used when proving correctness with respect to Vanilla C, as

we only consider operations that are *well-aligned*. In this algorithm, we check if the default location is in the location list first. If it is, then one of the possible locations for the pointer is an uninitialized location, and this would cause a segmentation fault at runtime. We return tag -1 if this is the case, and the program will get stuck and be unable to evaluate further.

Algorithm 64 $((l_f, \mu_f), j) \leftarrow \mathrm{GetLocation}((l, \mu), n, \sigma)$

```
 1: (ω1, ty1, α1, PermL(perm, ty1, a1, α1)) = σ(l)
 2: j = 1
 3: if (τ(ty1) ≠ n) ∨ (μ % τ(ty1) ≠ 0) then
 4:     j = 0
 5: end if
 6: if (n < τ(ty1) · α1 − μ) then
 7:     (lf, μf) = (l, μ + n)
 8: else
 9:     n2 = n − τ(ty1) · α1 − μ
10:     l1 = l
11:     while n2 ≥ 0 do
12:         (l2) = GetBlock(l1)
13:         (ω2, ty2, α2, PermL(perm, ty2, a2, α2)) = σ(l2)
14:         if (n2 = 0) then
15:             (lf, μf) = (l2, 0)
16:             n2 = −1
17:             if (j = 1) ∧ (τ(ty2) ≠ n) then
18:                 j = 0
19:             end if
20:         else
21:             (j, l1) = (0, l2)
22:             if (n2 < τ(ty2) · α2) then
23:                 (lf, μf) = (l2, n2)
24:                 n2 = −1
25:             else
26:                 n2 = n2 − τ(ty2) · α2
27:             end if
28:         end if
29:     end while
30: end if
31: return ((lf, μf), j)
```

Algorithm 64 ($\mathrm{GetLocation}$) is designed to obtain the next location that is n bytes ahead of the current location (l, μ). It takes the current location, the number of bytes to get to the next location, and memory as input and return the next location and a tag indicating whether the next location is *well-aligned* or not. We first look up the current memory block and initialize the tag as 1. Then, we check whether the next location is not *well-aligned* with the current block. If it is not, then it cannot be *well-aligned* with the location we return, either. Next, we check whether the next location is still within this block. If it is, we update the offset and return. If it is not, we must obtain the next memory block and keep checking. To do this, we first calculate the number of bytes that we need to move forward beyond the current block, and we initialize the memory block

identifier we are looking beyond as the original memory block identifier. Then we loop until we have finally found the next location to return. In each loop iteration, we grab the next block using Algorithm 68, and look up the location in memory. If the number of bytes we have left is 0, then our next location is at the start of the block. We perform one last check to see if the current location is *well-aligned* with the new location (which is possible if this is the start of the next block immediately after the one we were in). Otherwise, we set the tag as 0 and reset the block we are looking beyond to be the new block. Then, we check whether our new location is within this block. If it is not, we decrease the number of bytes we need to move forward by the number of bytes within this block, and repeat the loop.

Algorithm 65 $(\sigma_f) \leftarrow \mathrm{SetBytes}((l, \mu), ty, v, \sigma)$

1: $\sigma_f[l \rightarrow (\omega,\ ty_1,\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable}, ty_1, a, \alpha))] = \sigma$
2: $\omega_v = \mathrm{NULL}$
3: **if** $ty = a\ bty$ **then**
4: $\quad \omega_v = \mathrm{EncodeVal}(ty, v)$
5: **else**
6: $\quad \omega_v = \mathrm{EncodePtr}(ty, v)$
7: **end if**
8: $n_l = \tau(ty_1) \cdot \alpha - \mu$
9: **if** $(\tau(ty) < n_l - 1)$ **then**
10: $\quad \omega_f = \omega[0 : \mu - 1] + \omega_v + \omega[\mu + \tau(ty) :]$
11: $\quad \sigma_f = \sigma_f[l \rightarrow (\omega_f,\ ty_1,\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable}, ty_1, a, \alpha))]$
12: **else if** $(\tau(ty) = n_l - 1)$ **then**
13: $\quad \omega_f = \omega[0 : \mu - 1] + \omega_v$
14: $\quad \sigma_f = \sigma_f[l \rightarrow (\omega_f,\ ty_1,\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable}, ty_1, a, \alpha))]$
15: **else**
16: $\quad \omega_f = \omega[0 : \mu - 1] + \omega_v[0 : n_l - 1]$
17: $\quad \sigma_f = \sigma_f[l \rightarrow (\omega_f,\ ty_1,\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable}, ty_1, a, \alpha))]$
18: $\quad \omega_v = \omega_v[n_l :]$
19: $\quad n_v = \tau(ty) - n_l$
20: $\quad$ **while** $(n_v > 0)$ **do**
21: $\quad\quad l = \mathrm{GetBlock}(l)$
22: $\quad\quad \sigma_f[l \rightarrow (\omega_c,\ ty_c,\ \alpha_c,\ \mathrm{PermL}(\mathrm{Freeable}, ty_c, a, \alpha_c))] = \sigma_f$
23: $\quad\quad n_c = \tau(ty_c) \cdot \alpha_c$
24: $\quad\quad$ **if** $(n_v < n_c)$ **then**
25: $\quad\quad\quad \omega_f = \omega_v + \omega_c[0 : n_v]$
26: $\quad\quad$ **else if** $(n_v = n_c)$ **then**
27: $\quad\quad\quad \omega_f = \omega_v$
28: $\quad\quad$ **else**
29: $\quad\quad\quad \omega_f = \omega_v[0 : n_c - 1]$
30: $\quad\quad\quad \omega_v = \omega_v[n_c :]$
31: $\quad\quad$ **end if**
32: $\quad\quad n_v = n_v - n_c$
33: $\quad\quad \sigma_f = \sigma_f[l \rightarrow (\omega_f,\ ty_c,\ \alpha_c,\ \mathrm{PermL}(\mathrm{Freeable}, ty_c, a, \alpha_c))]$
34: $\quad$ **end while**
35: **end if**
36: **return** σ_f

Algorithm 65 (SetBytes) is designed to store a value into a location in memory that may not be *well-aligned*. It takes as input the location, the type to encode the byte representation of the value as, the value, and

the memory, and returns the updated memory. It is worthwhile to note here that we never change the privacy label or type for the location, we simply encode the value into a byte representation based on it's expected type (not that of the location). This prevents any unintentional encryption of public values or decryption of private values. When we later read from this location, we will again read from it as the type we are expecting to be there rather than the type that is there - this may result in garbage values being used in a program that was not ensured to be correct by the programmer, but it prevents any information leakage about private data when private data is stored in an incorrect position.

In this algorithm, we first remove the current block referred to by the location from memory, then obtain the byte representation of the value based on the type of the value - line 4 handling non-pointer values, and line 6 handling pointer values (i.e., pointer data structures). In line 8, we are finding the size of the location (in bytes) that is left after the offset given by the location we received as input. We then check this size against the size of the value we are trying to store into memory. In line 9, we are checking if the value's byte representation will fit somewhere in the middle of the block, with bytes at the beginning and at the end. We then take the first portion of what was in the block up until the offset and add the byte representation we are storing to it, and at the end add on whatever bytes were left beyond the chunk that we replaced. We then store this final byte representation back into this location.

In line 12, we have the the byte representation of the value we obtained should fit within this memory block, and will take up the space from the offset until the end of the block. Thus, we take the first part of the bytes currently stored in the block up until the offset, and replace the rest with our byte representation of the given value, storing it into memory. In line 15, we are entering the portion where we are overflowing from this block into block(s) that come after it. We first store here whatever portion of our byte representation of the value that can be stored within this block, then obtain the remaining portion and length that we still need to store. In line 20, we enter a loop that will iteratively store the rest into memory. It first obtains the next sequential block, then stores whatever it can into that block, appropriately keeping any bytes of data that were there if what we are storing will not take up the entire block. It continues this process until the entire byte representation for the value has been stored into memory.

Algorithm 66 ($\mathrm{GetBytes}$) is designed to take the byte representation from a specific location and interpret it as the given type. It takes as input the location, the expected type, and the memory, and returns a value of the expected type. It is important to note here that when we are trying to read a pointer data structure for a public pointer from a location that is not aligned, this function will automatically ignore the bytes for the

Algorithm 66 $(v_f) \leftarrow \text{GetBytes}((l, \mu), ty, \sigma)$

```
1: (ω, ty_1, α, PermL(Freeable, ty_1, a, α)) = σ(l)
2: v_f = NULL
3: n_c = τ(ty_1) · α − μ
4: if ty = a bty then
5:     if (τ(ty) ≤ n_c − 1) then
6:         ω_v = ω[μ : μ + τ(ty)]
7:     else
8:         ω_v = ω[μ :]
9:         n_v = τ(ty) − n_c
10:        while (n_v > 0) do
11:            l = GetBlock(l)
12:            (ω_c, ty_c, α_c, PermL(Freeable, ty_c, a, α_c)) = σ(l)
13:            n_c = τ(ty_c) · α_c
14:            ω_v = ω_v + ω_c[0 : min(n_v, n_c) − 1]
15:            n_v = n_v − n_c
16:        end while
17:    end if
18:    v_f = DecodeVal(ty, 1, ω_v)
19: else if (ty = public bty∗) then
20:    if (τ(public int) · 5 ≤ n_c − 1) then
21:        ω_v = ω[μ : μ + n_v]
22:    else
23:        ω_v = ω[μ :]
24:        n_v = τ(public int) · 5 − n_c
25:        while (n_v > 0) do
26:            l = GetBlock(l)
27:            (ω_c, ty_c, α_c, PermL(Freeable, ty_c, a, α_c)) = σ(l)
28:            n_c = τ(ty_c) · α_c
29:            ω_v = ω_v + ω_c[0 : min(n_v, n_c) − 1]
30:            n_v = n_v − n_c
31:        end while
32:    end if
33:    [α_1, [(l_1, μ_1)], [j_1], i_1] = DecodePtr(ty, 1, ω_v)
34:    v_f = [1, [(l_1, μ_1)], [1], i_1]
35: else if (ty = private bty∗) then
36:    (v_f) = GetBytesPrivPtr((l, μ), private bty∗, σ)
37: end if
38: return v_f
```

number of locations and the tag and assume they are both 1. If we are trying to read a pointer data structure for a private pointer, we will try to interpret the bytes as such and use the first public int-sized chunk to find out how many locations this pointer is intended to have. We split the behavior for obtaining the bytes for private pointers into its own algorithm (67) in order to reduce the length and complexity of this algorithm. As with any read that isn't necessarily *well-aligned*, this many introduce garbage into the evaluation due to errors in the original program, but will not leak anything about private data, as we will interpret the byte representations as the type we anticipate them to be, rather than the type of the memory block that the bytes are stored in.

Algorithm 67 (GetBytesPrivPtr) is an extension to Algorithm 66, handling the private pointer case

Algorithm 67 $(v_f) \leftarrow \text{GetBytesPrivPtr}((l, \mu), \text{private } bty*, \sigma)$

1: $(\omega,\ ty_1,\ \alpha,\ \text{PermL}(\text{Freeable}, ty_1, a, \alpha)) = \sigma(l)$
2: $v_f = \text{NULL}$
3: $n_c = \tau(ty_1) \cdot \alpha - \mu$
4: **if** $(\tau(\text{public int}) \leq n_c - 1)$ **then**
5: $\quad \omega_p = \omega[\mu : \mu + \tau(\text{public int})]$
6: **else**
7: $\quad \omega_p = \omega[\mu :]$
8: $\quad n_p = \tau(\text{public int}) - n_c$
9: $\quad l_1 = l$
10: $\quad$ **while** $(n_p > 0)$ **do**
11: $\quad\quad l_1 = \text{GetBlock}(l_1)$
12: $\quad\quad (\omega_c,\ ty_c,\ \alpha_c,\ \text{PermL}(\text{Freeable}, ty_c, a, \alpha_c)) = \sigma(l)$
13: $\quad\quad n_c = \tau(ty_c) \cdot \alpha_c$
14: $\quad\quad \omega_p = \omega_p + \omega_c[0 : min(n_p, n_c) - 1]$
15: $\quad\quad n_p = n_p - n_c$
16: $\quad$ **end while**
17: **end if**
18: $\alpha_p = \text{DecodeVal}(\text{public int}, 1, \omega_p)$
19: $n_v = \tau(\text{public int}) + \tau(\text{public int}) \cdot 2 \cdot \alpha_p + \tau(\text{private int}) \cdot 2 \cdot \alpha_p + \tau(\text{public int})$
20: **if** $(n_v \leq n_c - 1)$ **then**
21: $\quad \omega_v = \omega[\mu : \mu + n_v]$
22: **else**
23: $\quad \omega_v = \omega[\mu :]$
24: $\quad n_v = n_v - n_c$
25: $\quad$ **while** $(n_v > 0)$ **do**
26: $\quad\quad l = \text{GetBlock}(l)$
27: $\quad\quad (\omega_c,\ ty_c,\ \alpha_c,\ \text{PermL}(\text{Freeable}, ty_c, a, \alpha_c)) = \sigma(l)$
28: $\quad\quad n_c = \tau(ty_c) \cdot \alpha_c$
29: $\quad\quad \omega_v = \omega_v + \omega_c[0 : min(n_v, n_c) - 1]$
30: $\quad\quad n_v = n_v - n_c$
31: $\quad$ **end while**
32: **end if**
33: $v_f = \text{DecodePtr}(\text{private } bty*, \alpha_p, \omega_v)$
34: **return** v_f

where we may have multiple locations. In this case, we must obtain the first public integer-sized bytes and decode them to obtain how many locations should be stored for this private pointer (lines 4-18). We then determine the size of this private pointer data structure in bytes, and obtain that number of bytes from memory before returning the decoded pointer data structure as the final value.

Algorithm 68 $(l_{n+1}) \leftarrow \text{GetBlock}(l_n)$

1: **return** l_{n+1}

Algorithm 68 (GetBlock) is designed to select the identifier for the next block in memory after the current one. It takes a memory block identifier as input, and returns the next sequential memory block. We chose to formalize this as simply grabbing the next higher block, however, our formalization will work with any implementation of this (e.g., selecting a random block, looping through only allocated blocks, etc.).

Algorithm 69 $(n) \leftarrow \text{GetIndirection}(*)$

1: $n = |*|$
2: **return** n

Algorithm 69 (GetIndirection) takes the $*$ and returns the number of them. This is used in pointer declarations. We use the $|\ |$ syntax to indicate finding the number (or length) of the $*$.

Algorithm 70 $(v, j) \leftarrow \text{DerefPtr}(\sigma, \text{public } bty, (l, \mu))$

1: **if** $(l_{default} = l)$ **then**
2: ERROR
3: **return** $(\text{NULL}, -1)$
4: **end if**
5: $(v, j) = (\text{NULL}, 1)$
6: $(\omega,\ ty_1,\ \alpha,\ \text{PermL}(\text{Freeable}, ty_1, a, \alpha)) = \widehat{\sigma}(\widehat{l})$
7: **if** $(\text{public } bty = ty_1) \wedge (\mu = 0) \wedge (\alpha = 1)$ **then**
8: $v = \text{DecodeVal}(\text{public } bty,\ 1, \omega)$
9: **else if** $(\text{public } bty = ty_1) \wedge (\mu \,\%\, \tau(\text{public } bty) = 0) \wedge (\frac{\mu}{\tau(\text{public } bty)} < \alpha)$ **then**
10: $[v_0, ..., v_{\alpha-1}] = \text{DecodeVal}(\text{public } bty,\ \alpha, \omega)$
11: $n = \frac{\mu}{\tau(\text{public } bty)}$
12: $v = v_n$
13: **else**
14: $j = 0$
15: $v = \text{GetBytes}((l, \mu), \text{public } bty, \sigma)$
16: **end if**
17: **return** (v, j)

Algorithm 70 (DerefPtr) is designed to dereference a pointer at with a single level of indirection and a single location, obtaining the value stored at the location that the pointer refers to. It takes the memory, a type, and a location as input, and interprets the byte data at that location as a value of the given type. First, we check that we are not trying to update the default location, which is not valid (i.e., this would cause a segmentation fault). If we are, we return with the tag -1, which will not allow the statement that triggered this to resolve any further. Then, we look up the memory block associated with the location. In line 7, we check to see if this is a simple lookup by seeing if the location is of the correct type, the offset it 0, and there is only one element in the block. In line 9, we check to see if this is a simple lookup in an array block, where we have the correct type and the offset corresponds to an element of the array. For both of these cases, we can use Algorithm 42 to get the value(s) stored in the block and return the correct value. In line 14, we have a dereference that is not *well-aligned*, and so we set the tag to 0 and use Algorithm 66 to obtain a value of the anticipated type from the byte representation, although the value is likely garbage. Finally, we return the value and the tag.

Algorithm 71 (DerefPtrHLI) is designed to dereference a pointer of a level of indirection greater than 1

Algorithm 71 $([1, [(l_1, \mu_1)], [1], i], j) \leftarrow \text{DerefPtrHLI}(\sigma, a\ bty*, (l, \mu))$

```
1: if l_default = l then
2:     ERROR
3:     return ([0, [ ], [ ], 0], −1)
4: end if
5: (ω, ty_1, α, PermL(Freeable, ty_1, a_1, α)) = σ(l)
6: if (a bty* = ty_1) ∧ (μ = 0) ∧ (α̂ = 1) then
7:     [1, [(l_1, μ_1)], [1], i] = DecodePtr(a bty*, 1, ω)
8:     return ([1, [(l_1, μ_1)], [1], i], 1)
9: else
10:    [1, [(l_1, μ_1)], [1], i] = GetBytes((l, μ), a bty*, σ)
11:    return ([1, [(l_1, μ_1)], [1], i], 0)
12: end if
```

when the pointer only has a single location, obtaining the pointer data structure stored at that location. It takes the memory, a type, and a location as input, and interprets the byte representation from that location as a pointer data structure. For pointers, we currently assert that the location we are grabbing the pointer from must be at the beginning of the block, otherwise the pointer data is not aligned, as we do not currently support arrays of pointers. The behavior of this algorithm is similar to the previous algorithm, just obtaining a pointer data structure instead of a value. It returns the pointer data structure and a tag indicating whether the access was *well-aligned* or not. In this algorithm, we first check if the location we are trying to dereference is the default location. If it is, then this is an uninitialized location, and would cause a segmentation fault at runtime. We return tag -1 if this is the case, and the program will get stuck and be unable to evaluate further.

Algorithm 72 (DerefPrivPtr) is designed to to dereference a private pointer of a level of indirection greater than 1, obtaining the pointer data structure stored at that location. It takes the memory and the number of locations, location list, tag list, and type of pointer to be dereferenced as input. First, we check that we will not be trying to update the default location, which is not valid (i.e., this would cause a segmentation fault). If we are, we return with the tag -1, which will not allow the statement that triggered this to resolve any further. In line 5, we initialize the variables we are using to return values from this algorithm. Then, we check if there is only a single location, and if so, we can use Algorithm 71 (DerefPtrHLI) to obtain the final values. If there are multiple locations, we must iterate through all the locations and dereference each. For each location, we first look up the location in memory. We then check if the offset is 0 and the type is the expected private pointer type, in which case we have a *well-aligned* location and we can use Algorithm 44 to properly interpret the pointer data structure stored in that location. If not, we set the alignment tag to be 0, and use Algorithm 66 to obtain a pointer data structure from memory at the given location and offset. We then iterate through all the locations in the lower level location list, creating a combined location list and

Algorithm 72 $((\alpha_f,\ \bar{l}_f,\ \bar{j}_f),\ j_f) \leftarrow \text{DerefPrivPtr}(\alpha,\ \bar{l},\ \bar{j},\ \text{private } bty*,\ \sigma)$

```
1: if (l_default, 0) ∈ l̄ then
2:     ERROR
3:     return ((0, [ ], [ ]), −1)
4: end if
5: ((α_f, l̄_f, j̄_f), j_f) = ((0, [ ], [ ]), 1)
6: if (α = 1) ∧ ([(l, μ)] = l̄) then
7:     ([α_f, l̄_f, j̄_f, i], j_f) = DerefPtrHLI(σ, private bty*, (l, μ))
8: else
9:     [α_k, l̄_k, j̄_k, i_k] = [1, [(l_default, 0)], [1], 1]
10:    for all (l_k, μ_k) ∈ l̄ do
11:        (ω_k, ty_k, α_k, PermL(Freeable, ty_k, a_k, α_k) = σ(l_k)
12:        if (μ_k = 0) ∧ (private bty*) then
13:            [α_k, l̄_k, j̄_k, i_k] = DecodePtr(private bty*, α_k, ω_k)
14:        else
15:            j_f = 0
16:            [α_k, l̄_k, j̄_k, i_k] = GetBytes((l_k, μ_k), private bty*, σ)
17:        end if
18:        for all l_m ∈ l̄_k do
19:            if (l_m ∈ l̄_f) then
20:                pos = l̄_f.find(l_m])
21:                j̄_f[pos] = (j̄[k] ∧ j̄_k[m]) ∨ (¬j̄[k] ∧ j̄_k[m])
22:            else
23:                l̄_f.append(l_m)
24:                j̄_f.append(j̄[k] ∧ j̄_k[m])
25:            end if
26:        end for
27:    end for
28:    α_f = |l̄_f|
29: end if
30: return ((α_f, l̄_f, j̄_f), j_f)
```

corresponding tag list. Finally, we return the final number of locations, location list, and tag list that we've obtained by dereferencing the locations referred to by the pointer, as well as tag j_f to indicate whether any of the dereferences were not aligned.

Algorithm 73 (Retrieve_vals) is designed to privately obtain the value from the true location for a private pointer dereference. It takes as input the number of locations the pointer refers to, the list of locations and list of tags for the pointer, the private type that is expected to be at each location, and memory. First, we check that we will not be trying to dereference from the default location, which is not valid (i.e., this would cause a segmentation fault). If we are, we return with the tag -1, which will not allow the statement that triggered this to resolve any further. We then iterate through all of the locations for the pointer, obtaining the value at each location by interpreting the bytes at the location as the expected type. We take this value and and perform a private computation of whether to keep this value or not (i.e., whether this was from the true location or not) based on the list of tags for the pointer. Once we have touched all locations, the algorithm returns the final

Algorithm 73 $(v_f,\ j_f) \leftarrow \text{Retrieve_vals}(\alpha,\ \bar{l},\ \bar{j},\ \text{private } bty,\ \sigma)$

```
1: if (l_default ∈ l̄) then
2:     ERROR
3:     return (NULL, −1)
4: end if
5: j_f = 1
6: v_f = encrypt(0)
7: for all (l_m, μ_m) ∈ l̄ do
8:     (ω_m, ty_m, α_m, PermL(Freeable, ty_m, a_m, α_m) = σ(l_m)
9:     if (μ_m = 0) ∧ (ty_m = private bty) ∧ (α_m = 1) then
10:        v_m = DecodeVal(private bty, 1, ω_m)
11:        v_f = (j̄[m] ∧ v_m) ∨ (¬j̄[m] ∧ v_f)
12:    else if (μ_m < α_m) ∧ (ty_m = private bty) then
13:        [v_0, ..., v_{α_m−1}] = DecodeVal(private bty, α_m, ω_m)
14:        v_f = (j̄[m] ∧ v_{μ_m}) ∨ (¬j̄[m] ∧ v_f)
15:    else
16:        v_m = GetBytes((l_m, μ_m), private bty, σ)
17:        v_f = (j̄[m] ∧ v_m) ∨ (¬j̄[m] ∧ v_f)
18:        j_f = 0
19:    end if
20: end for
21: return (v_f, j_f)
```

private value we've obtained from memory and the tag indicating whether the dereference was *well-aligned* or not.

Algorithm 74 $\sigma_f \leftarrow \text{Free}(\sigma_1,\ l,\ \gamma)$

```
1: (ω, a bty*, 1, PermL(Freeable, a bty*, a, α)) = σ_1(l)
2: [1, [(l_1, μ_1)], [j_1], i] = DecodePtr(a bty*, 1, ω)
3: if CheckFreeable(γ, [(l_1, μ_1)], [j_1], σ_1) then
4:     σ_2[l_1 → (ω, ty, 1, PermL(Freeable, ty, a, 1))] = σ_1
5:     σ_f = σ_2[l_1 → (ω, ty, 1, PermL(None, ty, a, 1))]
6: else
7:     σ_f = σ_1
8: end if
9: return σ_f
```

Algorithm 74 (Free) corresponds to conventional memory deallocation when we call free to deallocate memory associated with some pointer. In particular, on input location l, we first check whether the location corresponds to memory that can be deallocated using Algorithm 76 (CheckFreeable). If CheckFreeable returns 1, we will mark location l as unavailable. Otherwise, calling Free has no effect on the state of memory.

Algorithm 75 (PFree) corresponds to deallocating memory associated with a pointer to private data which may be associated with multiple locations where the data may actually reside. It distinguishes between two main cases: when the number of locations associated with a pointer is 1 (i.e., $\alpha = 1$) and when it is larger than 1. In the former case, we invoke Free from Algorithm 74 as described above. Otherwise, the true location is not known and the location to be removed should be chosen based on public knowledge. For the purposes of

Algorithm 75 $(\sigma_f, \bar{l}, \bar{j}) \leftarrow \text{PFree}(\gamma, \sigma,\ l)$

1: $(\omega,\ \text{private } bty*,\ \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) = \sigma(l)$
2: **if** $(\alpha = 1)$ **then**
3: $\quad [1,\ [(l_1, \mu_1)],\ [j_1],\ 1] = \text{DecodePtr}(a\ bty*,\ 1,\ \omega)$
4: $\quad \sigma_f = \text{Free}(\sigma, l, \gamma)$
5: $\quad$ **return** $(\sigma_f, [(l_1, \mu_1)], [0])$
6: **else**
7: $\quad [\alpha,\ [(l_0, 0), ..., (l_{\alpha-1}, 0)],\ [j_0, ..., j_{\alpha-1}],\ 1] = \text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega)$
8: $\quad$ **if** $\neg\text{CheckFreeable}(\gamma, [l_0, ..., l_{\alpha-1}], [j_0, ..., j_{\alpha-1}], \sigma)$ **then**
9: $\quad\quad$ **return** $(\sigma, [l_0, ..., l_{\alpha-1}], [0])$
10: $\quad$ **end if**
11: $\quad \sigma_1[l_0 \rightarrow (\omega_0,\ \text{private } bty,\ 1,\ \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))] = \sigma$
12: $\quad \omega'_0 = \omega_0$
13: $\quad \bar{j} = [0, j_1, ..., j_{\alpha-1}]$
14: $\quad$ **for all** $l_m \in [l_1, ..., l_{\alpha-1}]$ **do**
15: $\quad\quad \sigma_2[l_m \rightarrow (\omega_m,\ \text{private } bty,\ 1,\ \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))] = \sigma_1$
16: $\quad\quad \omega'_m = (\omega_m \wedge \neg j_m) \vee (\omega_0 \wedge j_m)$
17: $\quad\quad \omega'_0 = (\omega_m \wedge j_m) \vee (\omega'_0 \wedge \neg j_m)$
18: $\quad\quad \sigma_3 = \sigma_2[l_m \rightarrow (\omega'_m,\ \text{private } bty,\ 1,\ \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))]$
19: $\quad\quad \sigma_1 = \sigma_3$
20: $\quad\quad \bar{j}[0] = \bar{j}[0] \vee j_m$
21: $\quad$ **end for**
22: $\quad \sigma_4 = \sigma_3[l_0 \rightarrow (\omega'_0,\ \text{private } bty,\ 1,\ \text{PermL}(\text{None}, \text{private } bty, \text{private}, 1))]$
23: $\quad \sigma_f = \text{UpdatePointerLocations}(\sigma_4, [(l_1, 0), ..., (l_{\alpha-1}, 0)],\ [j_1, ..., j_{\alpha-1}], l_0, j_0)$
24: $\quad$ **return** $(\sigma_f, [l_0, ..., l_{\alpha-1}], \bar{j})$
25: **end if**

this functionality, and without loss of generality, we deallocate the first location on the list. However, before we can proceed, we need to ensure that memory is in fact deallocatable. Thus, similar to Algorithm 74 we first call CheckFreeable to determine whether deallocation is permitted. Unlike calling CheckFreeable from Algorithm 74, we pass the entire list of α to locations to that function (note that CheckFreeable is written to take a list of locations, but we call it on a list consisting of a single entry from Free). This time, if locations which cannot be deallocated (e.g., the default location $l_{default}$ which corresponds to uninitialized pointers and locations associated with statically allocated variables) appear on the list, the pointer is viewed as not deallocatable because it may point to locations on which free has no effect. In this case, no changes are performed to the state and we return the original list of locations.

If CheckFreeable returns 1, we can proceed with deallocating memory associated with one of the locations, i.e., l_0. If l_0 is the true location of the pointer, this is all that is necessary. However, if l_0 is not the true location, deallocating it requires additional work because that location might be in use by other pointers. That is, based on the fact that freeing a pointer has been called, we know that the true location can be released, but it might not be safe to deallocate other locations associated with the pointer. For that reason, in Algorithm 75 we iterate through all locations l_1 through $l_{\alpha-1}$ and swap the content of the current location

l_m and l_0 if l_m is in fact the true location (i.e., flag j_m is set). That is, ω'_m corresponds to the updated content of location l_m: the content will remain unchanged if j_m is not set, and otherwise, it will be replaced with the content of location l_0. Similarly, ω'_0 corresponds to the updated content of location l_0. Note that it may be modified in at most one iteration of the loop, namely, when j_m is set. All other iterations will keep the value unchanged (and it will never be modified if none of the table $j_1, \ldots, j_{\alpha-1}$ are set and j_0 is instead 1). The function is written to be data-oblivious, i.e., to not reveal the true location associated with the pointer. Once the content of the locations has been updated, we can mark location l_0 as inaccessible, remove it from the list, and call function $\mathrm{UpdatePointerLocations}$ in Algorithm 77 with the remaining locations $l_1, \ldots, l_{\alpha-1}$.

Algorithm 76 $j \leftarrow \mathrm{CheckFreeable}(\gamma, \bar{l}, \bar{j}, \sigma)$

```
 1: if (l_default, 0) ∈ l̄ then
 2:     return 0
 3: end if
 4: for all (l_m, μ_m) ∈ l̄ do
 5:     if μ_m ≠ 0 then
 6:         return 0
 7:     end if
 8: end for
 9: if 1 ∉ j̄ then
10:     return 0
11: end if
12: for all x ∈ γ do
13:     (l_x, ty_x) = γ(x)
14:     if (l_x, 0) ∈ l̄ then
15:         return 0
16:     else if ty_x = a const bty* then
17:         (ω, ty_x, 1, PermL(Freeable, ty_x, a, 1)) = σ(l_x)
18:         [1, [(l_1, 0)], [1], 1] = DecodePtr(ty_x, 1, ω)
19:         if (l_1, 0) ∈ l̄ then
20:             return 0
21:         end if
22:     end if
23: end for
24: return 1
```

Algorithm 76 ($\mathrm{CheckFreeable}$) follows the behavior expected of free: if the location was properly allocated via a call to malloc, it is de-allocatable for the purpose of this function. In particular, the default location $l_{default}$ that corresponds to uninitialized pointers is not de-allocatable (and freeing such a pointer has no effect); similarly memory associated with statically declared variables is not de-allocatable via this mechanism (and freeing it here also has no effect). Thus, if $\mathrm{CheckFreeable}$ returns 1, we will proceed to deallocate a location, otherwise, there will be no effect on the state of memory as we cannot safely perform the deallocation operation.

In Algorithm 77 ($\mathrm{UpdatePointerLocations}$) we are given location l_r which is being removed and a

Algorithm 77 $\sigma_1 \leftarrow \text{UpdatePointerLocations}(\sigma, \bar{l}, \bar{j}, l_r, j_r)$

1: $\sigma_1 = [\,]$
2: **for all** $l_k \in \sigma$ **do**
3: $\quad(\omega_k,\ ty,\ n,\ \text{PermL}(\text{Freeable}, ty, a, n)) = \sigma(l_k)$
4: $\quad$**if** $(ty = \text{private } bty*)$ **then**
5: $\quad\quad[n,\ \bar{l}_k,\ \bar{j}_k,\ i] = \text{DecodePtr}(\text{private } bty*,\ n,\ \omega)$
6: $\quad\quad$**if** $l_r \in \bar{l}_k$ **then**
7: $\quad\quad\quad pos = \bar{l}_k.\text{find}(l_r)$
8: $\quad\quad\quad \bar{j}'_k = \bar{j}_k \setminus j_{k_pos}$
9: $\quad\quad\quad \bar{l}'_k = \bar{l}_k \setminus l_r$
10: $\quad\quad\quad [\alpha_{new}, \bar{l}_{new}, \bar{j}_{new}] = \text{CondAssign}([|\bar{l}|, \bar{l}, \bar{j}], [n-1, \bar{l}'_k, \bar{j}'_k], j_{k_pos})$
11: $\quad\quad\quad \omega'_k = \text{EncodePtr}(\text{private } bty*, [\alpha_{new}, \bar{l}_{new}, \bar{j}_{new}, i])$
12: $\quad\quad\quad \sigma_1 = \sigma_1[l_k \rightarrow (\omega'_k,\ ty,\ n,\ \text{PermL}(\text{Freeable}, ty, a, n))]$
13: $\quad\quad$**else**
14: $\quad\quad\quad \sigma_1 = \sigma_1[l_k \rightarrow (\omega_k,\ ty,\ n,\ \text{PermL}(\text{Freeable}, ty, a, n))]$
15: $\quad\quad$**end if**
16: $\quad$**else**
17: $\quad\quad \sigma_1 = \sigma_1[l_k \rightarrow (\omega_k,\ ty,\ n,\ \text{PermL}(\text{Freeable}, ty, a, n))]$
18: $\quad$**end if**
19: **end for**

list of other locations $\bar{l}$ associated with the pointer in question. In the event that l_r was not the true pointer location, its content has been moved to another location, but it still may remain in the lists of other pointers, which is what this function is to correct. In particular, the function iterates through other pointers in the system and searches for location l_r in their lists. If l_r is present (i.e., $l_r \in \bar{l}_k$), we need to remove it and replace it with another location from $\bar{l}$ to which the data has been moved. However, because we do not know which location in L is set and contains the relevant data, we are left with merging all locations in L with the pointer's current locations $\bar{l}'_k$ after removing l_r. This is done using function CondAssign from Algorithm 78 as described next.

Algorithm 78 (CondAssign) takes two pointer data structures with the associated number of locations, lists of locations, and lists of tags as well as a flag n_{res}. Its primary purpose is to merge two pointer data structures during the execution of conditional statements with private conditions. Here, n_{res} is a flag that indicates whether the true pointer location should be taken from the first or the second data structure; $n_{res} = 1$ means that the true location is in the first data structure. For example, when executing code `if (priv) p1 = p2;`, n_{res} is the result of evaluating private condition `priv`, the first data structure corresponds to `p1`'s data structure prior to executing this statement, and the second data structure corresponds to `p2`'s data structure. The function first computes the union of the two lists of locations and then updates their corresponding tags based on their tags at the time of calling this function and the value of n_{res}. For example, if a particular location l_m is found on both lists, we retain its tag from the first list if n_{res} is set and otherwise

retain its tag from the second list if n_{res} is not set. When l_m is found only in one of the lists, we use a similar logic and conditionally retain its original tag based on the value of n_{res}. If a tag is not retained, it is reset to 0. This ensures that for any pointer data structure only one tag is set to 1 and all others are set to 0.

Algorithm 78 $[\alpha_3,\ \bar{l}_3,\ \bar{j}_3] \leftarrow \mathrm{CondAssign}([\alpha_1,\ \bar{l}_1,\ \bar{j}_1],\ [\alpha_2,\ \bar{l}_2,\ \bar{j}_2],\ n_{res})$

1: $\bar{l}_3 = \bar{l}_1 \cup \bar{l}_2$
2: $\alpha_3 = |\bar{l}_3|$
3: $\bar{j}_3 = [\,]$
4: **for all** $l_m \in \bar{l}_3$ **do**
5: $\quad pos_1 = \bar{l}_1.\mathrm{find}(l_m)$
6: $\quad pos_2 = \bar{l}_2.\mathrm{find}(l_m)$
7: $\quad$ **if** $(pos_1\ \wedge\ pos_2)$ **then**
8: $\quad\quad j''_m = (n_{res} \wedge j'_{pos_2}) \vee (\neg\, n_{res} \wedge j_{pos_1})$
9: $\quad$ **else if** $(\neg\, pos_2)$ **then**
10: $\quad\quad j''_m = \neg\, n_{res} \wedge j_{pos_1}$
11: $\quad$ **else**
12: $\quad\quad j''_m = n_{res} \wedge j'_{pos_2}$
13: $\quad$ **end if**
14: $\quad \bar{j}_3.\mathrm{append}(j''_m)$
15: **end for**
16: **return** $[\alpha_3,\ \bar{l}_3,\ \bar{j}_3]$

Returning to our use of Algorithm 78 ($\mathrm{CondAssign}$) in Algorithm 77 ($\mathrm{UpdatePointerLocations}$), notice that we are also merging two pointer data structures based on a flag. This time the flag is $j_{k_{pos}}$, which indicates whether the true location is in the first or second list of locations. That is, if l_r was the true location of the pointer, the data has been moved and resides in one of the locations in $\bar{l}$. Otherwise, if l_r was not the true location, the data resides at one of the remaining locations associated with the pointer on its location list $\bar{l}'_k$. Thus, we merge the list of locations and update the corresponding tags in the same way this is done during evaluation of conditional statements with private conditions.

3.2 Correctness

The most challenging result is correctness, which we discuss first. Once correctness is proven, noninterference follows from a standard argument, with some adaptations needed to deal with the fact that private data is encrypted and that we want to show indistinguishability of evaluation traces.

We show the correctness of the Basic SMC^2 semantics with respect to the Vanilla C semantics. As usual we will do this by establishing a simulation relation between a Basic SMC^2 program and a corresponding Vanilla C program. To do so we face two main challenges. First, we need to guarantee that the private operations in a Basic SMC^2 program are reflected in the corresponding Vanilla C pro-

$SmcC$	$VanC$	$SmcC$	$VanC$	$SmcC$	$VanC$	$SmcC$	$VanC$	$SmcC$	$VanC$
$\Downarrow_{r}$	$\Downarrow'_{\widehat{r}}$	$\Downarrow_{r1}$	$\Downarrow'_{\widehat{r}}$	$\Downarrow_{ss}$	$\Downarrow'_{\widehat{ss}}$	$\Downarrow_{cv}$	$\Downarrow'_{\widehat{cv}}$	$\Downarrow_{cv1}$	$\Downarrow'_{\widehat{cv}}$
$\Downarrow_{cl}$	$\Downarrow'_{\widehat{cl}}$	$\Downarrow_{cl1}$	$\Downarrow'_{\widehat{cl}}$	$\Downarrow_{w}$	$\Downarrow'_{\widehat{w}}$	$\Downarrow_{w1}$	$\Downarrow'_{\widehat{w}}$	$\Downarrow_{w2}$	$\Downarrow'_{\widehat{w}}$
$\Downarrow_{ds}$	$\Downarrow'_{\widehat{ds}}$	$\Downarrow_{d}$	$\Downarrow'_{\widehat{d}}$	$\Downarrow_{d1}$	$\Downarrow'_{\widehat{d}}$	$\Downarrow_{dp}$	$\Downarrow'_{\widehat{dp}}$	$\Downarrow_{dp1}$	$\Downarrow'_{\widehat{dp}}$
$\Downarrow_{da}$	$\Downarrow'_{\widehat{da}}$	$\Downarrow_{da1}$	$\Downarrow'_{\widehat{da}}$	$\Downarrow_{ltf2}$	$\Downarrow'_{\widehat{ltf}}$	$\Downarrow_{bp}$	$\Downarrow'_{\widehat{bp}}$	$\Downarrow_{bp1}$	$\Downarrow'_{\widehat{bp}}$
$\Downarrow_{bp2}$	$\Downarrow'_{\widehat{bp}}$	$\Downarrow_{bp3}$	$\Downarrow'_{\widehat{bp}}$	$\Downarrow_{ltf}$	$\Downarrow'_{\widehat{ltf}}$	$\Downarrow_{ltf1}$	$\Downarrow'_{\widehat{ltf}}$	$\Downarrow_{ltt2}$	$\Downarrow'_{\widehat{ltt}}$
$\Downarrow_{ltf3}$	$\Downarrow'_{\widehat{ltf}}$	$\Downarrow_{ltt}$	$\Downarrow'_{\widehat{ltt}}$	$\Downarrow_{ltt3}$	$\Downarrow'_{\widehat{ltt}}$	$\Downarrow_{ltt1}$	$\Downarrow'_{\widehat{ltt}}$	$\Downarrow_{eqf}$	$\Downarrow'_{\widehat{eqf}}$
$\Downarrow_{eqf1}$	$\Downarrow'_{\widehat{eqf}}$	$\Downarrow_{eqf2}$	$\Downarrow'_{\widehat{eqf}}$	$\Downarrow_{eqf3}$	$\Downarrow'_{\widehat{eqf}}$	$\Downarrow_{eqt}$	$\Downarrow'_{\widehat{eqt}}$	$\Downarrow_{eqt1}$	$\Downarrow'_{\widehat{eqt}}$
$\Downarrow_{eqt3}$	$\Downarrow'_{\widehat{eqt}}$	$\Downarrow_{nef}$	$\Downarrow'_{\widehat{nef}}$	$\Downarrow_{nef1}$	$\Downarrow'_{\widehat{nef}}$	$\Downarrow_{nef2}$	$\Downarrow'_{\widehat{nef}}$	$\Downarrow_{eqt2}$	$\Downarrow'_{\widehat{eqt}}$
$\Downarrow_{net}$	$\Downarrow'_{\widehat{net}}$	$\Downarrow_{net1}$	$\Downarrow'_{\widehat{net}}$	$\Downarrow_{net3}$	$\Downarrow'_{\widehat{net}}$	$\Downarrow_{net2}$	$\Downarrow'_{\widehat{net}}$	$\Downarrow_{nef3}$	$\Downarrow'_{\widehat{nef}}$
$\Downarrow_{fre}$	$\Downarrow'_{\widehat{fre}}$	$\Downarrow_{frep}$	$\Downarrow'_{\widehat{fre}}$	$\Downarrow_{rp}$	$\Downarrow'_{\widehat{rp}}$	$\Downarrow_{rp1}$	$\Downarrow'_{\widehat{rp}}$	$\Downarrow_{rp2}$	$\Downarrow'_{\widehat{rp}}$
$\Downarrow_{wp}$	$\Downarrow'_{\widehat{wp}}$	$\Downarrow_{wp1}$	$\Downarrow'_{\widehat{wp}}$	$\Downarrow_{wp2}$	$\Downarrow'_{\widehat{wp}}$	$\Downarrow_{wdp3}$	$\Downarrow'_{\widehat{wdp1}}$	$\Downarrow_{wdp5}$	$\Downarrow'_{\widehat{wdp1}}$
$\Downarrow_{wdp}$	$\Downarrow'_{\widehat{wdp}}$	$\Downarrow_{wdp1}$	$\Downarrow'_{\widehat{wdp1}}$	$\Downarrow_{wdp2}$	$\Downarrow'_{\widehat{wdp1}}$	$\Downarrow_{wdp4}$	$\Downarrow'_{\widehat{wdp1}}$	$\Downarrow_{rdp}$	$\Downarrow'_{\widehat{rdp}}$
$\Downarrow_{rdp1}$	$\Downarrow'_{\widehat{rdp1}}$	$\Downarrow_{rdp2}$	$\Downarrow'_{\widehat{rdp1}}$	$\Downarrow_{rdp3}$	$\Downarrow'_{\widehat{rdp1}}$	$\Downarrow_{inp}$	$\Downarrow'_{\widehat{inp}}$	$\Downarrow_{inp3}$	$\Downarrow'_{\widehat{inp}}$
$\Downarrow_{out}$	$\Downarrow'_{\widehat{out}}$	$\Downarrow_{out1}$	$\Downarrow'_{\widehat{out1}}$	$\Downarrow_{inp4}$	$\Downarrow'_{\widehat{inp1}}$	$\Downarrow_{out3}$	$\Downarrow'_{\widehat{out}}$	$\Downarrow_{out4}$	$\Downarrow'_{\widehat{out1}}$
$\Downarrow_{ra}$	$\Downarrow'_{\widehat{ra}}$	$\Downarrow_{ra3}$	$\Downarrow'_{\widehat{ra}}$	$\Downarrow_{ra1}$	$\Downarrow'_{\widehat{ra}}$	$\Downarrow_{ra2}$	$\Downarrow'_{\widehat{ra}}$	$\Downarrow_{wa3}$	$\Downarrow'_{\widehat{wa}}$
$\Downarrow_{wa1}$	$\Downarrow'_{\widehat{wa}}$	$\Downarrow_{wa2}$	$\Downarrow'_{\widehat{wa}}$	$\Downarrow_{wa}$	$\Downarrow'_{\widehat{wa}}$	$\Downarrow_{wa4}$	$\Downarrow'_{\widehat{wa}}$	$\Downarrow_{wao1}$	$\Downarrow'_{\widehat{wao}}$
$\Downarrow_{rao1}$	$\Downarrow'_{\widehat{rao}}$	$\Downarrow_{rao}$	$\Downarrow'_{\widehat{rao}}$	$\Downarrow_{df}$	$\Downarrow'_{\widehat{df}}$	$\Downarrow_{wao}$	$\Downarrow'_{\widehat{wao}}$	$\Downarrow_{wao2}$	$\Downarrow'_{\widehat{wao}}$
$\Downarrow_{pin3}$	$\Downarrow'_{\widehat{pin3}}$	$\Downarrow_{pin7}$	$\Downarrow'_{\widehat{pin3}}$	$\Downarrow_{pin4}$	$\Downarrow'_{\widehat{pin3}}$	$\Downarrow_{pin}$	$\Downarrow'_{\widehat{pin}}$	$\Downarrow_{pin1}$	$\Downarrow'_{\widehat{pin}}$
$\Downarrow_{pin5}$	$\Downarrow'_{\widehat{pin3}}$	$\Downarrow_{pin2}$	$\Downarrow'_{\widehat{pin2}}$	$\Downarrow_{pin6}$	$\Downarrow'_{\widehat{pin2}}$	$\Downarrow_{wle}$	$\Downarrow'_{\widehat{wle}}$	$\Downarrow_{wlc}$	$\Downarrow'_{\widehat{wlc}}$
$\Downarrow_{bs}$	$\Downarrow'_{\widehat{bs}}$	$\Downarrow_{bs2}$	$\Downarrow'_{\widehat{bs}}$	$\Downarrow_{bs1}$	$\Downarrow'_{\widehat{bs}}$	$\Downarrow_{mal}$	$\Downarrow'_{\widehat{mal}}$	$\Downarrow_{malp}$	$\Downarrow'_{\widehat{mal}}, \Downarrow'_{\widehat{bm}}, \Downarrow'_{\widehat{ty}}$
$\Downarrow_{bs3}$	$\Downarrow'_{\widehat{bs}}$	$\Downarrow_{loc}$	$\Downarrow'_{\widehat{loc}}$	$\Downarrow_{sb}$	$\Downarrow'_{\widehat{sb}}$	$\Downarrow_{ty}$	$\Downarrow'_{\widehat{ty}}$	$\Downarrow_{ep}$	$\Downarrow'_{\widehat{ep}}$
$\Downarrow_{iet}$	$\Downarrow'_{\widehat{iet}}$	$\Downarrow_{ief}$	$\Downarrow'_{\widehat{ief}}$	$\Downarrow_{fd}$	$\Downarrow'_{\widehat{fd}}$	$\Downarrow_{fc1}$	$\Downarrow'_{\widehat{fc}}$	$\Downarrow_{fc}$	$\Downarrow'_{\widehat{fc}}$
$\Downarrow_{fpd}$	$\Downarrow'_{\widehat{fpd}}$								

Figure 3.30: Table of SMC^2 evaluation codes in $SmcC \setminus SmcCX$ and their congruent Vanilla C evaluation codes in $VanC \setminus VanCX$.

gram and that the evaluation steps between the two programs correspond. To address the former issue, we define an *erasure function* **Erase** which translates a Basic SMC^2 program into a Vanilla C program by erasing all labels and replacing all functions specific to Basic SMC^2 with their public equivalents. This function also translates memory. As an example, let us consider `pmalloc`; in this case, we have $\mathbf{Erase}(\text{pmalloc}(e,\ ty) = (\text{malloc}(\text{sizeof}(\mathbf{Erase}(ty)) \cdot \mathbf{Erase}(e))))$. That is, `pmalloc` is rewritten to use `malloc`, and since the given private type is now public we can use the sizeof function to find the size we will need to allocate. To address the latter issue, we have defined our operational semantics in terms of big-step evaluation judgments which allow the evaluation trees of the two programs to have a corresponding structure. In particular, notice how we designed the Private If Else rule to perform multiple operations at one step, guaranteeing that we have similar "synchronization points" in the two evaluation trees.

Second, we need to guarantee that at each evaluation step the memory used by a Basic SMC^2 program corresponds to the one used by the Vanilla C program. In our setting, with explicit memory management, manipulations of pointers, and array overshooting, the latter becomes particularly challenging. To better

understand the issue here, let us consider the the rule Private Free. Remember that our semantic model associates a pointer with a list of locations, and the Private Free rule frees the first location in the list, and relocates the content of that location if it is not the true location. Essentially, this rule may swap the content of two locations if the first location in the list is not the location intended to be freed and make the Basic SMC^2 memory and the Vanilla C memory look quite different. To address this challenge in the proof of correctness, we use a *map*, denoted ψ, to track the swaps that happen when the rule Private Free is used. The simulation uses and modifies this map to guarantee that the two memories correspond.

Another related challenge comes from array overshooting. If, by overshooting an array, a program goes over or into memory blocks of different types, we may end up in a situation where the locations in the Basic SMC^2 memory are significantly different from the ones in the Vanilla C memory. This is mostly due to the size of private types being larger than their public counterpart. One option to address this problem would be to keep a more complex map between the two memories. However, this can result in a much more complex proof, for capturing a behavior that is faulty, in principle. Instead, we prefer to focus on situations where overshooting arrays are *well-aligned*, in the sense that they access only memory locations and blocks of the right type and size. An illustration of this is given in Figure 3.4.

Before stating our correctness, we need to introduce some notation. We use codes $[d_1, \ldots, d_n]$, $[\widehat{d}_1, \ldots, \widehat{d}_m]$ in evaluations (i.e., $\Downarrow_{[d_1,\ldots,d_n]}$) to describe the rules of the semantics that are applied in order to derive the result. We write $[d_1, \ldots, d_n] \cong [\widehat{d}_1, \ldots, \widehat{d}_m]$ to state that the Basic SMC^2 codes $[d_1, \ldots, d_n]$ are in correspondence with the Vanilla C codes $[\widehat{d}_1, \ldots, \widehat{d}_m]$. Almost every Basic SMC^2 rule is in one-to-one correspondence with a single Vanilla C rule within an execution trace (exceptions being private-conditioned branches and `pmalloc`).

We write $s \cong \widehat{s}$ to state that the Vanilla C configuration statement $\widehat{s}$ can be obtained by applying the erasure function to the Basic SMC^2 statement s. Similarly, we can extend this notation to configuration by also using the map ψ. That is, we write $(\gamma, \sigma, \mathrm{acc}, s) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$ to state that the Vanilla C configuration $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$ can be obtained by applying the erasure function to the Basic SMC^2 configuration $(\gamma, \sigma, \mathrm{acc}, s)$, and memory $\widehat{\sigma}$ can be obtained from σ by using the map ψ.

We state correctness in terms of evaluation trees, since we will use evaluation trees to prove a strong form of noninterference in the next subsection. We use capital greek letters Π, Σ to denote evaluation trees. In the Basic SMC^2 semantics, we write $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, s) \Downarrow_{[d_1,\ldots,d_n]} (\gamma_1, \sigma_1, \mathrm{acc}_1, v)$, to stress that the evaluation tree Π proves as conclusion that configuration $(\gamma, \sigma, \mathrm{acc}, s)$ evaluates to configuration $(\gamma_1, \sigma_1, \mathrm{acc}_1, v)$

by means of the codes $[d_1, ..., d_n]$. Similarly, for the Vanilla C semantics. We then write $\Pi \cong_\psi \Sigma$ for the extension to evaluation trees of the congruence relation with map ψ.

We can now state our correctness result showing that if a Basic SMC^2 program s can be evaluated to a value v, and the evaluation is well-aligned (it is an evaluation where all the overshooting of arrays are well-aligned), then the Vanilla C program $\hat{s}$ obtained by applying the erasure function to s, i.e., $s \cong \hat{s}$ can be evaluated to $\hat{v}$ where $v \cong \hat{v}$. This property can be formalized in terms of congruence:

Theorem 3.2.1 (Semantic Correctness)**.** *Given configuration* $(\gamma, \sigma, \text{acc}, s)$*,* $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$ *and map* ψ *such that* $(\gamma, \sigma, \text{acc}, s) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$*, if* $\Pi \triangleright (\gamma, \sigma, \text{acc}, s) \Downarrow_{[d_1,...,d_n]} (\gamma_1, \sigma_1, \text{acc}, v_1)$ *for codes* $[d_1, ..., d_n] \in SmcC \setminus SmcCX$*, then there exists a derivation* $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s}) \Downarrow_{[\widehat{d}_1,...,\widehat{d}_m]} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v}_1)$ *for codes* $[\widehat{d}_1, ..., \widehat{d}_m] \in VanC \setminus VanCX$ *and a map* ψ_1 *such that* $[d_1, ..., d_n] \cong [\widehat{d}_1, ..., \widehat{d}_m]$*,* $(\gamma_1, \sigma_1, \text{acc}, v_1) \cong_{\psi_1} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v}_1)$*, and* $\Pi \cong_{\psi_1} \Sigma$*.*

Proof. Proof Sketch: By induction over all Basic SMC^2 semantic rules.

The bulk of the complexity of this proof lies with rules pertaining to Private If Else, handling of pointers, and freeing of memory. We first provide a brief overview of the intuition behind some assumptions we must make for the proof and reasoning behind the use of some of the elements of the rules; then we dive deeper into the details for the more complex cases. The full proof is available in Section 3.2.4, with this theorem identical to Theorem 3.2.2.

First, we need to assume private indexing is within bounds. Otherwise, we will not be able to prove correctness, because when using private indexing we will not go out of the bounds of the array in Basic SMC^2, whereas the Vanilla C equivalent would. We also need to assume that input files are congruent, otherwise we cannot reason over the data input functions.

Similarly, when reasoning about rules containing overshooting and offsets into memory blocks, we must assert that such operations are *well-aligned* by type (i.e., for overshooting, we can only assert correctness over memory blocks and elements of the same type, and for offsets, the offset must be aligned with the start of an element within the block, and the expected and actual types of memory the same). Going over or into memory blocks of different types could cause significantly different locations between Basic SMC^2 and Vanilla C due to private types being larger in size. An illustration of this is shown in Figure 3.4.

The correctness of most semantic rules follows easily, with Private Free being a notable exception. We leverage the correctness of Algorithm 75 (PFree), to show that correctness follows due to the deterministic definitions of this algorithm and those used by this algorithm. In this case, we must also show that the locations that are swapped within this rule, which is done to hide the true location, are deterministic based on our memory model definition. We use ψ to map the swapped locations, enabling us to show that, if these swaps were reversed, we would once again have memories that are directly congruent. This concept of locations being ψ-congruent is particularly necessary when reasoning about

pointers in other rule cases.

We make the assertion that $v \neq \text{skip}$ in some rules where skip is not allowed as a value - this is especially important for asserting that an expression cannot contain `pfree(e)` (as any expression containing `pfree(e)` would evaluate to skip) and thus that ψ could not have been modified over said evaluation.

Another common assertion we must make is that in a semantic rule, we do not accept a hard-coded location (l, μ), $(\widehat{l}, \widehat{\mu})$ as the starting statement s, $\widehat{s}$. Hard-coded locations could lead to evaluating locations that are not congruent to each other and therefore we would not be able to prove correctness over such statements. This makes it so we can easily assert that our starting statements are congruent (i.e., $s \cong \widehat{s}$).

For all the rules using private pointers, we will rely upon the pointer data structure containing a set of locations and their associated tags, only one of which being the true location. With this proven to be the case, it is then clear that the true location indicated within the private pointer's data structure in Basic SMC^2 will be ψ-congruent with the location given by the pointer data structure in Vanilla C. We define this correspondence between locations as location ψ-congruence - ensuring that memory block IDs are the same, and the position into the block is congruent (same if public, proportional if private

For rule Private Malloc, we must relate this rule to the sequence of Vanilla C rules for Malloc, Multiplication, and Size of Type. This is due to the definition of `pmalloc` as a helper that allows the user to write programs without knowing the size of private types. This case follows from the definition of translating the Basic SMC^2 program to a Vanilla C program, $\mathbf{Erase}(\text{pmalloc}(e,\ ty) = (\text{malloc}(\text{sizeof}(\mathbf{Erase}(ty)) \cdot \mathbf{Erase}(e))))$.

For the Private If Else rule, we must reason that our end results in memory after executing both branches and resolving correctly match the end result of having only executed the intended branch. The cases for both of these rules will have two subcases - one for the conditional being true, and the other for false. We must first show that ExtractVariables will correctly find all non-local variables that are modified within both branches, including non-assignment modifications such as use of the pre-increment operator $++x$, and that all such modified variables will be added to the list. These properties follow deterministically from the definition of the algorithm.

We then reason that $\text{InitializeVariables}$ will correctly create the assignment statements for our temporary variables, and that the original values for each of the modified variables will be stored into the **`else`** temporary variables. When we allocate these temporary variables, we place them into a specific portion of memory designated for such temporaries. In this way, we can easily maintain our congruence between locations. It is possible to introduce an additional tracking structure to maintain a mapping between all Basic SMC^2 locations and Vanilla C locations so that we do not need to place such temporaries into a specific portion of memory, but proving such a tracking structure and mapping scheme correct is trivial and introduces unnecessary complications in reasoning about congruence between memories.

Next we have the evaluation of the **`then`** branch, which will result in the values that are correct for if the condition had been true - this holds by induction. We then proceed to reason that RestoreVariables will properly create the

statements to store the ending results of the **then** branch into the **then** temporary variables, and restore all of the original values from the **else** variables (the original values being correctly stored follows from InitializeVariables and the evaluation of it's statements). The correct evaluation of the this set of statements follows by induction. Next we have the evaluation of the **else** branch, which will result in the values that are correct for if the condition had been false - this holds by induction and the values having been restored to the original values properly. We will then reason about the correctness of ResolveVariables. It must be set up to correctly take the information from the **then** temporary variable, the temporary variable for the condition for the branch, and the ending result for all variables from the **else** branch. For the resolution of pointers, we must also reason about Algorithm 78 (CondAssign), because the resolution of pointer data is more involved. By proving that this algorithm will correctly resolve the true locations for pointers, we will then have that the statements created by ResolveVariables will appropriately resolve all pointers.

□

3.2.1 Erasure Function

Here, we show the full erasure function in Figure 3.31. This function is intended to take a SMC^2 program or configuration and remove all private privacy labels, decrypt any private data, and clear any additional tracking features that are specific to SMC^2; this process will result in a Vanilla C program or configuration. This function precisely defines the expected correspondence between SMC^2 and Vanilla C, enabling us to reason about the correctness of SMC^2 with respect to the standard C semantics we show using Vanilla C.

Figure 3.31a shows erasure over an entire configuration, calling Erase on the four-tuple of the environment, memory, and two empty maps needed as the base for the Vanilla C environment and memory; removing the accumulator (i.e., replacing it with □); and calling Erase on the statement. Figure 3.31b shows erasure over types and type lists (i.e., for function types). Here, we remove any privacy labels given to the types, with unlabeled types being returned as is. For function types, we must iterate over the entire list of types as well as the return type. Figure 3.31c shows erasure over expression lists (i.e., from function calls) and parameter lists (i.e., from function definitions).

Figure 3.31d shows erasure over statements. For statements, we case over the various possible statements. When we reach a private value (i.e., $\mathrm{encrypt}(n)$), we decrypt and then return the decrypted value. For function pmalloc, we replace the function name with malloc, modifying the argument to appropriately evaluate the expected size of the type. For functions pfree, smcinput, and smcoutput, we simply replace the function name with its Vanilla C equivalent. All other cases recursively call the erasure function as needed, with the last case (_) handling all cases that are already identical to the Vanilla Cequivalent (i.e., NULL,

locations).

Figure 3.31e shows erasure over bytes stored in memory, which is used from within the erasure on the environment and memory. This function takes the byte-wise data representation, the type that it should be interpreted as, and the size expected for the data. For regular public types, we do not need to modify the byte-wise data. For regular private types (i.e., single values and array data), we get back the value(s) from the representation, decrypt, and obtain the byte-wise data for the decrypted value(s). For pointers with a single location, we must get back the pointer data structure, then simply remove the privacy label from the type stored there. For private pointers with multiple locations, we must declassify the pointer, retrieving it's true location and returning the byte-wise data for the pointer data structure with only that location. For functions, we get back the function data, then call Erase on the function body, remove the tag for whether the function has public side effects (i.e., replace with $\square$), and call Erase on the function parameter list.

Figure 3.32 shows erasure over the environment and memory. In order to properly handle all types of variables and data stored, we must iterate over both the SMC^2 environment and memory maps, and pass along the Vanilla C environment and memory maps as we remove elements from the SMC^2 maps and either add to them to the Vanilla C maps or discard them. The first case is the base case, when the SMC^2 environment and memory are both empty, and we return the Vanilla C environment and memory. Next, we have three cases which continue to iterate through the SMC^2 memory after the environment has been emptied. These cases are possible due to the fact that in SMC^2 we remove mappings from the environment once they are out of scope, but we never remove mappings from memory.

Then we have three cases to handle regular variables. The first adds mappings to the Vanilla C environment and memory without the privacy annotations on the types, and calls Erase on the byte-wise data stored at that location (the behavior of this is shown in Figure 3.31e and described later in this section). The other two remove temporary variables (an their corresponding data) inserted by an **`if else`** statement branching on private data. The cases for arrays, pointers, and functions behave similarly; however, when we have an array we handle the array pointer as well as the array data within those cases.

The following algorithms are used in the proof of correctness to help prove congruence between the SMC^2 semantics and the Vanilla C semantics - in particular, they assist us in proving the memories are equivalent in the presence of `pfree` in the SMC^2 semantics, which can free a location that was not the true location. These algorithms facilitate managing the locations that have been swapped and comparing the locations.

$\mathbf{Erase}((\gamma,\ \sigma,\ \mathbf{acc},\ s)) =$
$\quad (\mathbf{Erase}(\gamma,\ \sigma,\ [\,],\ [\,]),\ \square,\ \mathbf{Erase}(s))$

(a) Erasure function over configurations

$\mathbf{Erase}(ty) =$
$\mid\ a\ bty\ \ =>\ \ bty$
$\mid\ a\ bty\ *\ \ =>\ \ bty*$
$\mid\ \overline{ty} \rightarrow ty\ \ =>\ \ \mathbf{Erase}(\overline{ty}) \rightarrow \mathbf{Erase}(ty))$
$\mid\ _\ \ =>\ ty$

$\mathbf{Erase}(\overline{ty}) =$
$\mid\ [\,]\ \ =>\ \ [\,]$
$\mid\ ty :: \overline{ty}\ \ =>\ \ \mathbf{Erase}(ty) :: \mathbf{Erase}(\overline{ty})$

(b) Erasure function over types and type lists

$\mathbf{Erase}(\overline{e}) =$
$\mid\ \overline{e},\ e\ \ =>\ \ \mathbf{Erase}(\overline{e}),\ \mathbf{Erase}(e)$
$\mid\ e\ \ =>\ \ \mathbf{Erase}(e)$
$\mid\ \text{void}\ \ =>\ \ \mathbf{void}$

$\mathbf{Erase}(\overline{p}) =$
$\mid\ \overline{p},\ ty\ var\ \ =>\ \ \mathbf{Erase}(\overline{p}),\ \mathbf{Erase}(ty\ var)$
$\mid\ ty\ var\ \ =>\ \ \mathbf{Erase}(ty)\ \mathbf{Erase}(var)$
$\mid\ \text{void}\ \ =>\ \ \mathbf{void}$

(c) Erasure function over lists

$\mathbf{Erase}(s) =$
$\mid\ x[e]\ \ =>\ \ x[\mathbf{Erase}(e)]$
$\mid\ [v_0, ..., v_n]\ =>$
$\quad [\mathbf{Erase}(v_0),\ \mathbf{Erase}(...),\ \mathbf{Erase}(v_n)]$
$\mid\ \text{malloc}(e)\ \ =>\ \ \mathbf{malloc}(\mathbf{Erase}(e))$
$\mid\ \text{pmalloc}(e,\ ty)\ \ =>$
$\quad \mathbf{malloc}(\mathbf{sizeof}(\mathbf{Erase}(ty)) \cdot \mathbf{Erase}(e))$
$\mid\ \text{free}(e)\ \ =>\ \ \mathbf{free}(\mathbf{Erase}(e))$
$\mid\ \text{pfree}(e)\ \ =>\ \ \mathbf{free}(\mathbf{Erase}(e))$
$\mid\ \text{sizeof}(ty)\ \ =>\ \ \mathbf{sizeof}(\mathbf{Erase}(ty))$
$\mid\ \text{smcinput}(\overline{e})\ \ =>\ \ \mathbf{mcinput}(\mathbf{Erase}(\overline{e}))$
$\mid\ \text{smcoutput}(\overline{e})\ \ =>\ \ \mathbf{mcoutput}(\mathbf{Erase}(\overline{e}))$
$\mid\ x(\overline{e})\ \ =>\ \ x(\mathbf{Erase}(\overline{e}))$
$\mid\ e_1\ bop\ e_2\ \ =>\ \ \mathbf{Erase}(e_1)\ bop\ \mathbf{Erase}(e_2)$
$\mid\ uop\ x\ \ =>\ \ uop\ x$
$\mid\ (e)\ \ =>\ \ (\mathbf{Erase}(e))$
$\mid\ (ty)\ e\ \ =>\ \ \mathbf{Erase}(ty))\ \mathbf{Erase}(e)$
$\mid\ var = e\ \ =>\ \ \mathbf{Erase}(var) = \mathbf{Erase}(e)$
$\mid\ *x = e\ \ =>\ \ *x = \mathbf{Erase}(e)$
$\mid\ s_1;\ s_2\ \ =>\ \ \mathbf{Erase}(s_1);\ \mathbf{Erase}(s_2)$
$\mid\ \{s\}\ \ =>\ \ \{\mathbf{Erase}(s)\}$
$\mid\ ty\ var\ \ =>\ \ \mathbf{Erase}(ty)\ \mathbf{Erase}(var)$
$\mid\ ty\ var = e\ \ =>$
$\quad \mathbf{Erase}(ty)\ \mathbf{Erase}(var) = \mathbf{Erase}(e)$
$\mid\ ty\ x(\overline{p})\ \ =>\ \ \mathbf{Erase}(ty)\ x(\mathbf{Erase}(\overline{p}))$
$\mid\ ty\ x(\overline{p})\ \{s\}\ \ =>$
$\quad \mathbf{Erase}(ty\ x(\overline{p}))\ \{\mathbf{Erase}(s)\}$
$\mid\ \text{if}(e)\ s_1\ \text{else}\ s_2\ \ =>$
$\quad \mathbf{if}(\mathbf{Erase}(e))\ \mathbf{Erase}(s_1)\ \mathbf{else}\ \mathbf{Erase}(s_2)$
$\mid\ \text{while}\ (e)\ s\ \ =>\ \ \mathbf{while}\ (\mathbf{Erase}(e))\ \mathbf{Erase}(s)$
$\mid\ _\ \ =>\ s$

(d) Erasure function over statements

$\mathbf{Erase}(\omega,\ ty,\ \alpha) =$
$\mid\ (\omega,\ \text{public}\ bty,\ \alpha)\ =>\ \omega$
$\mid\ (\omega,\ \text{private}\ bty,\ 1)\ =>\ v_1 = \text{DecodeVal}(ty,\ 1,\ \omega);\ v_2 = \text{decrypt}(v_1);\ \omega_1 = \text{EncodeVal}(bty,\ v_2);\ \omega_1$
$\mid\ (\omega,\ \text{private}\ bty,\ \alpha)\ =>\ [v_1 = \text{DecodeVal}(ty,\ RT\alpha,\ \omega);$
$\quad [v'_1, ..., v'_\alpha] = [\text{decrypt}(v_1), \text{decrypt}(...), \text{decrypt}(v_\alpha)];\ \omega_1 = \text{EncodeVal}(bty,\ [v'_1,\ ...,\ v'_\alpha]);\ \omega_1$
$\mid\ (\omega,\ \text{public}\ bty\ *,\ 1)\ =>\ [1,\ [(l,\mu)],\ [1],\ i] = \text{DecodePtr}(\text{public}\ bty\ *,\ 1,\ \omega);$
$\quad \omega_1 = \text{EncodePtr}(bty\ *,\ [1,\ [(l,\mu)],\ [1],\ \text{Erase}(ty'),\ i]);\ \omega_1$
$\mid\ (\omega,\ \text{private}\ bty\ *,\ 1)\ =>\ [1,\ [(l,\mu)],\ [1],\ i] = \text{DecodePtr}(\text{private}\ bty\ *,\ 1,\ \omega);$
$\quad \text{if}\ (i = 1)\ \text{then}\ \{ty_1 = \text{public}\ bty;\ ty_2 = \text{private}\ bty\}\ \text{else}\ \{ty_1 = \text{public}\ bty*;\ ty_2 = \text{private}\ bty*\};$
$\quad \mu_1 = \frac{\mu \cdot \tau(ty_1)}{\tau(ty_2)};\ \omega_1 = \text{EncodePtr}(bty\ *,\ [1,\ [(l,\mu_1)],\ [1],\ \text{Erase}(ty'),\ i]);\ \omega_1$
$\mid\ (\omega,\ \text{private}\ bty\ *,\ \alpha)\ =>\ [\alpha,\ \bar{l},\ \bar{j},\ i] = \text{DecodePtr}(\text{private}\ bty\ *,\ \alpha,\ \omega);$
$\quad (l,\mu) = \text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ i],\ \text{private}\ bty*);$
$\quad \text{if}\ (i = 1)\ \text{then}\ \{ty_1 = \text{public}\ bty;\ ty_2 = \text{private}\ bty\}\ \text{else}\ \{ty_1 = \text{public}\ bty*;\ ty_2 = \text{private}\ bty*\};$
$\quad \mu_1 = \frac{\mu \cdot \tau(ty_1)}{\tau(ty_2)};\ \omega_1 = \text{EncodePtr}(bty\ *,\ [1,\ [(l,\mu_1)],\ [1],\ i]);\ \omega_1$
$\mid\ (\omega,\ \overline{ty} \rightarrow ty,\ 1)\ =>\ (s,\ n,\ \overline{p}) = \text{DecodeFun}(\omega);\ \omega_1 = \text{EncodeFun}(\text{Erase}(s),\ \square,\ \text{Erase}(\overline{p}));\ \omega_1$

(e) Erasure function over bytes

Figure 3.31: The Erasure function, broken down into various functionalities.

$\mathbf{Erase}(\boldsymbol{\gamma},\ \boldsymbol{\sigma}, \widehat{\gamma}, \widehat{\boldsymbol{\sigma}}) =$
match (γ, σ) with
| $([\,], [\,])$ => $(\widehat{\gamma},\ \widehat{\boldsymbol{\sigma}})$
| $([\,], \sigma_1[l \to (\text{NULL},\ \text{void}*,\ \alpha,\ \text{PermL}(\text{Freeable},\ \text{void}*,\ \text{public},\ \alpha))])$
=> $(\mathbf{Erase}([\,],\ \boldsymbol{\sigma_1},\ \widehat{\gamma},\ \widehat{\boldsymbol{\sigma}}[\boldsymbol{l} \to (\mathbf{NULL},\ \mathbf{void}*,\ \widehat{\boldsymbol{\alpha}},\ \mathbf{PermL}(\boldsymbol{perm},\ \mathbf{void}*,\ \mathbf{public},\ \widehat{\boldsymbol{\alpha}}))]))$
| $([\,], \sigma_1[l \to (\text{NULL},\ \text{void}*,\ \alpha,\ \text{PermL}(\text{Freeable},\ ty,\ \text{private},\ \alpha))])$
=> $\widehat{\alpha} = \left(\frac{\alpha}{\tau(ty)}\right) \cdot \tau(\text{Erase}(ty))$
$(\mathbf{Erase}([\,],\ \boldsymbol{\sigma_1},\ \widehat{\gamma},\ \widehat{\boldsymbol{\sigma}}[\boldsymbol{l} \to (\mathbf{NULL},\ \mathbf{void}*,\ \widehat{\boldsymbol{\alpha}},\ \mathbf{PermL}(\boldsymbol{perm},\ \mathbf{void}*,\ \mathbf{public},\ \widehat{\boldsymbol{\alpha}}))]))$
| $([\,], \sigma_1[l \to (\omega,\ ty,\ \alpha,\ \text{PermL}(perm,\ ty,\ a,\ \alpha))])$
=> $(\mathbf{Erase}([\,], \boldsymbol{\sigma_1}, \widehat{\gamma}, \widehat{\boldsymbol{\sigma}}[\boldsymbol{l} \to (\mathbf{Erase}(\boldsymbol{\omega}, \boldsymbol{ty}, \boldsymbol{\alpha}), \mathbf{Erase}(\boldsymbol{ty}), \boldsymbol{\alpha},\ \mathbf{PermL}(\boldsymbol{perm}, \mathbf{Erase}(\boldsymbol{ty}), \mathbf{public}, \boldsymbol{\alpha}))]))$
| $([\,], \sigma_1[l \to (\omega,\ ty,\ \alpha,\ \text{PermL}(perm,\ ty,\ a,\ \alpha))])$
=> $(\mathbf{Erase}([\,],\ \boldsymbol{\sigma_1},\ \widehat{\gamma}, \widehat{\boldsymbol{\sigma}}[\boldsymbol{l} \to (\mathbf{Erase}(\boldsymbol{\omega}, \boldsymbol{ty}, \boldsymbol{\alpha}), \mathbf{Erase}(\boldsymbol{ty}), \boldsymbol{\alpha}, \mathbf{PermL}(\boldsymbol{perm}, \mathbf{Erase}(\boldsymbol{ty}), \mathbf{public}, \boldsymbol{\alpha}))]))$
| $([\,], \sigma_1[l \to (\omega,\ ty,\ 1,\ \text{PermL_Fun}(\text{public}))])$
=> $(\mathbf{Erase}([\,],\ \boldsymbol{\sigma_1},\ \widehat{\gamma},\ \widehat{\boldsymbol{\sigma}}[\boldsymbol{l} \to (\mathbf{Erase}(\boldsymbol{\omega}, \boldsymbol{ty}, \mathbf{1}),\ \mathbf{Erase}(\boldsymbol{ty}),\ \mathbf{1},\ \mathbf{PermL_Fun}(\mathbf{public}))]))$
| $(\gamma_1[x \to (l,\ a\ bty)],\ \sigma_1[l \to (\omega,\ a\ bty,\ 1,\ \text{PermL}(perm,\ a\ bty,\ a,\ 1))])$
=> $(\mathbf{Erase}(\boldsymbol{\gamma_1}, \boldsymbol{\sigma_1}, \widehat{\gamma}[\boldsymbol{x} \to (\boldsymbol{l}, \boldsymbol{bty})],\ \widehat{\boldsymbol{\sigma}}[\boldsymbol{l} \to (\mathbf{Erase}(\boldsymbol{\omega}, \boldsymbol{a\ bty}, \mathbf{1}), \boldsymbol{bty}, \mathbf{1}, \mathbf{PermL}(\boldsymbol{perm}, \boldsymbol{bty}, \mathbf{public}, \mathbf{1}))]))$
| $(\gamma_1[res_n \to (l, \text{private}\ bty)], \sigma_1[l \to (\omega, \text{private}\ bty, 1, \text{PermL}(perm, \text{private}\ bty, \text{private}, 1))])$
=> $(\mathbf{Erase}(\boldsymbol{\gamma_1},\ \boldsymbol{\sigma_1},\ \widehat{\gamma},\ \widehat{\boldsymbol{\sigma}}))$
| $(\gamma_1[x_then_n \to (l,\ a\ bty)],\ \sigma_1[l \to (\omega,\ a\ bty,\ 1, \text{PermL}(perm,\ a\ bty,\ a,\ 1))])$ => $(\mathbf{Erase}(\boldsymbol{\gamma_1},\ \boldsymbol{\sigma_1},\ \widehat{\gamma},\ \widehat{\boldsymbol{\sigma}}))$
| $(\gamma_1[x_else_n \to (l,\ a\ bty)],\ \sigma_1[l \to (\omega,\ a\ bty,\ 1, \text{PermL}(perm,\ a\ bty,\ a,\ 1))])$ => $(\mathbf{Erase}(\boldsymbol{\gamma_1},\ \boldsymbol{\sigma_1},\ \widehat{\gamma},\ \widehat{\boldsymbol{\sigma}}))$
| $(\gamma_1[x \to (l,\ a\ \text{const}\ bty*)],\ \sigma_1[l \to (\omega,\ a\ \text{const}\ bty*,\ 1, \text{PermL}(perm,\ a\ \text{const}\ bty*,\ a,\ 1))])$
=> $\text{DecodePtr}(a\ \text{const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$;
$\sigma_1 = \sigma_2[l_1 \to (\omega_1,\ a\ bty,\ \alpha,\ \text{PermL}(perm,\ a\ bty,\ a,\ \alpha))]$;
$(\mathbf{Erase}(\boldsymbol{\gamma_1},\ \boldsymbol{\sigma_2},\ \widehat{\gamma}[\boldsymbol{x} \to (\boldsymbol{l},\ \mathbf{Erase}(\boldsymbol{a}\ \mathbf{const}\ \boldsymbol{bty*}))],$
$\widehat{\boldsymbol{\sigma}}[\boldsymbol{l} \to (\mathbf{Erase}(\boldsymbol{\omega}, \boldsymbol{a}\ \mathbf{const}\ \boldsymbol{bty*}, \mathbf{1}), \mathbf{const}\ \boldsymbol{bty*}), \mathbf{1}, \mathbf{PermL}(\boldsymbol{perm},\ \mathbf{const}\ \boldsymbol{bty*},\ \mathbf{public},\ \mathbf{1}))]$
$[\boldsymbol{l_1} \to (\mathbf{Erase}(\boldsymbol{\omega_1}, \boldsymbol{a\ bty}, \boldsymbol{\alpha}), \boldsymbol{bty}, \boldsymbol{\alpha}, \mathbf{PermL}(\boldsymbol{perm},\ \boldsymbol{bty},\ \mathbf{public},\ \boldsymbol{\alpha}))]))$
| $(\gamma_1[x_then_n \to (l,\ a\ \text{const}\ bty*)],\ \sigma_1[l \to (\omega,\ a\ \text{const}\ bty*,\ 1, \text{PermL}(perm,\ a\ \text{const}\ bty*,\ a,\ 1))])$
=> $\text{DecodePtr}(a\ \text{const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$;
$\sigma_1 = \sigma_2[l_1 \to (\omega_1,\ a\ bty,\ \alpha,\ \text{PermL}(perm,\ a\ bty,\ a,\ \alpha))]$; $(\mathbf{Erase}(\boldsymbol{\gamma_1},\ \boldsymbol{\sigma_2},\ \widehat{\gamma},\ \widehat{\boldsymbol{\sigma}}))$
| $(\gamma_1[x_else_n \to (l,\ a\ \text{const}\ bty*)],\ \sigma_1[l \to (\omega,\ a\ \text{const}\ bty*,\ 1, \text{PermL}(perm,\ a\ \text{const}\ bty*,\ a,\ 1))])$
=> $\text{DecodePtr}(a\ \text{const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$;
$\sigma_1 = \sigma_2[l_1 \to (\omega_1,\ a\ bty,\ \alpha,\ \text{PermL}(perm,\ a\ bty,\ a,\ \alpha))]$; $(\mathbf{Erase}(\boldsymbol{\gamma_1},\ \boldsymbol{\sigma_2},\ \widehat{\gamma},\ \widehat{\boldsymbol{\sigma}}))$
| $(\gamma_1[x \to (l,\ a\ bty*)],\ \sigma_1[l \to (\omega,\ a\ bty*,\ \alpha, \text{PermL}(perm,\ a\ bty*,\ a,\ \alpha))])$
=> $(\mathbf{Erase}(\boldsymbol{\gamma_1},\ \boldsymbol{\sigma_1},\ \widehat{\gamma}[\boldsymbol{x} \to (\boldsymbol{l},\ \mathbf{Erase}(\boldsymbol{a}\ \boldsymbol{bty*}))],$
$\widehat{\boldsymbol{\sigma}}[\boldsymbol{l} \to (\mathbf{Erase}(\boldsymbol{\omega}, \boldsymbol{ty}, \boldsymbol{n}),\ \mathbf{Erase}(\boldsymbol{ty}),\ \boldsymbol{\alpha},\ \mathbf{PermL}(\boldsymbol{perm},\ \mathbf{Erase}(\boldsymbol{ty}),\ \mathbf{public},\ \boldsymbol{\alpha}))]))$
| $(\gamma_1[x_then_n \to (l,\ a\ bty*)],\ \sigma_1[l \to (\omega,\ a\ bty*,\ \alpha, \text{PermL}(perm,\ a\ bty*,\ a,\ \alpha))])$
=> $(\mathbf{Erase}(\boldsymbol{\gamma_1},\ \boldsymbol{\sigma_1},\ \widehat{\gamma},\ \widehat{\boldsymbol{\sigma}}))$
| $(\gamma_1[x_else_n \to (l,\ a\ bty*)],\ \sigma_1[l \to (\omega,\ a\ bty*,\ \alpha, \text{PermL}(perm,\ a\ bty*,\ a,\ \alpha))])$
=> $(\mathbf{Erase}(\boldsymbol{\gamma_1},\ \boldsymbol{\sigma_1},\ \widehat{\gamma},\ \widehat{\boldsymbol{\sigma}}))$
| $(\gamma_1[temp_{ctr}_n \to (l,\ \text{private}\ bty*)],\ \sigma_1[l \to (\omega,\ \text{private}\ bty*,\ \alpha, \text{PermL}(perm,\ \text{private}\ bty*,\ \text{private},\ \alpha))])$
=> $(\mathbf{Erase}(\boldsymbol{\gamma_1},\ \boldsymbol{\sigma_1},\ \widehat{\gamma},\ \widehat{\boldsymbol{\sigma}}))$
| $(\gamma_1[x \to (l,\ \overline{ty} \to ty)], \sigma_1[l \to (\omega,\ \overline{ty} \to ty,\ 1,\ \text{PermL_Fun}(\text{public}))]$
=> $(\mathbf{Erase}(\boldsymbol{\gamma_1},\ \boldsymbol{\sigma_1},\ \widehat{\gamma}[\boldsymbol{x} \to (\boldsymbol{l},\ \mathbf{Erase}(\overline{\boldsymbol{ty}} \to \boldsymbol{ty}))],$
$\widehat{\boldsymbol{\sigma}}[\boldsymbol{l} \to (\mathbf{Erase}(\boldsymbol{\omega}, \overline{\boldsymbol{ty}} \to \boldsymbol{ty}, \mathbf{1}),\ \mathbf{Erase}(\overline{\boldsymbol{ty}} \to \boldsymbol{ty}),\ \mathbf{1},\ \mathbf{PermL_Fun}(\mathbf{public}))]))$

Figure 3.32: Erasure function over the environment and memory

Algorithm 79 $\bar{l}_1 \leftarrow \mathrm{GetLocationSwap}(\bar{l}, \bar{j})$

1: $\bar{l}_1 = [\,]$
2: **for all** $m \in \{0, ..., |\bar{j}| - 1\}$ **do**
3: **if** $\bar{j}[m] =_{\mathrm{private}} 1$ **then**
4: $\bar{l}_1.\mathrm{append}(\bar{l}[m])$
5: **end if**
6: **end for**
7: **return** $\bar{l}_1$

Algorithm 79, (GetLocationSwap), is used to analyze the location and tag lists returned by Algorithm PFree to show which location has been swapped, if any were swapped.

Algorithm 80 $\sigma_2 \leftarrow \mathrm{SwapMemory}(\sigma, \psi)$

1: **for all** $\bar{l} \in \psi$ **do**
2: **if** $(\bar{l} = [(l_1, 0), (l_2, 0)])$ **then**
3: $\sigma_1[l_1 \rightarrow (\omega_1, ty_1, n_1, \mathrm{PermL}(perm_1, ty_1, a_1, n_1))][l_2 \rightarrow (\omega_2, ty_2, n_2, \mathrm{PermL}(perm_2, ty_2, a_2, n_2))] = \sigma$
4: $\sigma_2 = \sigma_1[l_1 \rightarrow (\omega_2, ty_2, n_2, \mathrm{PermL}(perm_2, ty_2, a_2, n_2))][l_2 \rightarrow (\omega_1, ty_1, n_1, \mathrm{PermL}(perm_1, ty_1, a_1, n_1))]$
5: **end if**
6: $\sigma = \sigma_2$
7: **end for**
8: **return** σ_2

Algorithm 80, (SwapMemory), is used to swap two locations in memory, to get back the original memory before swaps occurred in order to easily compare the congruence of the Basic SMC^2 memory and the Vanilla C memory.

Algorithm 81 (GetFinalSwap) iterates over map ψ to get a map ψ_1 that has the final listing of congruent locations between SMC^2 memory and Vanilla C memory, particularly in the case where locations may have been swapped more than once. For example, if only l_1 and l_2 are swapped, then we have $\psi_1 = [(l_1, 0), (l_2, 0)][(l_2, 0), (l_1, 0)]$ (i.e., meaning that $l_1 \cong_{\psi_1} \widehat{l_2}$ and $l_2 \cong_{\psi_1} \widehat{l_1}$). If we first swap l_1 and l_2 and then swap l_2 and l_3, we have $\psi_1 = [(l_1, 0), (l_2, 0)][(l_2, 0), (l_3, 0)][(l_3, 0), (l_1, 0)]$ (i.e., meaning that $l_1 \cong_{\psi_1} \widehat{l_2}$, $l_2 \cong_{\psi_1} \widehat{l_3}$, and $l_3 \cong_{\psi_1} \widehat{l_1}$). This behavior extends for any additional location mappings within ψ.

Algorithm 82, (CheckIDCongruence), takes a Basic SMC^2 memory block identifier and a Vanilla C memory block identifier and checks whether they are congruent.

Algorithm 83 (CheckCodeCongruence) shows how we compare evaluation codes to ensure that two evaluation traces are corresponding between Basic SMC^2 and Vanilla C. In particular, it ensures that the evaluation trace for `pmalloc` in the Basic SMC^2 evaluation code trace has the corresponding Vanilla C evaluation code trace.

Algorithm 81 $\psi_1 \leftarrow \text{GetFinalSwap}(\psi)$

```
ψ_1 = [ ]
for all l̄ ∈ ψ do
    if (l̄ = [(l_1, 0), (l_2, 0)]) then
        if ([(l_1, 0), (l_m, 0)] ∉ ψ_1) then
            if ([(l_2, 0), (l_n, 0)] ∉ ψ_1) then
                ψ_1 = ψ_1[(l_1, 0), (l_2, 0)][(l_2, 0), (l_1, 0)]
            else
                ψ_2[(l_2, 0), (l_n, 0)] = ψ_1
                ψ_3 = ψ_2[(l_1, 0), (l_n, 0)][(l_2, 0), (l_1, 0)]
                ψ_1 = ψ_3
            end if
        else
            if ([(l_2, 0), (l_n, 0)] ∉ ψ_1) then
                ψ_2[(l_1, 0), (l_m, 0)] = ψ_1
                ψ_3 = ψ_2[(l_1, 0), (l_2, 0)][(l_2, 0), (l_m, 0)]
                ψ_1 = ψ_3
            else
                ψ_2[(l_1, 0), (l_m, 0)][(l_2, 0), (l_n, 0)] = ψ_1
                ψ_3 = ψ_2[(l_1, 0), (l_n, 0)][(l_2, 0), (l_m, 0)]
                ψ_1 = ψ_3
            end if
        end if
    end if
end for
return ψ_1
```

Algorithm 82 $j \leftarrow \text{CheckIDCongruence}(\psi,\ l_1,\ \widehat{l})$

```
l_2 = l̂
ψ_1 = GetFinalSwap(ψ)
if ([(l_1, 0), (l_2, 0)] ∈ ψ_1) then
    return 1
else if (([(l_1, 0), (l_m, 0)] ∈ ψ_1) ∧ (l_m ≠ l_2)) then
    return 0
else if (([(l_n, 0), (l_2, 0)] ∈ ψ_1) ∧ (l_n ≠ l_1)) then
    return 0
else if (l_1 = l_2) then
    return 1
else
    return 0
end if
```

Algorithm 83 $j \leftarrow \text{CheckCodeCongruence}(\overline{d}, \widehat{\overline{d}})$

```
if (|d̄| = 0) ∧ (|d̂̄| = 0) then
    return 1
else if (|d̄| = 1) ∧ (|d̂̄| = 1) then
    [d] = d̄
    [d̂] = d̂̄
    if d ≅ d̂ then
        return 1
    else
        return 0
    end if
else
    [d_0, ..., d_n] = d̄
    [d̂_0, ..., d̂_m] = d̂̄
    if d_0 = malp then
        if (d̂_0 = mal) ∧ (d̂_1 = bm) ∧ (d̂_2 = ty) then
            return CheckCodeCongruence([d_1, ..., d_n], [d̂_3, ..., d̂_m])
        else
            return 0
        end if
    else
        if d_0 ≅ d̂_0 then
            return CheckCodeCongruence([d_1, ..., d_n], [d̂_1, ..., d̂_m])
        else
            return 0
        end if
    end if
end if
```

3.2.2 Definitions

Definition 3.2.1. A memory location (l, μ), $(\widehat{l}, \widehat{\mu})$ is *aligned* if and only if the location refers to either the beginning of a memory block ($\mu = \widehat{\mu} = 0$) or the beginning of an element inside an array.

Definition 3.2.2. An overshooting memory access by an array is *well-aligned* if and only if:

- the initial memory location is *aligned* and of the expected type,
- the ending memory location is *aligned* and of the expected type, and
- all memory blocks or elements iterated over are of the expected type.

Definition 3.2.3. A SMC^2 alignment indicator and a Vanilla C alignment indicator are *congruent*, in symbols $j \cong \widehat{j}$, if and only if either $j = 1$ and $\widehat{j} = 1$ or $j = 0$ and $(\widehat{j} = 0) \vee (\widehat{j} = 1)$.

Definition 3.2.4. A location list is *aligned* if and only if for all locations (l_i, μ_i) in the list:

- all memory block identifiers l_i are of the expected type,
- all memory block identifiers l_i are of the same size, and
- all offsets μ_i are equal.

Definition 3.2.5. An overshooting memory access by a pointer is *well-aligned* if and only if:

- the initial location list $\overline{l}_i$ is *aligned*,
- the final location list $\overline{l}_f$ is *aligned*, and
- for each location in the initial location list, all memory blocks or elements iterated over to get to the corresponding location in the final location list are of the expected type.

Definition 3.2.6. A SMC^2 type and a Vanilla C type are *congruent*, in symbols $ty \cong \hat{ty}$, if and only if $\text{Erase}(ty) = \hat{ty}$.

Definition 3.2.7. A SMC^2 type list and a Vanilla C type list are *congruent*, in symbols $\overline{ty} \cong \hat{\overline{ty}}$, if and only if $\text{Erase}(\overline{ty}) = \hat{\overline{ty}}$.

Definition 3.2.8. A SMC^2 expression list and a Vanilla C expression list are *congruent*, in symbols $\overline{e} \cong \hat{\overline{e}}$, if and only if $\text{Erase}(\overline{e}) = \hat{\overline{e}}$.

Definition 3.2.9. A SMC^2 parameter list and a Vanilla C parameter list are *congruent*, in symbols $\overline{p} \cong \hat{\overline{p}}$, if and only if $\text{Erase}(\overline{p}) = \hat{\overline{p}}$.

Definition 3.2.10. A SMC^2 statement and a Vanilla C statement are *congruent*, in symbols $s \cong \hat{s}$, if and only if $\text{Erase}(s) = \hat{s}$.

Definition 3.2.11. $\psi = [\,] \mid \psi[\bar{l}]$

A map ψ is defined as a list of lists of locations that is formed by tracking which locations are privately switched during the execution of the statement $\text{pfree}(x)$ in a SMC^2 program s to enable comparison with the *congruent* Vanilla C program $\widehat{s}$.

Definition 3.2.12. A SMC^2 memory block identifier and a Vanilla C memory block identifier are ψ-*congruent*, in symbols $l \cong_\psi \widehat{l}$, given map ψ, if and only if $\text{CheckIDCongruence}(\psi, l, \widehat{l}) = 1$.

Definition 3.2.13. A SMC^2 location and a Vanilla C location are ψ-*congruent*, in symbols $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$, given SMC^2 type ty correlating to (l, μ) and Vanilla C type $\widehat{ty}$ correlating to $(\widehat{l}, \widehat{\mu})$, if and only if $ty \cong \widehat{ty}$, $l \cong_\psi \widehat{l}$, and either ty is a public type and $\mu = \widehat{\mu}$, or ty is a private type and $(\mu) \cdot \left(\frac{\tau(\widehat{ty})}{\tau(ty)}\right) = \widehat{\mu}$.

Definition 3.2.14. A SMC^2 pointer data structure for a pointer of type $ty \in \{a \text{ const } bty*, a\ bty*\}$ and a Vanilla C pointer data structure for a pointer of type $\widehat{ty} \in \{\text{const } \widehat{bty}*, \widehat{bty}*\}$ are ψ-*congruent*, in symbols $[\alpha, \bar{l}, \bar{j}, i] \cong_\psi [1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}]$, given map ψ, if $ty \cong \widehat{ty}$, $i = \widehat{i}$ and either $a = \text{public}$ and $\alpha = 1$, $\bar{l} = (l, \mu)$ such that $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$, and $\bar{j} = [1]$ or $a = \text{private}$ and $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ i],\ \text{private } bty*) = (l, \mu)$ such that $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$.

Definition 3.2.15. A SMC^2 environment and memory pair and a Vanilla C environment and memory pair are ψ-*congruent*, in symbols $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, if and only if $\text{Erase}(\gamma, \sigma, [\,], [\,]) = (\widehat{\gamma}, \widehat{\sigma}')$, and $\text{SwapMemory}(\widehat{\sigma}', \psi) = \widehat{\sigma}$.

Definition 3.2.16. A SMC^2 byte-wise representation ω of a given type ty and size n and a Vanilla C byte-wise representation $\widehat{\omega}$ are ψ-*congruent*, in symbols $\omega \cong_\psi \hat{\omega}$, if and only if either $ty \neq \text{private } bty*$ and $\text{Erase}(\omega, ty, n) = \hat{\omega}$ or $ty = \text{private } bty*$ and $\text{Erase}(\omega, ty, n) = \hat{\omega}_1$ such that the pointer data structure stored in ω and the pointer data structure stored in $\hat{\omega}$ are ψ-*congruent* by Definition 3.2.14.

Definition 3.2.17. A SMC^2 value and Vanilla C value are ψ-*congruent*, in symbols $v \cong_\psi \widehat{v}$, if and only if either $v \neq (l, \mu)$, $\widehat{v} \neq (\widehat{l}, \widehat{\mu})$, and $v \cong \widehat{v}$, or $v = (l, \mu)$, $\widehat{v} = (\widehat{l}, \widehat{\mu})$, and $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$.

Definition 3.2.18. A SMC^2 statement and Vanilla C statement are ψ-*congruent*, in symbols $s \cong_\psi \widehat{s}$, if and only if for all $v_i \in s$, $\widehat{v}_i \in \widehat{s}$, $v_i \cong_\psi \widehat{v}_i$ and otherwise $s \cong \widehat{s}$.

Definition 3.2.19. A SMC^2 expression list and a Vanilla C expression list are ψ-*congruent*, in symbols $\bar{e} \cong_\psi \hat{\bar{e}}$, given a map ψ, if and only if $\forall e \neq (l, \mu) \in \bar{e}$, $\text{Erase}(e) = \widehat{e}$ and $\forall e = (l, \mu) \in \bar{e}$, $e \cong_\psi \widehat{e}$ by Definition 3.2.10.

Definition 3.2.20. A SMC^2 configuration and a Vanilla C configuration are ψ-*congruent*, in symbols $(\gamma,\ \sigma,\ \text{acc},\ s) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s})$, if and only if $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $s \cong_\psi \widehat{s}$.

Definition 3.2.21. A SMC^2 evaluation code and a Vanilla C evaluation code are *congruent*, in symbols $d \cong \widehat{d}$, if and only if $(\gamma, \sigma, \text{acc}, s) \Downarrow_d (\gamma_1, \sigma_1, \text{acc}, v)$ and $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s}) \Downarrow_{\widehat{d}} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v})$ such that $(\gamma, \sigma, \text{acc}, s) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$ and $(\gamma_1, \sigma_1, \text{acc}, v) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v})$ by Definition 3.2.20.

Definition 3.2.22. The SMC^2 evaluation code trace $([malp, d_1])$ for the statement $\text{pmalloc}(e, ty)$ and the Vanilla C evaluation code trace $([\widehat{mal}, \widehat{bm}, \widehat{ty}, \widehat{d}_1])$ for the congruent statement $\text{malloc}(\text{sizeof}(\widehat{ty}) \cdot \widehat{e})$ are *congruent* if and only if $d_1 \cong \widehat{d}_1$.

Definition 3.2.23. A SMC^2 evaluation code trace and a Vanilla C evaluation code trace are *congruent*, in symbols $[d_1, ..., d_n] \cong [\widehat{d}_1, ..., \widehat{d}_m]$, if and only if $\text{CheckCodeCongruence}([d_1, ..., d_n], [\widehat{d}_1, ..., \widehat{d}_m]) = 1$ by Algorithm 83.

Definition 3.2.24. A SMC^2 derivation $\Pi \triangleright (\gamma, \sigma, \text{acc}, s) \Downarrow_{[d_1,...,d_n]} (\gamma_1, \sigma_1, \text{acc}, v)$ and a Vanilla C derivation $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s}) \Downarrow'_{[\widehat{d}_1,...,\widehat{d}_m]} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v})$ are ψ-*congruent*, in symbols $\Pi \cong'_\psi \Sigma$, if and only if given initial map ψ and ψ' derived from evaluating Π, $(\gamma, \sigma, \text{acc}, s) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$, $[d_1, ..., d_n] \cong [\widehat{d}_1, ..., \widehat{d}_m]$, $(\gamma_1, \sigma_1, \text{acc}, v) \cong'_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v})$.

Definition 3.2.25. Two input files are *congruent*, in symbols $inp \cong \widehat{inp}$, if and only if for all mappings of variables to values $x = v \in inp$ and $\widehat{x} = \widehat{v} \in \widehat{inp}$, $x = \widehat{x}$ and $v \cong \widehat{v}$ by Definition 3.2.10.

Definition 3.2.26. Two output files are *congruent*, in symbols $out \cong \widehat{out}$, if and only if for all mappings of variables to values $x = v \in out$ and $\widehat{x} = \widehat{v} \in \widehat{out}$, $x = \widehat{x}$ and $v \cong \widehat{v}$ by Definition 3.2.10.

3.2.3 Lemmas

Axiom 3.2.1. *Given an array $[v_0, ..., v_{n-1}]$ and a private index i, we assume that the private index is within the bound of the array, in symbols $0 \leq i \leq n - 1$.*

Axiom 3.2.2. *For every execution of an `if else` statement branching on private data at any level of nesting, the temporary variables injected by SMC^2 will be unique when compared to all variables in the current environment, and these variables can only be accessed and modified through the helper algorithms for the `if else` statement branching on private data.*

Axiom 3.2.3. *Given a SMC^2 program of statement s and a ψ-congruent Vanilla C program of statement $\widehat{s}$, in symbols $s \cong_\psi \widehat{s}$, any time a new memory block identifier is obtained from the available pool in the SMC^2 program such that $l = \phi()$, an identical memory block identifier is obtained from the available pool in the Vanilla C program such that $\widehat{l} = \phi()$ and $l = \widehat{l}$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$.*

Axiom 3.2.4. *Given a SMC^2 configuration and the congruent Vanilla C configuration, we assume that all input files used during execution using these configurations are* congruent.

Axiom 3.2.5. *Given a SMC^2 private pointer data structure $[\alpha, \bar{l}, \bar{j}, i]$ stored at memory block l and ψ-*congruent *Vanilla C pointer data structure $[1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$ stored at ψ-*congruent *memory block $\widehat{l}$, we consider $l, \widehat{l}$ to be* equally freeable *if either:*

- *both* $\text{CheckFreeable}(\gamma, \bar{l}, \bar{j}) = 1$ *and* $\text{CheckFreeable}(\widehat{\gamma}, [(\widehat{l}_1, \widehat{\mu}_1)], [1]) = 1$, *or*

- *both* $\text{CheckFreeable}(\gamma,\ \bar{l},\ \bar{j}) = 0$ *and* $\text{CheckFreeable}(\widehat{\gamma},\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1]) = 0$.

Lemma 3.2.1. *Given an initial map ψ, environment γ, memory σ, accumulator acc, and expression e, if $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, v)$ such that $v \neq \text{skip}$, then $\text{pfree}(e_1) \notin e$ and the ending map ψ_1 is equivalent to ψ.*

Proof. By definition of SMC^2 rule pfree, skip is returned from the evaluation of $\text{pfree}(e_1)$. Therefore, by case analysis of the rules, if $v \neq \text{skip}$, then $\text{pfree}(e_1) \notin e$. By Definition 3.2.11, ψ is only modified after the execution of function pfree; therefore we have that $\psi_1 = \psi$. □

Lemma 3.2.2. *Given configuration $(\gamma, \sigma, \text{acc}, s)$, if $(\gamma, \sigma, \text{acc}, s) \Downarrow_{d_1} (\gamma_1, \sigma_1, \text{acc}, v)$, then $(l, \mu) \notin s$.*

Proof. Proof by contradiction using all semantic rules. □

Lemma 3.2.3. *Given map ψ and statement $s, \widehat{s}$, if $s \cong_\psi \widehat{s}$ and $(l, \mu) \notin s$, then $s \cong \widehat{s}$.*

Proof. Proof by case analysis over possible SMC^2 statements, using Definition 3.2.10. □

Lemma 3.2.4. *Given map ψ and configuration $(\gamma, \sigma, \text{acc}, s)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s})$ such that $(\gamma, \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $s \cong \widehat{s}$, if $(l, \mu) \notin s$, then $s \cong_\psi \widehat{s}$ and $(\gamma, \sigma, \text{acc}, s) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s})$.*

Proof. By Definition 3.2.10, if $s \cong \widehat{s}$ then $\text{Erase}(s) = \widehat{s}$. Given $s \cong \widehat{s}$ and $(l, \mu) \notin s$, we have for all $v_i \in s$, $\widehat{v}_i \in \widehat{s}$, $v_i \cong \widehat{v}_i$, and therefore by Definition 3.2.17 we have $v_i \in s$, $\widehat{v}_i \in \widehat{s}$, $v_i \cong_\psi \widehat{v}_i$. Therefore, by Definition 3.2.18 we have $s \cong_\psi \widehat{s}$ and by Definition 3.2.20 we have $(\gamma, \sigma, \text{acc}, s) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s})$. □

Lemma 3.2.5. *Given map ψ, number $n_1, n_2, \widehat{n}_1, \widehat{n}_2$, and bop $\in \{<, <=, >=, >, ==, !=\}$, if $n_1 \cong_\psi \widehat{n}_1$, $n_2 \cong_\psi \widehat{n}_2$, and n_1 bop n_2, then $\widehat{n}_1$ bop $\widehat{n}_2$.*

Proof. By Definition 3.2.17 we have $n_1 \cong \widehat{n}_1$ and $n_2 \cong \widehat{n}_2$. Given n_1 *bop* n_2 such that *bop* $\in \{<, <=, >=, >, ==, !=\}$, by definition of Erase and decrypt we will also have $\widehat{n}_1$ *bop* $\widehat{n}_2$. □

Lemma 3.2.6. *Given SMC^2 statement* $\text{pmalloc}(e, ty)$ *and Vanilla C statement* $\text{malloc}(\widehat{e})$, *if* $\widehat{e} = \text{sizeof}(\text{Erase}(ty)) \cdot \text{Erase}(e)$ *then* $\text{pmalloc}(e, ty) \cong \text{malloc}(\widehat{e})$.

Proof. By definition of Erase. □

Lemma 3.2.7. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, memory block identifier $l, \widehat{l}$, and size $n, \widehat{n}$, if $\sigma_1 = \sigma\big[l \rightarrow \big(\text{NULL}, \text{void}*, n, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, n)\big]$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, $l = \widehat{l}$, and $n = \widehat{n}$ then $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\text{NULL}, \text{void}*, \widehat{n}, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, \widehat{n}))]$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$.*

Proof. By definition of the Erase function. □

Lemma 3.2.8. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, memory block identifier $l, \widehat{l}$, type ty, and size $n, \widehat{n}$, if $\sigma_1 = \sigma\big[l \to \big(\text{NULL},\ \text{void}*,\ n,\ \text{PermL}(\text{Freeable}, ty, \text{private}, n)\big]$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, $l = \widehat{l}$, $ty \cong \widehat{ty}$, and $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$, then $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \to (\text{NULL}, \text{void}*, \widehat{n}, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, \widehat{n}))]$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$.*

Lemma 3.2.9. *Given map ψ, array $[v_0, ..., v_{n-1}]$, $[\widehat{v}_0, ..., \widehat{v}_{n-1}]$, and index $i, \widehat{i}$,*
if $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, $i \cong_\psi \widehat{i}$, and $v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge v_m$, then $v \cong_\psi \widehat{v}_{\widehat{i}}$.

Lemma 3.2.10. *Given map ψ, array $[v_0, ..., v_{n-1}]$, $[\widehat{v}_0, ..., \widehat{v}_{n-1}]$, and index $i, \widehat{i}$,*
if $[v_0, ..., v_{n-1}] = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, $i \cong_\psi \widehat{i}$, and $v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge \text{encrypt}(v_m)$, then $v \cong_\psi \widehat{v}_{\widehat{i}}$.

Lemma 3.2.11. *Given map ψ, array $[v_0, ..., v_{n-1}]$, $[\widehat{v}_0, ..., \widehat{v}_{n-1}]$, value $v, \widehat{v}$, and index $i, \widehat{i}$, if $[v_0, ..., v_{n-1}] = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, $0 < i < n$, $i \cong_\psi \widehat{i}$, and $\forall v_m \in [v_0, ..., v_{n-1}]$. $v'_m = ((i = \text{encrypt}(m)) \wedge v) \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, then $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$ and $[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}]$.*

Lemma 3.2.12. *Given map ψ, array $[v_0, ..., v_{n-1}]$, $[\widehat{v}_0, ..., \widehat{v}_{n-1}]$, value $v, \widehat{v}$, and index $i, \widehat{i}$, if $[v_0, ..., v_{n-1}] = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, $0 < i < n$, $i \cong_\psi \widehat{i}$, and $\forall v_m \in [v_0, ..., v_{n-1}]$. $v'_m = ((i = \text{encrypt}(m)) \wedge v') \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, then $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$ and $[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}]$.*

Lemma 3.2.13 ($\text{encrypt}(n) \in \sigma \implies \hat{n} \in \widehat{\sigma}$)**.** *Given program traces Π and Σ such that $\Pi \cong_\psi \Sigma$ and values n and $\hat{n}$ such that $n \cong \hat{n}$, if an encrypted value n is written to memory σ at location l in trace Π, then the corresponding decrypted value $\hat{n}$ is written to memory $\widehat{\sigma}$ at the ψ-*congruent *location $\widehat{l}$ in trace Σ.*

Lemma 3.2.14. *Given ψ, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$ such that $x \in \gamma$ and $\widehat{x} \in \widehat{\gamma}$,*
if $\gamma(x) = (l, ty)$, then $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{ty})$ where $l = \widehat{l}$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, and $ty \cong \widehat{ty}$.

Proof. This holds by Definition 3.2.15 and the definition of function Erase. □

Lemma 3.2.15. *Given ψ, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l \cong_\psi \widehat{l}$ such that $l \in \sigma$ and $\widehat{l} \in \widehat{\sigma}$,*
if $\sigma(l) = (\omega, ty, n, \text{PermL}(perm, ty, a, n))$,
then $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{ty}, \widehat{n}, \text{PermL}(perm, \widehat{ty}, \text{public}, \widehat{n}))$ where $\omega \cong_\psi \widehat{\omega}$, $ty \cong \widehat{ty}$, $n = \widehat{n}$, and $perm = perm$.

Proof. This holds by Definition 3.2.15 and the definition of function Erase. □

Lemma 3.2.16. *Given ψ, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l \cong_\psi \widehat{l}$ such that $l \in \sigma$ and $\widehat{l} \in \widehat{\sigma}$,*
if $\sigma(l) = (\omega, a\ bty*, n, \text{PermL}(perm, ty, a, n))$,
then $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(perm, \widehat{ty}, \text{public}, 1))$ where $\omega \cong_\psi \widehat{\omega}$, $ty \cong \widehat{ty}$, and $perm = perm$.

Proof. This holds by Definition 3.2.15 and the definition of function Erase. □

Lemma 3.2.17. *Given ψ and $(\gamma, \sigma, \text{acc}, s) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$, if $(\gamma, \sigma, \text{acc}, s) \Downarrow_d (\gamma_1, \sigma_1, \text{acc}, v)$ and $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s}) \Downarrow_{\widehat{d}} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v})$ such that $(\gamma_1, \sigma_1, \text{acc}, v) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v})$, then $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$.*

Proof. Proof Idea:

Proof by induction over congruent evaluations. Using the definition of function Erase, we show that with every rule that adds to γ or adds to or modifies σ maintains both $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$ and $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ by Definition 3.2.15. □

Lemma 3.2.18. *Given array* $[v_0, ..., v_{n-1}], [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, *if* $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$ *then* $n = \widehat{n}$.

Proof. By definition of $\text{Erase}([v_0, ..., v_{n-1}])$. □

Lemma 3.2.19. *Given expression* $v, \widehat{v}$, *if* $(l, \mu) \neq v$ *and* $\text{Erase}(v) = \widehat{v}$, *then* $v \cong_\psi \widehat{v}$ *for any possible* ψ.

Proof. By Definition 3.2.10. □

Lemma 3.2.20. *Given evaluation trace* $\Pi \triangleright (\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, v)$, $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v})$, *if* $(l, \mu) \neq v$ *and* $\text{Erase}(e) = \widehat{e}$, *then* $e \cong_\psi \widehat{e}$ *for any possible map* ψ.

Proof. By Definition 3.2.10 and case analysis of SMC^2 semantic rules. □

Lemma 3.2.21. *Given an initial map* ψ, *environment* γ, *memory* σ, *accumulator* acc, *and stmt* s,
if $(\gamma, \sigma, \text{acc}, s) \Downarrow_d (\gamma_1, \sigma_1, \text{acc}, v)$ *and* $\text{pfree}(e) \notin s$, *then the ending map* ψ_1 *is equivalent to* ψ.

Proof. By definition of ψ. □

Lemma 3.2.22. *Given binary operation bop and value* $v_1, v_2, \widehat{v}_1, \widehat{v}_2$,
if $v_3 = v_1 \ bop_a \ v_2$, $v_1 \cong \widehat{v}_1$, *and* $v_2 \cong \widehat{v}_2$, *then* $\widehat{v}_3 = \widehat{v}_1 \ bop \ \widehat{v}_2$ *such that* $v_3 \cong \widehat{v}_3$.

Proof. By definition of Erase. □

Lemma 3.2.23. *For every SMC^2 execution of an* `if else` *statement branching on private data at any level of nesting, the memory block identifiers, in symbols* l_{then}, l_{else} *given by phi for temporary variables used to track changes inside the* ***`if else`*** *statement, in symbols* x_{then}, x_{else}, *will be unique and selected from a separate pool of memory block identifiers designated for this use.*

Proof. Proof Idea:

By definition of ϕ, when called with argument $temp$, in symbols $\phi(temp)$, a unique memory block identifier will be given from a separate pool of memory block identifiers than those accessed from a regular call of $\phi()$. Algorithm InitializeVariables is the only place in the semantics where this property of ϕ is leveraged. □

Lemma 3.2.24. *Given type* ty, $\widehat{ty}$ *and value* $n, \widehat{n}$,
if $n_1 = \text{Cast}(\text{public}, ty, n)$, $ty \cong \widehat{ty}$, *and* $n = \widehat{n}$ *then* $\widehat{n}_1 = \text{Cast}(\text{public}, \widehat{ty}, \widehat{n})$ *such that* $n_1 = \widehat{n}_1$.

Proof. By definition of algorithm Cast and Cast. □

Lemma 3.2.25. *Given type* ty, $\widehat{ty}$ *and number* n, $\widehat{n}$,
if $n_1 = \text{Cast}(\text{private},\ ty,\ n)$, $ty \cong \widehat{ty}$, *and* $n \cong_\psi \widehat{n}$ *then* $\widehat{n}_1 = \text{Cast}(\text{public},\ \widehat{ty},\ \widehat{n})$ *such that* $n_1 \cong_\psi \widehat{n}_1$.

Proof. By definition of algorithms Cast and Cast and function Erase. □

Lemma 3.2.26. *Given variable name* $x, \widehat{x}$ *and input party number* $n, \widehat{n}$ *such that the corresponding input files* inp_n, $\widehat{inp_n}$ *are congruent,*
if $\text{InputValue}(x, n) = n_1$, $x = \widehat{x}$, *and* $n = \widehat{n}$, *then* $\text{InputValue}(\widehat{x},\ \widehat{n}) = \widehat{n}_1$ *such that* $n_1 \cong \widehat{n}_1$.

Proof. By definition of algorithm InputValue and by Definition 3.2.25. □

Lemma 3.2.27. *Given variable name* $x, \widehat{x}$, *input party number* $n, \widehat{n}$ *such that the corresponding input files* inp_n, $\widehat{inp_n}$ *are congruent, and array length* $n_1, \widehat{n}_1$, *if* $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, $x = \widehat{x}$, $n = \widehat{n}$, *and* $n_1 = \widehat{n}_1$, *then* $\text{InputArray}(\widehat{x}, \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$ *such that* $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$.

Proof. By definition of algorithm InputArray and by Definition 3.2.25. □

Lemma 3.2.28. *Given variable name* $x, \widehat{x}$ *and input party number* $n, \widehat{n}$ *such that the corresponding input files* out_n, $\widehat{out}_{\widehat{n}}$ *are congruent, if* $\text{OutputValue}(x, n, n_1)$, $x = \widehat{x}$, $n = \widehat{n}$, *and* $n_1 \cong \widehat{n}_1$, *then* $\text{OutputValue}(\widehat{x}, \widehat{n}, \widehat{n}_1)$ *such that* $out_n \cong \widehat{out}_{\widehat{n}}$.

Proof. By definition of algorithm OutputArray and by Definition 3.2.26. □

Lemma 3.2.29. *Given variable name* x, $\widehat{x}$, *input party number* n, $\widehat{n}$ *such that the corresponding input files* out_n, $\hat{out}_{\widehat{n}}$ *are congruent, and array* $[m_0, ..., m_{n_1}]$, $[\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$,
if $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$, $x = \widehat{x}$, $n = \widehat{n}$, *and* $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$,
then $\text{OutputArray}(\widehat{x}, \widehat{n}, [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}])$ *such that* $out_n \cong \widehat{out}_{\widehat{n}}$.

Proof. By definition of algorithm OutputArray and by Definition 3.2.26. □

Lemma 3.2.30. *Given parameter list* $\overline{p}, \widehat{\overline{p}}$,
if $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$ *and* $\overline{p} \cong \widehat{\overline{p}}$, *then* $\text{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}}$ *where* $\overline{ty} \cong \widehat{\overline{ty}}$.

Proof. By definition of algorithm GetFunTypeList. □

Lemma 3.2.31. *Given parameter list* $\overline{p}, \widehat{\overline{p}}$ *and expression list* $\overline{e}, \widehat{\overline{e}}$, *if* $\text{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$, $\overline{p} \cong \widehat{\overline{p}}$, *and* $\overline{e} \cong \widehat{\overline{e}}$, *then* $\text{GetFunParamAssign}(\widehat{\overline{p}}, \widehat{\overline{e}}) = \widehat{s}_1$ *where* $s_1 \cong_\psi \widehat{s}_1$.

Proof. By definition of GetFunParamAssign. □

Lemma 3.2.32. *Given map ψ, memory $\sigma, \widehat{\sigma}$ and environment $\gamma, \widehat{\gamma}$ such that $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and variable name $x, \widehat{n}$, if $x \notin \gamma$ and $x = \widehat{x}$, then $\widehat{x} \notin \widehat{\gamma}$.*

Proof. By definition of function Erase and Definition 3.2.15. □

Lemma 3.2.33. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, variable name $x, \widehat{x}$, and type $\overline{ty} \to ty, \widehat{\overline{ty}} \to \widehat{ty}$, if $x \in \gamma$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $x = \widehat{x}$, $\overline{ty} \to ty \cong \widehat{\overline{ty}} \to \widehat{ty}$, and $\gamma(x) = (l, \overline{ty} \to ty)$ then $\widehat{x} \in \widehat{\gamma}$ such that $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{\overline{ty}} \to \widehat{ty})$ and $l = \widehat{l}$.*

Proof. By Definition 3.2.15, Erase, and Algorithms pfree. By Algorithm CheckFreeable, no memory block identifier that belongs to a variable in γ can be freed using pfree, and therefore l must be equal to $\widehat{l}$. □

Lemma 3.2.34. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, variable name $x, \widehat{x}$, memory block identifier $l, \widehat{l}$, and type $ty, \widehat{ty}$, if $\gamma_1 = \gamma[x \to (l, ty)]$, $x = \widehat{x}$, $l = \widehat{l}$, $ty \cong \widehat{ty}$, and $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, then $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \to (\widehat{l}, \widehat{ty})]$ such that $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma})$.*

Proof. By Definition 3.2.15 and the structure of the environment. □

Lemma 3.2.35. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, type $ty \in \{a\ bty, a\ \text{const}\ bty*, a\ bty*\}, \widehat{ty}$, byte representation $\omega, \widehat{\omega}$, number $n, \widehat{n}$, and permission $perm, \widehat{perm}$, if $\sigma_2 = \sigma_1[l \to (\omega, ty, n, \text{PermL}(perm, ty, a, n))]$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l \cong_\psi \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$, and $ty \cong \widehat{ty}$, then $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \to (\omega, ty, \widehat{n}, \text{PermL}(perm, \widehat{ty}, \text{public}, \widehat{n}))]$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.*

Proof. By Definition 3.2.15 and the structure of memory. □

Lemma 3.2.36. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, type $ty \in \{a\ bty, a\ \text{const}\ bty*, a\ bty*\}, \widehat{ty}$, byte representation $\omega, \widehat{\omega}$, number $n, \widehat{n}$, and permission $perm, \widehat{perm}$, if $\sigma_1 = \sigma_2[l \to (\omega, ty, n, \text{PermL}(perm, ty, a, n))]$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $l \cong_\psi \widehat{l}$, then $\widehat{\sigma}_1 = \widehat{\sigma}_2[\widehat{l} \to (\widehat{\omega}, \widehat{ty}, \widehat{n}, \text{PermL}(perm, \widehat{ty}, \text{public}, \widehat{n}))]$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $\omega \cong_\psi \widehat{\omega}$, $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$, $ty \cong \widehat{ty}$, and $perm = perm$.*

Proof. By Definition 3.2.15 and the structure of memory. □

Lemma 3.2.37. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, type $ty \in \{a\ bty, a\ bty*\}, \text{void}*, \widehat{ty}$, byte representation $\omega, \widehat{\omega}$, and number $n, \widehat{n}$, if $\sigma_1 = \sigma_2[l \to (\omega, \text{void}*, n, \text{PermL}(\text{Freeable}, ty, a, n))]$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $ty \cong \widehat{ty}$, and $l \cong_\psi \widehat{l}$, then $\widehat{\sigma}_1 = \widehat{\sigma}_2[\widehat{l} \to (\widehat{\omega}, \text{void}*, \widehat{n}, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, \widehat{n}))]$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $\omega \cong_\psi \widehat{\omega}$, and $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$.*

Proof. By Definition 3.2.15. □

Lemma 3.2.38. *Given map ψ, memory $\sigma, \widehat{\sigma}$ and environment $\gamma, \widehat{\gamma}$ such that $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and memory block identifier $l, \widehat{l}$, if $\text{Free}(\sigma, l, \gamma) = \sigma_1$ and $l \cong_\psi \widehat{l}$, then $\text{Free}(\widehat{\sigma}, \widehat{l}, \widehat{\gamma}) = \widehat{\sigma}_1$ such that $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$.*

Proof. By definition of Free, the ψ-congruent location will be marked as deallocated. □

Lemma 3.2.39. *Given map ψ, memory $\sigma, \widehat{\sigma}$ and environment $\gamma, \widehat{\gamma}$ such that $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and memory block identifier $l, \widehat{l}$ such that $l, \widehat{l}$ are* equally freeable *by Axiom 3.2.5,*
if $\mathrm{PFree}(\sigma, l, \gamma) = (\sigma_1, \bar{l}, \bar{j})$, $l \cong_\psi \widehat{l}$, $\mathrm{GetLocationSwap}(\bar{l}, \bar{j}) = \bar{l}'$, $\psi_1 = \psi[\bar{l}']$, $\mathrm{SwapMemory}(\widehat{\sigma}_1, \psi_1) = \widehat{\sigma}_2$, *and* $(\gamma, \sigma_1) \cong (\widehat{\gamma}, \widehat{\sigma}_2)$, *then* $\mathrm{Free}(\widehat{\sigma}, \widehat{l}, \widehat{\gamma}) = \widehat{\sigma}_1$ *such that* $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$.

Proof. Proof Idea:

If the number of locations α referred to by the pointer stored in memory block l is 1, then PFree calls Free and returns the updated memory from Free, the location that was freed, and tag list of $[\mathrm{encrypt}(0)]$. Therefore, by Lemma 3.2.38, we will have that $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$. We then call $\mathrm{GetLocationSwap}(\bar{l}, \bar{j}) = \bar{l}'$, which would obtain $\bar{l}' = [\,]$ and in turn $\psi_1 = \psi[\,]$. By definition of algorithm SwapMemory, $\mathrm{SwapMemory}(\widehat{\sigma}_1, \psi) = \widehat{\sigma}_2$ and $\mathrm{SwapMemory}(\widehat{\sigma}_1, \psi_1) = \widehat{\sigma}_2'$ such that $\psi_1 = \psi[\,]$, we have $\widehat{\sigma}_2 = \widehat{\sigma}_2'$. Therefore, also we have that $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$.

Otherwise, when $\alpha > 1$, we free the first location relating to memory block identifier l_0. If this location was the true location, then PFree will not swap any new locations, and in turn neither will SwapMemory, as calling $\mathrm{GetLocationSwap}(\bar{l}, \bar{j}) = \bar{l}'$ would obtain $\bar{l}' = [\,]$ and in turn $\psi_1 = \psi[\,]$. By definition of algorithm SwapMemory, $\mathrm{SwapMemory}(\widehat{\sigma}_1, \psi) = \widehat{\sigma}_2$ and $\mathrm{SwapMemory}(\widehat{\sigma}_1, \psi_1) = \widehat{\sigma}_2'$ such that $\psi_1 = \psi[\,]$, we have $\widehat{\sigma}_2 = \widehat{\sigma}_2'$. This gives us that $(\gamma, \sigma_2) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_2)$. If this location was not the true location, then PFree will return a tag list with two locations marked, the first being memory block identifier l_0, and the second at the memory block identifier indicated by the other marker in the tag list. Then $\mathrm{GetLocationSwap}(\bar{l}, \bar{j}) = \bar{l}'$ such that $\bar{l}' = [(l_1, 0), (l_2, 0)]$, $\psi_1 = \psi[\bar{l}']$, and $\mathrm{SwapMemory}(\widehat{\sigma}_1, \psi_1) = \widehat{\sigma}_2$ will then perform this swap in the Vanilla C memory so that we obtain $\widehat{\sigma}_2$ such that $(\gamma, \sigma_1) \cong (\widehat{\gamma}, \widehat{\sigma}_2)$. Therefore, by Definition 3.2.15 we have $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$. □

Lemma 3.2.40. *Given type $a\ bty$, $\widehat{bty}$, and value v, $\widehat{v}$, if* $\mathrm{EncodeVal}(a\ bty, v) = \omega$, *$a\ bty \cong \widehat{bty}$, and $v \cong_\psi \widehat{v}$, then* $\mathrm{EncodeVal}(\widehat{bty}, \widehat{v}) = \widehat{\omega}$ *such that $\omega \cong_\psi \widehat{\omega}$.*

Proof. By definition of Algorithm EncodeVal, EncodeVal and function Erase. □

Lemma 3.2.41. *Given type $a\ bty$, $\widehat{bty}$, number $n, \widehat{n}$, and byte representation $\omega, \widehat{\omega}$, if* $\mathrm{DecodeVal}(a\ bty, n, \omega) = v$, *$a\ bty \cong \widehat{bty}$, $n \cong \widehat{n}$, and $\omega \cong_\psi \widehat{\omega}$, then* $\mathrm{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}) = \widehat{v}$ *and $v \cong_\psi \widehat{v}$.*

Proof. By case analysis of the semantics, Lemma 3.2.40, definition of Algorithm DecodeVal, DecodeVal and function Erase. □

Lemma 3.2.42. *Given map ψ, pointer type $ty \in \{a\ \mathrm{const}\ bty*, a\ bty*\}$, $\widehat{ty} \in \{\mathrm{const}\ \widehat{bty}*, \widehat{bty}*\}$, and pointer data structure $[1, [(l, \mu)], [1], i]$, $[1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}]$, if* $\mathrm{EncodePtr}(ty, [1, [(l, \mu)], [1], i]) = \omega$, *$ty \cong \widehat{ty}$, $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$, then* $\mathrm{EncodePtr}(\widehat{ty}, [1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}]) = \omega'$ *such that $\omega \cong_\psi \widehat{\omega}$.*

Proof. By definition of Algorithm EncodePtr, EncodePtr, and definition of function Erase. □

Lemma 3.2.43. *Given map ψ, pointer type $ty \in \{\text{private const } bty*, \text{private } bty*\}$, $\widehat{ty} \in \{\text{const } \widehat{bty}*, \widehat{bty}*\}$, and pointer data structure $[\alpha, \overline{l}, \overline{j}, i]$, $[1, (\widehat{l}, \widehat{\mu}), [1], \widehat{i}]$, if $\text{EncodePtr}(ty, [\alpha, \overline{l}, \overline{j}, i]) = \omega$, $ty \cong \widehat{ty}$, and $\text{DeclassifyPtr}([\alpha, \overline{l}, \overline{j}, i], \text{private } bty*) = (l, \mu)$ such that $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$, then $\text{EncodePtr}(\widehat{ty}, [1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}]) = \widehat{\omega}$ such that $\omega \cong_\psi \widehat{\omega}$.*

Proof. By definition of Algorithm EncodePtr, EncodePtr, and definition of function Erase. □

Lemma 3.2.44. *Given map ψ, pointer type $ty \in \{a \text{ const } bty*, a\ bty*\}$, $\widehat{ty} \in \{\text{const } \widehat{bty}*, \widehat{bty}*\}$ and byte representation $\omega, \widehat{\omega}$, if $\text{DecodePtr}(ty, 1, \omega) = [1, (l, \mu), [1], i]$, $ty \cong \widehat{ty}$, and $\omega \cong_\psi \widehat{\omega}$, then $\text{DecodePtr}(\widehat{ty}, 1, \widehat{\omega}) = [1, (\widehat{l}, \widehat{\mu}), [1], \widehat{i}]$ such that $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$.*

Proof. By case analysis of the semantics, Lemma 3.2.42, definition of Algorithm DecodePtr, DecodePtr, and definition of Erase. □

Lemma 3.2.45. *Given pointer type $ty \in \{\text{private const } bty*, \text{private } bty*\}$, $\widehat{ty} \in \{\text{const } \widehat{bty}*, \widehat{bty}*\}$, number $\alpha, 1$, and byte representation $\omega, \widehat{\omega}$, if $\text{DecodePtr}(ty, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$, $ty \cong \widehat{ty}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha, \overline{l}, \overline{j}, i], \text{private } bty*) = (l, \mu)$, then $\text{DecodePtr}(\widehat{ty}, 1, \widehat{\omega}) = [1, (\widehat{l}, \widehat{\mu}), [1], \widehat{i}]$ such that $[\alpha, \overline{l}, \overline{j}, i] \cong [1, (\widehat{l}, \widehat{\mu}), [1], \widehat{i}]$ and $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$.*

Proof. By case analysis of the semantics, Lemma 3.2.43, definition of Algorithm DecodePtr, DecodePtr, definition of Erase, and Definition 3.2.14 (pointer list congruence). □

Lemma 3.2.46. *Given statement $s, \widehat{s}$, number n, and parameter list $\overline{p}, \widehat{\overline{p}}$, if $\text{EncodeFun}(s, n, \overline{p}) = \omega$, $s \cong \widehat{s}$, and $\overline{p} \cong \widehat{\overline{p}}$, then $\text{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}}) = \widehat{\omega}$ and $\omega \cong_\psi \widehat{\omega}$.*

Proof. By definition of Algorithm EncodeFun, EncodeFun, and definition of Erase. □

Lemma 3.2.47. *Given byte representation $\omega, \widehat{\omega}$, if $\text{DecodeFun}(\omega) = (s, n, \overline{p})$, and $\omega \cong_\psi \omega$, then $\text{DecodeFun}(\widehat{\omega}) = (\widehat{s}, \square, \widehat{\overline{p}})$, $s \cong \widehat{s}$ and $\overline{p} \cong \widehat{\overline{p}}$.*

Proof. By case analysis of the semantics, Lemma 3.2.46, definition of Algorithms DecodeFun and DecodeFun, and definition of function Erase. □

Lemma 3.2.48. *Given public type $ty, \widehat{ty}$, if $n = \tau(ty)$ and $ty \cong \widehat{ty}$, then $\widehat{n} = \tau(\widehat{ty})$ such that $n = \widehat{n}$.*

Proof. By definition of function Erase and the size of two public types being equivalent. □

Lemma 3.2.49. *Given $*, *$ if $\text{GetIndirection}(*) = i$ and $|*| = |*|$, then $\text{GetIndirection}(*) = \widehat{i}$ such that $i = \widehat{i}$.*

Proof. By definition of function Erase, when two types are congruent, their levels of indirection will be the same. Therefore, when we evaluate the level of indirection from the number of $*$, we will get the same number in both SMC^2 and Vanilla C. □

Lemma 3.2.50. *Given map ψ, location $(l_1, \mu_1), (\widehat{l_1}, \widehat{\mu_1})$, type $ty, \widehat{ty}$, number $n, \widehat{n}$, environment $\gamma, \widehat{\gamma}$, and memory $\sigma, \widehat{\sigma}$, if* $\text{GetLocation}((l_1,\ \mu_1),\ n, \sigma) = ((l_2, \mu_2), j)$, $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu_1})$, $ty \cong \widehat{ty}$, $\tau(ty) = n$, $\tau(\widehat{ty}) = \widehat{n}$, *and* $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, *then* $\text{GetLocation}((\widehat{l_1},\ \widehat{\mu_1}),\ \widehat{n}, \widehat{\sigma}) = ((\widehat{l_2},\ \widehat{\mu_2}), \widehat{j})$ *such that* $(l_2, \mu_2) \cong_\psi (\widehat{l_2}, \widehat{\mu_2})$ *and* $j \cong \widehat{j}$.

Proof. By definition of algorithms GetLocation and Erase and Definition 3.2.13. □

Lemma 3.2.51. *Given location list $\bar{l}$, location $(\widehat{l}, \widehat{\mu})$, type $ty, \widehat{ty}$, number $n, \widehat{n}$, map ψ, environment $\gamma, \widehat{\gamma}$, and memory $\sigma, \widehat{\sigma}$, if* $\text{IncrementList}(\bar{l}, n, \sigma) = (\bar{l}', j)$, $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ i],\ ty) = (l_1, \mu_1)$ *such that* $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu_1})$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $ty \cong \widehat{ty}$, $\tau(ty) = n$, $\tau(\widehat{ty}) = \widehat{n}$, *and* $\text{DeclassifyPtr}([\alpha,\ \bar{l}',\ \bar{j},\ i],\ \text{private } bty*) = (l_2, \mu_2)$, *then* $\text{GetLocation}((\widehat{l_1}, \widehat{\mu_1}), \widehat{n}, \widehat{\sigma}) = ((\widehat{l_2}, \widehat{\mu_2}), \widehat{j})$ *such that* $(l_2, \mu_2) \cong_\psi (\widehat{l_2}, \widehat{\mu_2})$ *and* $j \cong \widehat{j}$.

Proof. By definition of algorithms IncrementList, GetLocation, and Erase and Definitions 3.2.13 and 3.2.14. □

Lemma 3.2.52. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, value $v, \widehat{v}$, and type $a\ bty, \widehat{bty}$, if* $\text{UpdateVal}(\sigma_1, l, v, a\ bty) = \sigma_2$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l \cong_\psi \widehat{l}$, $v \cong_\psi \widehat{v}$, *and* $a\ bty \cong \widehat{bty}$, *then* $\text{UpdateVal}(\widehat{\sigma}_1, \widehat{l}, \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$ *such that* $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Proof. By definition of UpdateVal, UpdateVal, and Erase. □

Lemma 3.2.53. *Given map ψ, environment γ, $\widehat{\gamma}$, memory σ_1, $\widehat{\sigma}_1$, location $(l,\ \mu)$, $(\widehat{l},\ \widehat{\mu})$, value v, $\widehat{v}$, and type $a\ bty, \widehat{bty}$, if* $\text{UpdateOffset}(\sigma_1, (l, \mu), v, a\ bty) = (\sigma_2, j)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$, $v = \widehat{v}$, *and* $a\ bty \cong \widehat{bty}$, *then* $\text{UpdateOffset}(\widehat{\sigma}_1, (\widehat{l}, \widehat{\mu}), \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, \widehat{j})$ *such that* $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ *and* $j \cong \widehat{j}$.

Proof. By definition of Algorithm UpdateOffset, UpdateOffset, and Erase, as well as Definition 3.2.13 and 3.2.3. □

Lemma 3.2.54. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, location $(l, \mu), (\widehat{l}, \widehat{\mu})$, pointer data structure $[\alpha, \bar{l}, \bar{j}, i]$, $[1, [(\widehat{l_1}, \widehat{\mu_1})], [1], \widehat{i}]$, and type $a\ bty*, \widehat{bty}*$, if* $\text{UpdatePtr}(\sigma,\ (l, \mu),\ [\alpha, \bar{l}, \bar{j}, i],\ a\ bty*) = (\sigma_1, j)$, $a\ bty* \cong \widehat{bty}*$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$, *and* $[\alpha, \bar{l}, \bar{j}, i] \cong_\psi [1, [(\widehat{l_1}, \widehat{\mu_1})], [1], \widehat{i}]$, *then* $\text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, \widehat{\mu}),\ [1,\ [(\widehat{l_1}, \widehat{\mu_1})],\ [1],\ i],\ \widehat{bty}*) = (\widehat{\sigma}_1, \widehat{j})$ *such that* $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ *and* $j \cong \widehat{j}$.

Proof. By definition of UpdatePtr, UpdatePtr, and Erase, as well as Definition 3.2.14, 3.2.13, and 3.2.3. □

Lemma 3.2.55. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, number of locations α, memory block identifier list $\bar{l}$, location $(\widehat{l_1}, \widehat{\mu_1})$, tag list $\bar{j}, \widehat{\bar{j}}$, level of indirection i, type $\text{private } bty, \widehat{bty}$, and value $v, \widehat{v}$,*
if $\text{UpdatePriv}(\sigma_1, \alpha, \bar{l}, \bar{j}, \text{private } bty, v) = (\sigma_2, j)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ i],\ \text{private } bty*) =$

(l_1, μ_1), $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu_1})$, private $bty \cong \widehat{bty}$, *and* $v \cong_\psi \widehat{v}$,
then UpdateOffset$(\widehat{\sigma_1}, (\widehat{l_1}, \widehat{\mu_1}), \widehat{v}, \widehat{bty}) = (\widehat{\sigma_2}, \widehat{j})$ *such that* $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma_2})$ *and* $j \cong \widehat{j}$.

Proof. By definition of UpdatePriv, UpdateOffset, and Erase. □

Lemma 3.2.56. *Given map* ψ*, environment* $\gamma, \widehat{\gamma}$*, memory* $\sigma_1, \widehat{\sigma_1}$*, location* $(\widehat{l_1}, \widehat{\mu_1})$*, and pointer data structure* $[\alpha, \overline{l}, \overline{j}, i]$, $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1]$, $[1, [(\widehat{l_e}, \widehat{\mu_e})], [1], \widehat{i}-1]$ *and type* private $bty*$, $\widehat{bty}*$,
if UpdatePrivPtr$(\sigma_1, [\alpha, \overline{l}, \overline{j}, i], [\alpha_e, \overline{l}_e, \overline{j}_e, i-1]$, private $bty*) = (\sigma_2, j)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma_1})$, DeclassifyPtr$([\alpha, \overline{l}, \overline{j}, i]$, private $bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu_1})$, private $bty* \cong \widehat{bty}*$ *and* $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1] \cong_\psi [1, [(\widehat{l_e}, \widehat{\mu_e})], [1], \widehat{i}-1]$, *then* UpdatePtr$(\widehat{\sigma_1}, (\widehat{l_1}, \widehat{\mu_1}), [1, [(\widehat{l_e}, \widehat{\mu_e})], [1], \widehat{i}-1], \widehat{bty}*) = (\widehat{\sigma_2}, \widehat{j})$ *such that* $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma_2})$ *and* $j = \widehat{j}$.

Proof. By definition of UpdatePrivPtr, UpdatePtr, and Erase, as well as Definition 3.2.14 and 3.2.13. □

Lemma 3.2.57. *Given map* ψ*, environment* $\gamma, \widehat{\gamma}$*, memory* $\sigma, \widehat{\sigma}$*, type* private $bty*, \widehat{bty}*$*, pointer data structure* $[\alpha, \overline{l}, \overline{j}, i]$*, location* $(\widehat{l_1}, \widehat{\mu_1})$*, and level of indirection* $\widehat{i}$,
if DerefPrivPtr$(\alpha, \overline{l}, \overline{j}$, private $bty*, \sigma) = ((\alpha', \overline{l}', \overline{j}'), j)$, $i = \widehat{i}$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, DeclassifyPtr$([\alpha, \overline{l}, \overline{j}, i]$, private $bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu_1})$, DeclassifyPtr$([\alpha', \overline{l}', \overline{j}', i-1]$, private $bty*) = (l_2, \mu_2)$, *and* private $bty* \cong \widehat{bty}*$, *then* DerefPtrHLI$(\widehat{\sigma}, \widehat{bty}*, (\widehat{l_1}, \widehat{\mu_1})) = ([1, [(\widehat{l_2}, \widehat{\mu_2})], [1], \widehat{i}-1], 1)$ *such that* $[\alpha', \overline{l}', \overline{j}', i-1] \cong_\psi [1, [(\widehat{l_2}, \widehat{\mu_2})], [1], \widehat{i}-1]$, $(l_2, \mu_2) \cong_\psi (\widehat{l_2}, \widehat{\mu_2})$, *and* $j \cong \widehat{j}$.

Proof. By definition of DerefPrivPtr, DerefPrivPtr, and Erase. □

Lemma 3.2.58. *Given map* ψ*, environment* $\gamma, \widehat{\gamma}$*, memory* $\sigma, \widehat{\sigma}$*, type* public $bty, \widehat{bty}$*, and location* (l_1, μ_1), $(\widehat{l_1}, \widehat{\mu_1})$*,*
if DerefPtr$(\sigma$, public $bty, (l_1, \mu_1)) = (v, j)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, public $bty \cong \widehat{bty}$, *and* $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu_1})$, *then* $(\widehat{v}, \widehat{j}) =$ DerefPtr$(\widehat{\sigma}, \widehat{bty}, (\widehat{l_1}, \widehat{\mu_1}))$ *such that* $v \cong_\psi \widehat{v}$ *and* $j \cong \widehat{j}$.

Proof. By definition of DerefPtr, DerefPtr, and Erase. □

Lemma 3.2.59. *Given map* ψ*, environment* $\gamma, \widehat{\gamma}$*, memory* $\sigma, \widehat{\sigma}$*, type* public $bty*, \widehat{bty}*$*, and location* (l_1, μ_1), $(\widehat{l_1}, \widehat{\mu_1})$*,*
if DerefPtrHLI$(\sigma$, public $bty*, (l_1, \mu_1)) = ([1, [(l_2, \mu_2)], [1], i-1], j)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, public $bty* \cong \widehat{bty}*$, *and* $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu_1})$, *then* $([1, [(\widehat{l_2}, \widehat{\mu_2})], [1], \widehat{i}-1], \widehat{j}) =$ DerefPtrHLI$(\widehat{\sigma}, \widehat{bty}*, (\widehat{l_1}, \widehat{\mu_1}))$ *such that* $[1, [(l_2, \mu_2)], [1], i-1] \cong_\psi [1, [(\widehat{l_2}, \widehat{\mu_2})], [1], \widehat{i}-1]$, $(l_2, \mu_2) \cong_\psi (\widehat{l_2}, \widehat{\mu_2})$, *and* $j \cong \widehat{j}$.

Proof. By definition of DerefPtrHLI, DerefPtrHLI, and Erase. □

Lemma 3.2.60. *Given map* ψ*, environment* $\gamma, \widehat{\gamma}$*, memory* $\sigma, \widehat{\sigma}$*, type* private $bty, \widehat{bty}$*, pointer data structure* $[\alpha, \overline{l}, \overline{j}, 1]$*, and location* $(\widehat{l_1}, \widehat{\mu_1})$*, if* Retrieve_vals$(\alpha, \overline{l}, \overline{j}$, private $bty, \sigma) = (v, j)$, DeclassifyPtr$([\alpha, \overline{l}, \overline{j}, 1]$, private $bty*)$ $= (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu_1})$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, *and* private $bty \cong \widehat{bty}$, *then* DerefPtr$(\widehat{\sigma}, \widehat{bty}, (\widehat{l_1}, \widehat{\mu_1})) = (\widehat{v}, \widehat{j})$ *such that* $v \cong_\psi \widehat{v}$ *and* $j \cong \widehat{j}$.

Proof. By definition of DerefPtr, Retrieve_vals, and Erase. □

Lemma 3.2.61. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, type $a\ bty, \widehat{bty}$, value $v, \widehat{v}$, array index $i, \widehat{i}$ and size $n, \widehat{n}$, if* $\mathrm{WriteOOB}(v, i, n, l, a\ bty, \sigma_1) = (\sigma_2, j)$, $v \cong_\psi \widehat{v}$, $i = \widehat{i}$, $n = \widehat{n}$, $l = \widehat{l}$, $a\ bty \cong \widehat{bty}$, *and* $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, *then* $\mathrm{WriteOOB}(\widehat{v},\ \widehat{i},\ \widehat{n},\ \widehat{l},\ \widehat{bty},\ \widehat{\sigma}_1) = (\widehat{\sigma}_2,\ \widehat{j})$ *such that* $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ *and* $j \cong \widehat{j}$.

Proof. Proof Idea:

By definition of WriteOOB, if the number returned with the updated memory is 1, then the out of bounds access was *well-aligned* by Definition 3.2.2. Therefore, when we iterate over the ψ-*congruent* Vanilla C memory, the resulting out of bounds access will also be *well-aligned*. We use the definition of WriteOOB, WriteOOB, and Erase to help prove this. □

Lemma 3.2.62. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, memory block identifier $l, \widehat{l}$, type $a\ bty, \widehat{bty}$, and array index $i, \widehat{i}$ and size $n, \widehat{n}$, if* $\mathrm{ReadOOB}(i,\ n,\ l,\ a\ bty,\ \sigma) = (v,\ j)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $i = \widehat{i}$, $n = \widehat{n}$, $l = \widehat{l}$, *and* $a\ bty \cong \widehat{bty}$, *then* $\mathrm{ReadOOB}(\widehat{i},\ \widehat{n},\ \widehat{l},\ \widehat{bty},\ \widehat{\sigma}) = (\widehat{v},\ \widehat{j})$ *such that* $v \cong_\psi \widehat{v}$ *and* $j \cong \widehat{j}$.

Proof. By definition of ReadOOB, if the number returned with the updated memory is 1, then the out of bounds access was *well-aligned* by Definition 3.2.2. Therefore, when we iterate over the ψ-*congruent* Vanilla C memory, the resulting out of bounds access will also be *well-aligned*. We use the definition of ReadOOB, ReadOOB, and Erase to help prove this. □

Lemma 3.2.63. *Given array* $[v_0, ..., v_{n-1}]$, $[\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$ *and value* $v, \widehat{v}$,
if $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{v}{v_i}\right)$, $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, $i = \widehat{i}$ *and* $v \cong_\psi \widehat{v}$,
then $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$ *such that* $[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}]$.

Proof. Proof Idea:

This equation replaces the value at index $i, \widehat{i}$ with the new value $v, \widehat{v}$. Given that the initial arrays were ψ-*congruent* ($[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$), the new values are ψ- *congruent* ($v \cong_\psi \widehat{v}$) and we have the same index ($i = \widehat{i}$), then the resulting arrays will also be ψ-*congruent* ($[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}]$). We can prove this using Erase and Definition 3.2.10. □

Lemma 3.2.64. *Given* $(\gamma, \sigma, \mathrm{acc}, \mathrm{private\ int}\ res_\mathrm{acc} = n)$ *where* $\mathrm{Label}(n,\ \gamma) = \mathrm{private}$,
if $(\gamma,\ \sigma,\ \mathrm{acc},\ \mathrm{private\ int}\ res_\mathrm{acc}) \Downarrow_{ds} (\gamma_1,\ \sigma_1,\ \mathrm{acc},\ \mathrm{skip})$
then $\gamma_1 = \gamma :: \gamma_A$ *such that* $\gamma_A = [res_\mathrm{acc} \rightarrow (\mathrm{private\ int},\ l)]$ *and* $\sigma_1 = \sigma :: \sigma_A$ *such that* $\sigma_A = [l \rightarrow (\mathrm{EncodeVal}(\mathrm{private\ int}, n),\ \mathrm{private\ int},\ 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private\ int}, \mathrm{private}, 1))]$.

Proof.

Given $(\gamma,\ \sigma,\ \text{acc},\ \text{private int } res_\text{acc} = n)$ where $\text{Label}(n,\ \gamma) = \text{private}$, we have $(\gamma,\ \sigma,\ \text{acc},\ \text{private int } res_\text{acc} = n) \Downarrow_{ds} (\gamma_1,\ \sigma_2,\ \text{acc},\ \text{skip})$ by rule Declaration Assignment if $(\gamma,\ \sigma,\ \text{acc},\ \text{private int } res_\text{acc}) \Downarrow_{d1} (\gamma_1,\ \sigma_1,\ \text{acc},\ \text{skip})$ and $(\gamma_1,\ \sigma_1, \text{acc}, res_\text{acc} = n) \Downarrow_{w2} (\gamma_1,\ \sigma_2,\ \text{acc},\ \text{skip})$.

Given $(\gamma,\ \sigma,\ \text{acc},\ \text{private int } res_\text{acc})$, by rule Private Declaration we have $(\gamma,\ \sigma, \text{acc}, \text{private int } res_\text{acc}) \Downarrow_{d1} (\gamma_1,\ \sigma_1, \text{acc}, \text{skip})$ where $l = \phi()$, $\gamma_1 = \gamma[res_\text{acc} \rightarrow (l,\ \text{private int})]$,
$\omega = \text{EncodeVal}(\text{private int}, \text{NULL})$, $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private int}, 1, \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$.
By Axiom 3.2.2, we have $res_\text{acc} \notin \gamma$. By definition of ϕ, we have $l \notin \sigma$. Therefore, we have $\gamma_1 = \gamma[res_\text{acc} \rightarrow (\text{private int},\ l)]$ and $\sigma_1 = \sigma[l \rightarrow (\omega,\ \text{private int},\ 1,\ \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$.

Given $(\gamma_1,\ \sigma_1,\ \text{acc},\ res_\text{acc} = n)$ where $\text{Label}(res_\text{acc},\ \gamma_1) = \text{Label}(n,\ \gamma_1) = \text{private}$, by rule Write Private Variable we have $(\gamma_1,\ \sigma_1,\ \text{acc},\ res_\text{acc} = n) \Downarrow_{w2} (\gamma_1,\ \sigma_2,\ \text{acc},\ \text{skip})$ where $\gamma_1(res_\text{acc}) = (l,\ \text{private int})$ and $\text{UpdateVal}(\sigma_1, l, n, \text{private } bty) = \sigma_2$. By definition of UpdateVal, we have $\sigma_1 = \sigma_3[l \rightarrow (\omega,\ \text{private int},\ 1,\ \text{PermL}(\text{Freeable},\ \text{private int},\ \text{private},\ 1))]$ and $\sigma_2 = \sigma_3[l \rightarrow (\omega',\ \text{private int},\ 1,\ \text{PermL}(\text{Freeable},\ \text{private int},\ \text{private}, 1))]$. Given $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private int}, 1, \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$, we have $\sigma_3 = \sigma$ and therefore $\sigma_2 = \sigma[l \rightarrow (\omega',\ \text{private int}, 1,$
$\text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$.

Given $\gamma_1 = \gamma[res_\text{acc} \rightarrow (\text{private int},\ l)]$, we can conclude that $\gamma_1 = \gamma :: \gamma_A$ where $\gamma_A = [res_\text{acc} \rightarrow (\text{private int},\ l)]$.

Given $\sigma_2 = \sigma[l \rightarrow (\omega',\ \text{private int},\ 1,\ \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$, we can conclude that $\sigma_2 = \sigma :: \sigma_A$ where $\sigma_A = [l \rightarrow (\omega',\ \text{private int},\ 1,\ \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$.

Therefore, we can conclude $(\gamma,\ \sigma,\ \text{acc}, \text{private int } res = n) \Downarrow_{ds} (\gamma :: \gamma_A,\ \sigma :: \sigma_A,\ \text{acc},\ \text{skip})$ by rule Declaration Assignment.

□

Lemma 3.2.65. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, accumulator* acc*, and statement* $\text{private int } res_{\text{acc}+1} = n$*, if* $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ *and* $(\gamma,\ \sigma,\ \text{acc},\ \text{private int } res_{\text{acc}+1} = n) \Downarrow_{d_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ *then* $(\gamma_1,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$.

Proof. By definition of Erase. □

Lemma 3.2.66. *Given statement* s*, environment* γ*, memory* σ*, and accumulator* $\mathrm{acc} > 0$*, the data stored at the memory block identified by* l *can be modified within* σ *during the execution of* s *if and only if either* $x = e \in s$ *and* $\gamma(x) = (l, ty)$ *or if* $x[e_1] = e_2 \in s$ *or* $x = e \in s$ *such that* $\gamma(x) = (l_1, \text{private const } bty*)$*,* $\sigma(l_1) = (\omega, \text{private const } bty*, 1, \mathrm{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$*, and* $\mathrm{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l, 0)], [1], 1]$.

Proof. Proof by contradiction to show that no modifications can be made to memory σ except those from assignments are allowed within the scope (i.e., $\mathrm{acc} > 0$) of the if else branching on a private condition. □

Lemma 3.2.67. *Given statement* s_1, s_2*, memory* σ*,*
if $\mathrm{Extract_variables}(s_1, s_2) = x_{list_acc+1}$*, then* $\forall x \notin x_{list_acc+1}, x = e \notin \{s_1;s_2\}$.

Proof. By Algorithm ExtractVariables, all cases where variables are assigned to within either statement will be found. □

Lemma 3.2.68. *Given environment* γ*, temporary variable environment* γ_A*, memory* σ*, temporary variable memory* σ_A*, accumulator* acc*, and statement* s*, if* $\gamma_1 = \gamma :: \gamma_A$*,* $\sigma_1 = \sigma :: \sigma_A$*, and* $(\gamma_1, \sigma_1, \mathrm{acc}+1, s) \Downarrow_{d_1} (\gamma_2, \sigma_2, \mathrm{acc}+1, \mathrm{skip})$*, then* $\gamma_2 = \gamma_2' :: \gamma_A$*, and* $\sigma_2 = \sigma_2' :: \sigma_A$.

Proof. By Axiom 3.2.2. □

Lemma 3.2.69. *Given variable name list* x_{list}*, environment* γ*, memory* σ*, and accumulator* acc*,*
if $\mathrm{InitializeVariables}(x_{list}, \gamma, \sigma, \mathrm{acc}) = (\gamma', \sigma')$*, then* $\forall x \in x_{list}$*,* $\{x_{then_acc}, x_{else_acc}\} \in \gamma'$.

Proof. By definition of Algorithm InitializeVariables. □

Lemma 3.2.70. *Given variable name list* x_{list}*, environment* γ*, memory* σ*, and accumulator* acc*,*
if $\mathrm{InitializeVariables}(x_{list}, \gamma, \sigma, \mathrm{acc}) = (\gamma', \sigma')$*, then* $\gamma' = \gamma :: \gamma_1$ *and* $\sigma' = \sigma :: \sigma_1$.

Proof. By definition of Algorithm InitializeVariables and Lemma 3.2.69. □

Lemma 3.2.71. *Given map* ψ*, variable list* x_{list_acc+1}*, environment* $\gamma, \widehat{\gamma}$*, memory* $\sigma, \widehat{\sigma}$*, and accumulator* acc*, if* $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ *and* $\mathrm{InitializeVariables}(x_{list_acc+1}, \gamma, \sigma, \mathrm{acc}+1) = (\gamma_1, \sigma_1)$ *then* $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$.

Proof. By definition of Erase. □

Lemma 3.2.72. *Given an initial map* ψ*, environment* γ*, memory* σ*, accumulator* acc*,*
if $(\gamma, \sigma, \mathrm{acc}+1, s) \Downarrow_{d_1} (\gamma_1, \sigma_1, \mathrm{acc}+1, v)$ *then* $\mathrm{pfree}(e) \notin s$ *and* $\psi_1 = \psi$.

Proof. Proof by contradiction over the semantics showing that when $\mathrm{acc} > 0$, $\mathrm{pfree}(e)$ cannot be executed and therefore $\mathrm{pfree}(e) \notin s$ if $(\gamma, \sigma, \mathrm{acc}+1, s) \Downarrow_{d_1} (\gamma_1, \sigma_1, \mathrm{acc}+1, v)$. □

Lemma 3.2.73. *Given map ψ, environment $\gamma, \widehat{\gamma}$, temporary variable environment γ_A, memory $\sigma, \widehat{\sigma}$, and temporary variable memory σ_A, if $(\gamma :: \gamma_A,\ \sigma :: \sigma_A) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, then $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$.*

Proof. By definition of Erase and Definition 3.2.15. □

Lemma 3.2.74. *Given statement s_1, s_2, environment γ, memory σ, and accumulator* acc,
if Extract_variables$(s_1,\ s_2) = x_{list_acc+1}$, InitializeVariables$(x_{list_acc+1},\ \gamma,\ \sigma,\ \mathrm{acc}+1) = (\gamma :: \gamma_A,\ \sigma :: \sigma_A)$,
and $(\gamma :: \gamma_A,\ \sigma :: \sigma_A, \mathrm{acc}+1,\ s_1) \Downarrow_{d_1} (\gamma_1 :: \gamma_A,\ \sigma_1 :: \sigma_A,\ \mathrm{acc}+1,\ \mathrm{skip})$,
then $\forall x \in x_{list_acc+1}$, the corresponding temporary variable x_{else_acc} maintains the original value for x from the starting memory σ, and the only differences between memory σ and σ_1 that can occur are stored at the memory blocks with identifiers l such that $\gamma(x) = (l, ty)$.

Proof. By definition of InitializeVariables, if the portion of the memory containing the temporary *then* and *else* variables remains unmodified through the subsequent execution of any allowed statement s_1, then x_{else_acc} will maintain the original value stored in x from the starting memory σ. By Lemma 3.2.66 we have that the only differences between memory σ and σ_1 that can occur are stored at the memory blocks with identifiers l such that $\gamma(x) = (l, ty)$. □

Lemma 3.2.75. *Given original memory σ, updated memory σ_1, temporary memory σ_A, σ_B such that σ_B stores the temporary x_{then_acc} and x_{else_acc} variables, updated environment γ_1, and accumulator* acc,
if $\forall x \in x_{list_acc+1}$, the corresponding temporary variable x_{else_acc} maintains the original value for x from the starting memory σ, and the only differences between memory σ and σ_1 that can occur are stored at the memory blocks with identifiers l such that $\gamma(x) = (l, ty)$, and RestoreVariables$(x_{list_acc+1}, \gamma_1 :: \gamma_A :: \gamma_B, \sigma_1 :: \sigma_A :: \sigma_B, \mathrm{acc}+1) = \sigma_2$, *then $\sigma_2 = \sigma :: \sigma_A :: \sigma_C$ such that $\forall x \in x_{list_acc+1}$, x_{then_acc} remains unchanged in σ_C, x_{then_acc} is updated with the modified values for x from the execution of the* **`then`** *branch, and x is updated to its original value from σ.*

Proof. By definition of Algorithm RestoreVariables. □

Lemma 3.2.76. *Given map ψ, variable list x_{list_acc+1}, private condition result variable name $res_{\mathrm{acc}+1}$, accumulator* acc, *starting environment $\gamma, \widehat{\gamma}$,* **`else`** *environment γ_e, temporary variable environment γ_A,* **`then`** *memory $\sigma_t, \widehat{\sigma}_t$,* **`else`** *memory $\sigma_e, \widehat{\sigma}_e$, and temporary variable environment σ_A,*
if ResolveVariables$(x_{list_acc+1}, \gamma_e :: \gamma_A,\ \sigma_e :: \sigma_A, \mathrm{acc}+1,\ res_{\mathrm{acc}+1}) = \sigma_7$, $res_{\mathrm{acc}+1} \neq_{\mathrm{private}} \mathrm{encrypt}(0)$, σ_A *stores all modifications made to that variable within the* **`then`** *branch using the temporary variables x_{then_acc}, and $(\gamma, \sigma_t) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_t)$, then $\sigma_f = \sigma_t :: \sigma_A$ such that $(\gamma, \sigma_f) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_t)$.*

Proof. By definition of Algorithm ResolveVariables, we have $\sigma_f = \sigma_t :: \sigma_A$. By Definition 3.2.15 we have $(\gamma, \sigma_t :: \sigma_A) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_t)$. □

Lemma 3.2.77. *Given map ψ, variable list x_{list_acc+1}, private condition result variable name res_{acc+1}, accumulator acc, starting environment $\gamma, \widehat{\gamma}$, **else** environment γ_e, temporary variable environment γ_A, **then** memory $\sigma_t, \widehat{\sigma}_t$, **else** memory $\sigma_e, \widehat{\sigma}_e$, and temporary variable environment σ_A,*
if ResolveVariables(x_{list_acc+1}, $\gamma_e :: \gamma_A$, $\sigma_e :: \sigma_A$, $acc + 1$, res_{acc+1}) $= \sigma_f$, $res_{acc+1} =_{private}$ encrypt(0), $(\gamma, \sigma_e) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_e)$ *then* $\sigma_f = \sigma_e :: \sigma_A$ *such that* $(\gamma, \sigma_f) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_e)$.

Proof. By definition of Algorithm ResolveVariables, we have $\sigma_f = \sigma_e :: \sigma_A$. By Definition 3.2.15 we have $(\gamma, \sigma_e :: \sigma_A) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_e)$. □

3.2.4 Proof of Correctness

Theorem 3.2.2 (Semantic Correctness)**.** *Given configuration (γ, σ, acc, s), $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$ and map ψ such that $(\gamma, \sigma, acc, s) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$, if $\Pi \triangleright (\gamma, \sigma, acc, s) \Downarrow_{[d_1,...,d_n]} (\gamma_1, \sigma_1, acc, v_1)$ for codes $[d_1, ..., d_n] \in SmcC \setminus SmcCX$, then there exists a derivation $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s}) \Downarrow_{[\widehat{d}_1,...,\widehat{d}_m]} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v}_1)$ for codes $[\widehat{d}_1, ..., \widehat{d}_m] \in VanC \setminus VanCX$ and a map ψ_1 such that $[d_1, ..., d_n] \cong [\widehat{d}_1, ..., \widehat{d}_m]$, $(\gamma_1, \sigma_1, acc, v_1) \cong_{\psi_1} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v}_1)$, and $\Pi \cong_{\psi_1} \Sigma$.*

Proof.
Case $\Pi \triangleright (\gamma, \sigma, acc, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma_2, acc, n_3)$
Given $\Pi \triangleright (\gamma, \sigma, acc, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma_2, acc, n_3)$ by SMC^2 rule Private Less Than True, we have Label$(e_1, \gamma) =$ Label$(e_2, \gamma) = private$, $(\gamma, \sigma, acc, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, acc, n_1)$, $(\gamma, \sigma_1, acc, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, acc, n_2)$, $n_1 <_{private} n_2$, and encrypt$(1) = n_3$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2)$ and ψ such that $(\gamma, \sigma, acc, e_1 < e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 < e_2 \cong_\psi \widehat{e}_1 < \widehat{e}_2$. Given $(\gamma, \sigma, acc, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma_2, acc, n_3)$, by Lemma 3.2.2 we have $(l, \mu) \notin e_1 < e_2$. Therefore, by Lemma 3.2.3, we have $e_1 < e_2 \cong \widehat{e}_1 < \widehat{e}_2$. By Definition 3.2.10 we have $e_1 < e_2 =$ Erase$(e_1) <$ Erase$(e_2) = \widehat{e}_1 < \widehat{e}_2$, and therefore $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma, acc, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, acc, n_1)$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$, $(l, \mu) \notin e_1 < e_2$, and ψ such that $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\gamma, \sigma, acc, e_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$ and ψ_1 such that $(\gamma, \sigma_1, acc, n_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$ and $d_1 \cong \widehat{d}_1$. Given $n_1 \neq$ skip, by Lemma 3.2.1 we have $\psi_1 = \psi$, therefore $(\gamma, \sigma_1, acc, n_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$. Therefore, by Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$.

Given $(\gamma, \sigma_1, acc, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, acc, n_2)$, ψ, $(l, \mu) \notin e_1 < e_2$, and $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\gamma, \sigma_1, acc, e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}_1,$

$\square, \widehat{e}_2) \Downarrow_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n}_2)$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, n_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \widehat{n}_2)$ and $d_2 \cong \widehat{d}_2$. Given $n_2 \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$ and therefore $(\gamma, \sigma_2, \text{acc}, n_2) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \widehat{n}_2)$. Therefore, by Definition 3.2.20 we have $(\gamma,\ \sigma_2) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_2 \cong_{\psi} \widehat{n}_2$.

Given $n_1 <_{\text{private}} n_2$, $n_1 \cong_{\psi} \widehat{n}_1$, and $n_2 \cong_{\psi} \widehat{n}_2$, by Lemma 3.2.5 we have $\widehat{n}_1 < \widehat{n}_2$.

Given $\text{encrypt}(1) = n_3$ and $\text{Erase}(\text{encrypt}(1)) = 1$ by Definition 3.2.17 we have $n_3 \cong_{\psi} 1$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n}_1)$, $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n}_2)$, and $\widehat{n}_1 < \widehat{n}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{\widehat{ltt}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, 1)$ by Vanilla C rule Less Than True.

Given $(\gamma,\ \sigma_2) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_3 \cong_{\psi} 1$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, n_3) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, 1)$. Therefore, we have $(\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma_2, \text{acc}, n_3) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{\widehat{ltt}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, 1)$, $\Pi \cong_{\psi} \Sigma$, and $ltt1 \cong \widehat{ltt}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltf1} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqt1} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqf1} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{nef1} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{net1} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$ by SMC^2 rule Public-Private Less Than True, we have $\text{Label}(e_1,$

$\gamma) = \text{public}$, $\text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n_1)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n_2)$, $\text{encrypt}(n_1) <_{\text{private}} n_2$, and $\text{encrypt}(1) = n_3$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2)$ and ψ such that $(\gamma, \sigma, \text{acc}, e_1 < e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 < e_2 \cong_\psi \widehat{e}_1 < \widehat{e}_2$. Given $(\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$, by Lemma 3.2.2 we have $(l, \mu) \notin e_1 < e_2$. Therefore, by Lemma 3.2.3, we have $e_1 < e_2 \cong \widehat{e}_1 < \widehat{e}_2$. By Definition 3.2.10 we have $e_1 < e_2 = \text{Erase}(e_1) < \text{Erase}(e_2) = \widehat{e}_1 < \widehat{e}_2$, and therefore $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n_1)$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$, $(l, \mu) \notin e_1 < e_2$, and ψ such that $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\gamma, \sigma, \text{acc}, e_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, n_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$ and $d_1 \cong \widehat{d}_1$. Given $n_1 \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$, therefore $(\gamma, \sigma_1, \text{acc}, n_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$. Therefore, by Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$, and by definition of Erase, we have $n_1 = \widehat{n}_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n_2)$, ψ, $(l, \mu) \notin e_1 < e_2$, and $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\gamma, \sigma_1, \text{acc}, e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, n_2) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$ and $d_2 \cong \widehat{d}_2$. Given $n_2 \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$, therefore $(\gamma, \sigma_2, \text{acc}, n_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$. Therefore, by Definition 3.2.20 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $n_2 \cong_\psi \widehat{n}_2$.

Given $\text{encrypt}(n_1) <_{\text{private}} n_2$, $n_1 = \widehat{n}_1$, and $n_2 \cong_\psi \widehat{n}_2$, by Definition 3.2.10 we have $\text{Erase}(\text{encrypt}(n_1)) = \widehat{n}_1$, and therefore $\text{encrypt}(n_1) \cong_\psi \widehat{n}_1$ by Definition 3.2.17. Therefore by Lemma 3.2.5 we have $\widehat{n}_1 < \widehat{n}_2$.

Given $\text{encrypt}(1) = n_3$ and $\text{Erase}(\text{encrypt}(1)) = 1$ by Definition 3.2.17 we have $n_3 \cong_\psi 1$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2)$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$, $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$, and $\widehat{n}_1 < \widehat{n}_2$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{\widehat{ltt}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, 1)$ by Vanilla C rule Less Than True.

Given $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $n_3 \cong_\psi 1$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, n_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \square, 1)$. Therefore, we have $(\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{\widehat{ltt}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, 1)$, $\Pi \cong_\psi \Sigma$ and $ltt2 \cong \widehat{ltt}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltf2} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqt2} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqf2} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{net2} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{nef2} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt3} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltf3} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqt3} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqf3} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{net3} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{nef3} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$ by SMC^2 rule Public Less Than True, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n_1)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n_2)$, and $n_1 <_{\text{public}} n_2$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2)$ and ψ such that $(\gamma, \sigma, \text{acc}, e_1 < e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 < e_2 \cong_\psi \widehat{e}_1 < \widehat{e}_2$. Given $(\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, n_3)$, by Lemma 3.2.2 we have $(l, \mu) \notin e_1 < e_2$. Therefore, by Lemma 3.2.3, we have $e_1 < e_2 \cong \widehat{e}_1 < \widehat{e}_2$. By Definition 3.2.10 we have $e_1 < e_2 = \text{Erase}(e_1) < \text{Erase}(e_2) = \widehat{e}_1 < \widehat{e}_2$, and therefore $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n_1)$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$, $(l, \mu) \notin e_1 < e_2$, and ψ such that $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\gamma, \sigma, \text{acc}, e_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, n_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$ and $d_1 \cong \widehat{d}_1$. Given $n_1 \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$, therefore $(\gamma, \sigma_1, \text{acc}, n_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$. Therefore, by Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$, and by definition of Erase, we have $n_1 = \widehat{n}_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n_2)$, ψ, $(l, \mu) \notin e_1 < e_2$, and $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\gamma, \sigma_1, \text{acc}, e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, n_2) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$ and $d_2 \cong \widehat{d}_2$. Given $n_2 \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$, therefore $(\gamma, \sigma_2, \text{acc}, n_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$. Therefore, by Definition 3.2.20 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $n_2 \cong_\psi \widehat{n}_2$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n_2, \gamma) = \text{public}$, and by definition of Erase, we have $n_2 = \widehat{n}_2$.

Given $n_1 <_{\text{public}} n_2$, $n_1 = \widehat{n}_1$, and $n_2 = \widehat{n}_2$, we have $\widehat{n}_1 < \widehat{n}_2$.

By Definition 3.2.17, we have $1 \cong_\psi 1$. Given $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $1 \cong_\psi 1$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, 1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \square, 1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2)$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$, $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$, and $\widehat{n}_1 < \widehat{n}_2$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{\widehat{ltt}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, 1)$ by Vanilla C rule Less Than True.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $1 \cong_\psi 1$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, 1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, 1)$. Therefore, we have $(\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{\widehat{ltt}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, 1)$, $\Pi \cong_\psi \Sigma$, and $ltt \cong \widehat{ltt}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltf} (\gamma, \sigma_2, \text{acc}, 0)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqt} (\gamma, \sigma_2, \text{acc}, 1)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqf} (\gamma, \sigma_2, \text{acc}, 0)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{net} (\gamma, \sigma_2, \text{acc}, 1)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{nef} (\gamma, \sigma_2, \text{acc}, 0)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp} (\gamma, \sigma_2, \text{acc}, n_3)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp} (\gamma, \sigma_2, \text{acc}, n_3)$ by SMC^2 rule Public Addition, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n_1)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n_2)$, and $n_1 +_{\text{public}} n_2 = n_3$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma,\ \text{acc},\ e_1 + e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 + e_2 \cong_\psi \widehat{e}_1 + \widehat{e}_2$. Given $(\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp} (\gamma, \sigma_2, \text{acc}, n_3)$, by Lemma 3.2.2 we have $(l, \mu) \notin e_1 + e_2$. Therefore, by Lemma 3.2.3, we have $e_1 + e_2 \cong \widehat{e}_1 + \widehat{e}_2$. By Definition 3.2.10 we have $e_1 + e_2 = \text{Erase}(e_1) + \text{Erase}(e_2) = \widehat{e}_1 + \widehat{e}_2$, and therefore $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma,\ \sigma_1,\ \text{acc},\ n_1)$, ψ, $(l, \mu) \notin e_1 + e_2$, and $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\gamma, \sigma, \text{acc}, e_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n}_1)$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \text{acc},\ n_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n}_1)$ and $d_1 \cong \widehat{d}_1$. Given $n_1 \neq \text{skip}$, by

Lemma 3.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1,\ \mathrm{acc},\ n_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{n}_1)$. Therefore, by Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$. By Definition 3.2.17 we have $n_1 \cong \widehat{n}_1$. Given $\mathrm{Label}(e_1, \gamma) = \mathrm{public}$, we have $\mathrm{Label}(n_1, \gamma) = \mathrm{public}$, and by definition of Erase, we have $n_1 = \widehat{n}_1$.

Given $(\gamma,\ \sigma_1,\ \mathrm{acc},\ e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \mathrm{acc}, n_2)$, ψ, $(l, \mu) \notin e_1 + e_2$, and $(\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{e}_2)$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\gamma,\ \sigma_1,\ \mathrm{acc},\ e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{e}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{e}_2) \Downarrow_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box, \widehat{n}_2)$ and ψ_2 such that $(\gamma, \sigma_2, \mathrm{acc}, n_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \widehat{n}_2)$ and $d_2 \cong \widehat{d}_2$. Given $n_2 \neq \mathrm{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$, therefore $(\gamma, \sigma_2, \mathrm{acc}, n_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \widehat{n}_2)$. Therefore, by Definition 3.2.20 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_2 \cong_\psi \widehat{n}_2$. By Definition 3.2.17 we have $n_2 \cong \widehat{n}_2$. Given $\mathrm{Label}(e_2, \gamma) = \mathrm{public}$, we have $\mathrm{Label}(n_2, \gamma) = \mathrm{public}$, and by definition of Erase, we have $n_2 = \widehat{n}_2$.

Given $n_1 +_{\mathrm{public}} n_2 = n_3$, $n_1 = \widehat{n}_1$, and $n_2 = \widehat{n}_2$, we have $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$ where $n_3 = \widehat{n}_3$. Therefore by Definition 3.2.17 we have $n_3 \cong_\psi \widehat{n}_3$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}_1 + \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}_1) \Downarrow_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{n}_1)$, $(\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{e}_2) \Downarrow_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \widehat{n}_2)$, and $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{\widehat{bp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \widehat{n}_3)$ by Vanilla C rule Addition.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_3 \cong_\psi \widehat{n}_3$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \mathrm{acc}, n_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \widehat{n}_3)$. Therefore, we have $(\gamma, \sigma, \mathrm{acc}, e_1{+}e_2) \Downarrow_{bp} (\gamma, \sigma_2, \mathrm{acc}, n_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{\widehat{bp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \widehat{n}_3)$, $\Pi \cong_\psi \Sigma$, and $bp \cong \widehat{bp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, e_1{-}e_2) \Downarrow_{bs} (\gamma, \sigma_2, \mathrm{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, e_1{+}e_2) \Downarrow_{bp} (\gamma,\ \sigma_2, \mathrm{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, e_1{\cdot}e_2) \Downarrow_{bm} (\gamma, \sigma_2, \mathrm{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, e_1{+}e_2) \Downarrow_{bp} (\gamma,\ \sigma_2, \mathrm{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, e_1{+}e_2) \Downarrow_{bp1} (\gamma, \sigma_2, \mathrm{acc}, n_3)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \mathrm{acc},\ e_1{+}e_2) \Downarrow_{bp1} (\gamma,\ \sigma_2,\ \mathrm{acc},\ n_3)$ by SMC^2 rule Private Addition, we have $\mathrm{Label}(e_1, \gamma) = \mathrm{Label}(e_2, \gamma) = \mathrm{private}$, $(\gamma, \sigma, \mathrm{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \mathrm{acc}, n_1)$, $(\gamma, \sigma_1, \mathrm{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \mathrm{acc}, n_2)$, and $n_1 +_{\mathrm{private}} n_2 = n_3$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2)$ and ψ such that $(\gamma, \sigma, \mathrm{acc}, e_1 + e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 + e_2 \cong_\psi \widehat{e}_1 + \widehat{e}_2$. Given $(\gamma, \sigma, \mathrm{acc}, e_1 + e_2) \Downarrow_{bp1} (\gamma, \sigma_2, \mathrm{acc}, n_3)$, by Lemma 3.2.2 we have $(l, \mu) \notin e_1 + e_2$. Therefore, by Lemma 3.2.3, we have $e_1 + e_2 \cong \widehat{e}_1 + \widehat{e}_2$. By Definition 3.2.10 we have $e_1 + e_2 = \mathrm{Erase}(e_1) + \mathrm{Erase}(e_2) = \widehat{e}_1 + \widehat{e}_2$, and therefore $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma, \mathrm{acc}, e_1) \Downarrow_{d_1} (\gamma,\ \sigma_1,\ \mathrm{acc},\ n_1)$, ψ, $(l, \mu) \notin e_1 + e_2$, and $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\gamma, \sigma, \mathrm{acc}, e_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n}_1)$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \mathrm{acc},\ n_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n}_1)$ and $d_1 \cong \widehat{d}_1$. Given $n_1 \neq \mathrm{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1,\ \mathrm{acc},\ n_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n}_1)$. Therefore, by Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$. By Definition 3.2.17 we have $n_1 \cong \widehat{n}_1$.

Given $(\gamma,\ \sigma_1,\ \mathrm{acc},\ e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \mathrm{acc}, n_2)$, ψ, $(l, \mu) \notin e_1 + e_2$, and $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\gamma,\ \sigma_1,\ \mathrm{acc},\ e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n}_2)$ and ψ_2 such that $(\gamma, \sigma_2, \mathrm{acc}, n_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \widehat{n}_2)$ and $d_2 \cong \widehat{d}_2$. Given $n_2 \neq \mathrm{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$, therefore $(\gamma, \sigma_2, \mathrm{acc}, n_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n}_2)$. Therefore, by Definition 3.2.20 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_2 \cong_\psi \widehat{n}_2$. By Definition 3.2.17 we have $n_2 \cong \widehat{n}_2$.

Given $n_1 +_{\mathrm{private}} n_2 = n_3$, $n_1 \cong \widehat{n}_1$, and $n_2 \cong \widehat{n}_2$, by Definition 3.2.10 we have $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$ where $n_3 \cong_\psi \widehat{n}_3$ by Definition 3.2.17.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n}_1)$, $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n}_2)$, and $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{\widehat{bp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n}_3)$ by Vanilla C rule Addition.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_3 \cong_\psi \widehat{n}_3$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \mathrm{acc}, n_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n}_3)$. Therefore, we have $(\gamma, \sigma, \mathrm{acc}, e_1{+}e_2) \Downarrow_{bp1} (\gamma, \sigma_2, \mathrm{acc}, n_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{\widehat{bp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n}_3)$, $\Pi \cong_\psi \Sigma$, and $bp1 \cong \widehat{bp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, e_1{-}e_2) \Downarrow_{bs1} (\gamma, \sigma_2, \mathrm{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, e_1{+}e_2) \Downarrow_{bp1} (\gamma,\ \sigma_2, \mathrm{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, e_1{\cdot}e_2) \Downarrow_{bm1} (\gamma, \sigma_2, \mathrm{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, e_1{+}e_2) \Downarrow_{bp1} (\gamma,\ \sigma_2, \mathrm{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1{+}e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1{+}e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$ by SMC^2 rule Public-Private Addition, we have $\text{Label}(e_1, \gamma) = \text{public}$, $\text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n_1)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n_2)$, and $\text{encrypt}(n_1) +_{\text{private}} n_2 = n_3$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2)$ and ψ such that $(\gamma, \sigma, \text{acc}, e_1 + e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 + e_2 \cong_\psi \widehat{e}_1 + \widehat{e}_2$. Given $(\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$, by Lemma 3.2.2 we have $(l, \mu) \notin e_1 + e_2$. Therefore, by Lemma 3.2.3, we have $e_1 + e_2 \cong \widehat{e}_1 + \widehat{e}_2$. By Definition 3.2.10 we have $e_1 + e_2 = \text{Erase}(e_1) + \text{Erase}(e_2) = \widehat{e}_1 + \widehat{e}_2$, and therefore $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n_1)$, ψ, $(l, \mu) \notin e_1 + e_2$, and $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\gamma, \sigma, \text{acc}, e_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, n_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$ and $d_1 \cong \widehat{d}_1$. Given $n_1 \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$, therefore $(\gamma, \sigma_1, \text{acc}, n_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$. Therefore, by Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$. By Definition 3.2.17 we have $n_1 \cong \widehat{n}_1$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$, and by definition of Erase, we have $n_1 = \widehat{n}_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n_2)$, ψ, $(l, \mu) \notin e_1 + e_2$, and $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\gamma, \sigma_1, \text{acc}, e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, n_2) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$ and $d_2 \cong \widehat{d}_2$. Given $n_2 \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$, therefore $(\gamma, \sigma_2, \text{acc}, n_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$. Therefore, by Definition 3.2.20 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $n_2 \cong_\psi \widehat{n}_2$. By Definition 3.2.17 we have $n_2 \cong \widehat{n}_2$.

Given $\text{encrypt}(n_1) +_{\text{private}} n_2 = n_3$, $n_1 = \widehat{n}_1$, and $n_2 \cong \widehat{n}_2$, by Definition 3.2.10 we have $\text{Erase}(\text{encrypt}(n_1)) = \widehat{n}_1$, and therefore $\text{encrypt}(n_1) \cong \widehat{n}_1$. By Definition 3.2.10, we have $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$ where $n_3 \cong \widehat{n}_3$. By Definition 3.2.17 we have $n_3 \cong_\psi \widehat{n}_3$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2)$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$, $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$, and $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{\widehat{bp}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_3)$ by Vanilla C rule Addition.

Given $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $n_3 \cong_\psi \widehat{n}_3$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, n_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_3)$. Therefore, we have $(\gamma, \sigma, \text{acc}, e_1{+}e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{\widehat{bp}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_3)$, $\Pi \cong_\psi \Sigma$, and $bp2 \cong \widehat{bp}$

by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 - e_2) \Downarrow_{bs2} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp3} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 - e_2) \Downarrow_{bs3} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 \cdot e_2) \Downarrow_{bm2} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 \cdot e_2) \Downarrow_{bm3} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iep} (\gamma, \sigma_7, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iep} (\gamma, \sigma_7, \text{acc}, \text{skip})$ by SMC^2 rule Private If Else, we have $\text{Label}(e, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, $(\gamma, \sigma_1, \text{acc}, \text{private int } res_{\text{acc}+1} = n) \Downarrow_{d_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$, $\text{Extract_variables}(s_1, s_2) = x_{list_\text{acc}+1}$, $\text{InitializeVariables}(x_{list_\text{acc}+1}, \gamma_1, \sigma_2, \text{acc}+1) = (\gamma_2, \sigma_3)$, $(\gamma_2, \sigma_3, \text{acc}+1, s_1) \Downarrow_{d_3} (\gamma_3, \sigma_4, \text{acc}+1, \text{skip})$,
$\text{RestoreVariables}(x_{list_\text{acc}+1}, \gamma_3, \sigma_4, \text{acc}+1) = \sigma_5$, $(\gamma_2, \sigma_5, \text{acc}+1, s_2) \Downarrow_{d_4} (\gamma_4, \sigma_6, \text{acc}+1, \text{skip})$, and $\text{ResolveVariables}(x_{list_\text{acc}+1}, \gamma_4, \sigma_6, \text{acc}+1, res_{\text{acc}+1}) = \sigma_7$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \Box, \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2)$ and ψ such that $(\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \Box, \text{if } (\widehat{e})\widehat{s}_1 \text{ else } \widehat{s}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $\text{if } (e)\ s_1 \text{ else } s_2 \cong_\psi \text{if } (\widehat{e})\widehat{s}_1 \text{ else } \widehat{s}_2$. Given $(\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iep} (\gamma, \sigma_7, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{if } (e)\ s_1 \text{ else } s_2$. Therefore, by Lemma 3.2.3, we have $\text{if } (e)\ s_1 \text{ else } s_2 \cong \text{if } (\widehat{e})\widehat{s}_1 \text{ else } \widehat{s}_2$. By Definition 3.2.10, we have $\text{Erase}(\text{if } (e)\ s_1 \text{ else } s_2) = \text{if } (\text{Erase}(e)\ \text{Erase}(s_1) \text{ else } \text{Erase}(s_2)$, $\text{Erase}(e) = \widehat{e}$, $\text{Erase}(s_1) = \widehat{s}_1$, and $\text{Erase}(s_2) = \widehat{s}_2$. Therefore, we have $e \cong \widehat{e}$, $s_1 \cong \widehat{s}_1$, and $s_2 \cong \widehat{s}_2$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, ψ, and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\gamma, \sigma, \text{acc}, e) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{e})$. Given $(\gamma, \sigma,$

$\text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, by the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{n})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, n) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{n})$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$, therefore $(\gamma, \sigma_1, \text{acc}, n) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{n})$. Therefore, by Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$ and $n \cong_{\psi} \widehat{n}$.

Given $(\gamma, \sigma_1, \text{acc}, \text{private int } res_{\text{acc}+1} = n) \Downarrow_{d_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.64 we have $\gamma_1 = \gamma :: \gamma_A$ such that $\gamma_A = [res_\text{acc} \rightarrow (\text{private int}, l_{res})]$ and $\sigma_2 = \sigma_1 :: \sigma_A$ such that $\sigma_A = [l_{res} \rightarrow (\text{EncodeVal}(\text{private int}, n), \text{private int}, 1, \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$. By Lemma 3.2.65, we have $(\gamma_1, \sigma_2) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$ and $(\gamma :: \gamma_A, \sigma_1 :: \sigma_A) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$. Given $\text{pfree}(e) \notin \text{private int } res_{\text{acc}+1} = n$, by Definition 3.2.11 ψ is not updated over the evaluation $(\gamma, \sigma_1, \text{acc}, \text{private int } res_{\text{acc}+1} = n) \Downarrow_{d_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$.

Given $\text{Extract_variables}(s_1, s_2) = x_{list_\text{acc}+1}$, by Lemma 3.2.67 we have $\forall x \notin x_{list_\text{acc}+1}, x = e \notin \{s_1; s_2\}$ and by Lemma 3.2.66 no other modifications to memory can occur within the evaluation of s_1 or s_2.

Given $\text{InitializeVariables}(x_{list_\text{acc}+1}, \gamma_1, \sigma_2, \text{acc}+1) = (\gamma_2, \sigma_3)$, $\gamma_1 = \gamma :: \gamma_A$, and $\sigma_2 = \sigma_1 :: \sigma_A$, by Lemma 3.2.70 we have $\gamma_2 = \gamma :: \gamma_A :: \gamma_B$ and $\sigma_3 = \sigma_1 :: \sigma_A :: \sigma_B$. Given $x_{list_\text{acc}+1}$, by Lemma 3.2.69, we have $\forall x \in x_{list_\text{acc}+1}, \{x_{then_\text{acc}}, x_{else_\text{acc}}\} \in \gamma 2$. By Lemma 3.2.71, we have $(\gamma_2, \sigma_3) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$ and $(\gamma :: \gamma_A :: \gamma_B, \sigma_1 :: \sigma_A :: \sigma_B) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$.

Given $(\gamma_2, \sigma_3, \text{acc}+1, s_1) \Downarrow_{d_3} (\gamma_3, \sigma_4, \text{acc}+1, \text{skip})$, $\gamma_2 = \gamma :: \gamma_A :: \gamma_B$, and $\sigma_3 = \sigma_1 :: \sigma_A :: \sigma_B$, $(\gamma :: \gamma_A :: \gamma_B, \sigma_1 :: \sigma_A :: \sigma_B) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$, and $s_1 \cong \widehat{s}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{s}_1)$ such that $(\gamma_2, \sigma_3, \text{acc}+1, s_1) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{s}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{s}_1) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma}_1, \widehat{\sigma}_2, \Box, \text{skip})$ and ψ_2 such that $(\gamma_3, \sigma_4, \text{acc}+1, \text{skip}) \cong_{\psi_2} (\widehat{\gamma}_1, \widehat{\sigma}_2, \Box, \text{skip})$. By Lemma 3.2.72, we have $\text{pfree}(e) \notin s_1$ and $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma_3, \sigma_4) \cong_{\psi} (\widehat{\gamma}_1, \widehat{\sigma}_2)$. By Lemma 3.2.68, we have $\gamma_3 = \gamma'_3 :: \gamma_A :: \gamma_B$ and $\sigma_4 = \sigma'_4 :: \sigma_A :: \sigma_B$. By Lemma 3.2.73 we have $(\gamma'_3, \sigma'_4) \cong_{\psi} (\widehat{\gamma}_1, \widehat{\sigma}_2)$.

By Lemma 3.2.74 we have that $\forall x \in x_{list_\text{acc}+1}$, the corresponding temporary variable x_{else_acc} maintains the original value for x from the starting memory σ_1, and the only differences between memory σ_1 and σ'_4 that can occur are stored at the memory blocks with identifiers l such that $\gamma(x) = (l, ty)$.

Given $\text{RestoreVariables}(x_{list_\text{acc}+1}, \gamma_3, \sigma_4, \text{acc}+1) = \sigma_5$, $\gamma_3 = \gamma'_3 :: \gamma_A :: \gamma_B$, $\sigma_4 = \sigma'_4 :: \sigma_A :: \sigma_B$, by Lemma 3.2.75 we have $\sigma_5 = \sigma_1 :: \sigma_A :: \sigma_C$ such that $\forall x \in x_{list_\text{acc}+1}, x_{then_\text{acc}}$ remains unchanged in σ_C, x_{then_acc} is updated with the modified values for x from the execution of the **then** branch, and x is updated to its original value from σ. By Definition 3.2.15, we have $(\gamma_3, \sigma_5) \cong_{\psi} (\widehat{\gamma}_1, \widehat{\sigma}_1)$. By Lemma 3.2.17, we have $(\gamma, \sigma_5) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$. By

Definition 3.2.15, we have $(\gamma :: \gamma_A :: \gamma_B, \sigma_5) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and given $\gamma_2 = \gamma :: \gamma_A :: \gamma_B$ we have $(\gamma_2, \sigma_5) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$.

Given $(\gamma_2,\ \sigma_5,\ \text{acc}+1,\ s_2) \Downarrow_{d_4} (\gamma_4,\ \sigma_6, \text{acc}+1, \text{skip})$, $s_2 \cong \widehat{s}_2$, and $(\gamma_2, \sigma_5) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, by Lemma 3.2.4 we have $(\gamma_2,\ \sigma_5, \text{acc}+1, s_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{s}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{s}_2) \Downarrow'_{\widehat{d}_4} (\widehat{\gamma}_2,\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ and ψ_3 such that $(\gamma_4,\ \sigma_6,\ \text{acc}+1,\ \text{skip}) \cong_{\psi_3} (\widehat{\gamma}_2,\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. By Lemma 3.2.72, we have $\text{pfree}(e) \notin s_2$ and $\psi_3 = \psi$. By Definition 3.2.20 we have $(\gamma_4,\ \sigma_6) \cong_\psi (\widehat{\gamma}_2,\ \widehat{\sigma}_3)$. By Lemma 3.2.68, we have $\gamma_4 = \gamma'_4 :: \gamma_A :: \gamma_B$ and $\sigma_6 = \sigma'_6 :: \sigma_A :: \sigma_C$. By Lemma 3.2.73 we have $(\gamma'_4, \sigma'_6) \cong_\psi (\widehat{\gamma}_2, \widehat{\sigma}_3)$. By Lemma 3.2.17, we have $(\gamma, \sigma'_6) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Subcase $n \neq_{\text{private}} \text{encrypt}(0)$

Given $n \cong_\psi \widehat{n}$ and $n \neq_{\text{private}} \text{encrypt}(0)$, we have $\widehat{n} \neq 0$.

Given $\text{ResolveVariables}(x_{list_\text{acc}+1}, \gamma_4,\ \sigma_6, \text{acc}+1, res_{\text{acc}+1}) = \sigma_7$, by Lemma 3.2.77 we have $\sigma_7 = \sigma'_4 :: \sigma_A :: \sigma_C$ and $(\gamma, \sigma_7) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$, $\widehat{n} \neq 0$, and $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{s}_1) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \Downarrow'_{\widehat{iet}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule If Else True.

Given $(\gamma,\ \sigma_7) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma_7, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iep} (\gamma,\ \sigma_7, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \Downarrow'_{\widehat{iet}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $iep \cong \widehat{iet}$ by Definition 3.2.21.

Subcase $n =_{\text{private}} \text{encrypt}(0)$

Given $n \cong_\psi \widehat{n}$ and $n =_{\text{private}} \text{encrypt}(0)$, we have $\widehat{n} = 0$.

Given $\text{ResolveVariables}(x_{list_\text{acc}+1}, \gamma_4,\ \sigma_6, \text{acc}+1, res_{\text{acc}+1}) = \sigma_7$, by Lemma 3.2.77 we have $\sigma_7 = \sigma'_6 :: \sigma_A :: \sigma_C$ and $(\gamma, \sigma_7) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$, $\widehat{n} = 0$, and $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{s}_2) \Downarrow'_{\widehat{d}_4} (\widehat{\gamma}_2,\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \Downarrow'_{\widehat{ief}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule If Else True.

Given $(\gamma,\ \sigma_7) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma, \sigma_7, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have

$(\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iep} (\gamma,\ \sigma_7, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \Downarrow'_{\widehat{ief}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip}), \Pi \cong_\psi \Sigma$, and $iep \cong \widehat{ief}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iet} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iet} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule If Else True, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, $n \neq 0$, and $(\gamma, \sigma_1, \text{acc}, s_1) \Downarrow_{d_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{if } (\widehat{e})\widehat{s}_1 \text{ else } \widehat{s}_2)$ and ψ such that $(\gamma,\ \sigma,\ \text{acc},\ \text{if } (e)\ s_1 \text{ else } s_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{if } (\widehat{e})\widehat{s}_1 \text{ else } \widehat{s}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{if } (e)\ s_1 \text{ else } s_2 \cong_\psi \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2$. Given $(\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iet} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{if } (e)\ s_1 \text{ else } s_2$. Therefore, by Lemma 3.2.3 we have $\text{if } (e)\ s_1 \text{ else } s_2 \cong \text{if } (\widehat{e})\widehat{s}_1 \text{ else } \widehat{s}_2$. By Definition 3.2.10, we have $\text{Erase}(\text{if } (e)\ s_1 \text{ else } s_2) = \text{if } (\text{Erase}(e)$ $\text{Erase}(s_1) \text{ else } \text{Erase}(s_2)$, $\text{Erase}(e) = \widehat{e}$, $\text{Erase}(s_1) = \widehat{s}_1$, and $\text{Erase}(s_2) = \widehat{s}_2$. Therefore, we have $e \cong \widehat{e}$, $s_1 \cong \widehat{s}_1$, and $s_2 \cong \widehat{s}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, ψ, $(l, \mu) \notin \text{if } (e)\ s_1 \text{ else } s_2$, and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$ and $d_1 \cong \widehat{d}_1$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1,\ \text{acc},\ n) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_\psi \widehat{n}$. By Definition 3.2.17 we have $n \cong \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$.

Given $n \neq 0$ and $n = \widehat{n}$, we have $\widehat{n} \neq 0$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, ψ, $(l, \mu) \notin \text{if } (e)\ s_1 \text{ else } s_2$, and $s_1 \cong \widehat{s}_1$, by Lemma 3.2.4 we have $(\gamma,\ \sigma_1,\ \text{acc},\ s_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{s}_1)$. Given $(\gamma,\ \sigma_1,\ \text{acc},\ s_1) \Downarrow_{d_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{s}_1) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2, \square, \text{skip})$ and ψ_2 such that $(\gamma_1, \sigma_2, \text{acc}, \text{skip}) \cong_{\psi_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2, \square, \text{skip})$ and $d_2 \cong \widehat{d}_2$. By Definition 3.2.20, we have $(\gamma_1, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$, $\widehat{n} \neq 0$, and $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{s}_1) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \Downarrow'_{\widehat{iet}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule If Else True.

Given $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iet} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \Downarrow'_{\widehat{iet}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_{\psi_2} \Sigma$,

and $iet \cong \widehat{iet}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{ief} (\gamma, \sigma_2, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iet} (\gamma, \sigma_2, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \&x) \Downarrow_{loc} (\gamma, \sigma, \text{acc}, (l, 0))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \&x) \Downarrow_{loc} (\gamma, \sigma, \text{acc}, (l, 0))$ by SMC^2 rule Address Of, we have $\gamma(x) = (l,\ ty)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \&\widehat{x})$ and ψ such that $(\gamma, \sigma, \text{acc}, \&x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \&\widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\&x \cong_\psi \&\widehat{x}$. By Definition 3.2.18 we have $\text{Erase}(\&x) = \&\ \text{Erase}(x)$ and $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l,\ ty)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{ty})$ such that $l = \widehat{l}$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, and $ty \cong \widehat{ty}$ by Lemma 3.2.14.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square, \&\widehat{x})$ and $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{ty})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \&\widehat{x}) \Downarrow'_{\widehat{loc}} (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{l}, 0))$ by Vanilla C rule Address Of.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$, by Definition 3.2.20 we have $(\gamma, \sigma, \text{acc}, (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{l}, 0))$. Therefore, we have $(\gamma, \sigma, \text{acc}, \&x) \Downarrow_{loc} (\gamma, \sigma, \text{acc}, (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \&\widehat{x}) \Downarrow'_{\widehat{loc}} (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{l}, 0))$, $\Pi \cong_\psi \Sigma$, and $loc \cong \widehat{loc}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{sizeof}(ty)) \Downarrow_{ty} (\gamma, \sigma, \text{acc}, n)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{sizeof}(ty)) \Downarrow_{ty} (\gamma, \sigma, \text{acc}, n)$ by SMC^2 rule Size of Type, we have $n = \tau(ty)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{sizeof}(\widehat{ty}))$ and ψ such that $(\gamma,\ \sigma,\ \text{acc}, \text{sizeof}(ty)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{sizeof}(\widehat{ty}))$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{sizeof}(ty) \cong_\psi \text{sizeof}(\widehat{ty})$. By Definition 3.2.18 we have $\text{Erase}(\text{sizeof}(ty)) = \text{sizeof}(\text{Erase}(ty))$ and $\text{Erase}(ty) = \widehat{ty}$. Therefore, we have $ty \cong \widehat{ty}$.

Given $n = \tau(ty)$, $ty \cong \widehat{ty}$ and $\text{Label}(ty, \gamma) = \text{public}$, we have $\widehat{n} = \tau(\widehat{ty})$ and $n = \widehat{n}$ by Lemma 3.2.48. By Definition 3.2.17 we have $n \cong_\psi \widehat{n}$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{sizeof}(\widehat{ty}))$ and $\widehat{n} = \tau(\widehat{ty})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{sizeof}(\widehat{ty})) \Downarrow'_{\widehat{ty}} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{n})$ by Vanilla C rule Size of Type.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $n \cong_{\psi} \widehat{n}$, by Definition 3.2.20 we have $(\gamma, \sigma, \text{acc}, n) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{n})$. Therefore, we have $(\gamma, \sigma, \text{acc}, \text{sizeof}(ty)) \Downarrow_{ty} (\gamma, \sigma, \text{acc}, n) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{sizeof}(\widehat{ty})) \Downarrow'_{\widehat{ty}} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{n})$, $\Pi \cong_{\psi} \Sigma$ and $ty \cong \widehat{ty}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow_{wle} (\gamma, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow_{wle} (\gamma, \sigma_1, \text{acc}, \text{skip})$ by SMC^2 rule While End, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma,\ \sigma_1, \text{acc}, n)$, $\text{Label}(e, \gamma) = \text{public}$, and $n\ =\ 0$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{while } (\widehat{e})\ \widehat{s})$ and ψ such that $(\gamma,\ \sigma,\ \text{acc}, \text{while } (e)\ s) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{while } (\widehat{e})\ \widehat{s})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{while } (e)\ s \cong_{\psi} \text{while } (\widehat{e})\ \widehat{s}$. Given $(\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow_{wle} (\gamma, \sigma_1, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{while } (e)\ s$. Therefore, by Lemma 3.2.3 we have $\text{while } (e)\ s \cong \text{while } (\widehat{e})\ \widehat{s}$. By Definition 3.2.10 we have $\text{Erase}(\text{while } (e)\ \widehat{s}) = \text{while } (\text{Erase}(e))\ \text{Erase}(s)$, $\text{Erase}(e) = \widehat{e}$, and $\text{Erase}(s) = \widehat{s}$. Therefore, we have $e \cong \widehat{e}$ and $s \cong \widehat{s}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, $e \cong \widehat{e}$, $(l, \mu) \notin \text{while } (e)\ s$, and ψ, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_{\psi} (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$ such that $(\gamma, \sigma_1, \text{acc}, n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$ and $d_1 \cong \widehat{d_1}$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1,\ \text{acc},\ n) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_{\psi} \widehat{n}$. By Definition 3.2.17 we have $n \cong \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$.

Given $n\ =\ 0$ and $n = \widehat{n}$, we have $\widehat{n}\ =\ 0$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{while } (\widehat{e})\ \widehat{s})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$, and $\widehat{n}\ =\ 0$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{while } (\widehat{e})\ \widehat{s}) \Downarrow'_{\widehat{wle}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \text{skip})$ by Vanilla C rule While End.

Given $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_1, \text{acc}, \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, \text{while } (e)\ s) \Downarrow_{wle} (\gamma, \sigma_1, \text{acc}, \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{while } (\widehat{e})\ \widehat{s}) \Downarrow'_{\widehat{wle}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \text{skip})$, $\Pi \cong_{\psi} \Sigma$, and $wle \cong \widehat{wle}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow_{wlc} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ \text{while } (e)\ s) \Downarrow_{wlc} (\gamma,\ \sigma_3, \text{acc},\ \text{skip})$ by SMC^2 rule While Continue, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, $n \neq 0$, $(\gamma, \sigma_1, \text{acc}, s) \Downarrow_{d_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$, and $(\gamma_1, \sigma_2, \text{acc}, \text{while } (e)\ s) \Downarrow_{d_3} (\gamma_2, \sigma_3, \text{acc}, \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \text{while } (\widehat{e})\ \widehat{s})$ and ψ such that $(\gamma,\ \sigma,\ \text{acc}, \text{while } (e)\ s) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \Box, \text{while } (\widehat{e})\ \widehat{s})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{while } (e)\ s \cong_\psi \text{while } (\widehat{e})\ \widehat{s}$. Given $(\gamma,\ \sigma,\ \text{acc}, \text{while } (e)\ s) \Downarrow_{wlc} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{while } (e)\ s$. Therefore, by Lemma 3.2.3 we have $\text{while } (e)\ s \cong \text{while } (\widehat{e})\ \widehat{s}$. By Definition 3.2.10 we have $\text{Erase}(\text{while } (e)\ \widehat{s}) = \text{while } (\text{Erase}(e))\ \text{Erase}(s)$, $\text{Erase}(e) = \widehat{e}$, and $\text{Erase}(s) = \widehat{s}$. Therefore, we have $e \cong_\psi \widehat{e}$ and $s \cong_\psi \widehat{s}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, $e \cong \widehat{e}$, $(l, \mu) \notin \text{while } (e)\ s$, and ψ, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{n})$ such that $(\gamma, \sigma_1, \text{acc}, n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{n})$ and $d_1 \cong \widehat{d_1}$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1,\ \text{acc},\ n) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{n})$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_\psi \widehat{n}$. By Definition 3.2.17 we have $n \cong \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$.

Given $n \neq 0$ and $n = \widehat{n}$, we have $\widehat{n} \neq 0$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, $s \cong \widehat{s}$, $(l, \mu) \notin \text{while } (e)\ s$, and ψ, by Lemma 3.2.4 we have $(\gamma,\ \sigma_1,\ \text{acc},\ s) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{s})$. Given $(\gamma,\ \sigma_1,\ \text{acc},\ s) \Downarrow_{d_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{s}) \Downarrow'_{\widehat{d_2}} (\widehat{\gamma}_1,\ \widehat{\sigma}_2, \Box, \text{skip})$ such that $(\gamma_1, \sigma_2, \text{acc}, \text{skip}) \cong_{\psi_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2, \Box, \text{skip})$ and $d_2 \cong \widehat{d_2}$. By Definition 3.2.20, we have $(\gamma_1, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$. By Lemma 3.2.17, we have $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$, $\text{while } (e)\ s \cong \text{while } (\widehat{e})\ \widehat{s}$, and $(l, \mu) \notin \text{while } (e)\ s$, by Lemma 3.2.4 we have $(\gamma,\ \sigma_2, \text{acc}, \text{while } (e)\ s) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box, \text{while } (\widehat{e})\ \widehat{s})$. Given $(\gamma,\ \sigma_2, \text{acc}, \text{while } (e)\ s) \Downarrow_{d_3} (\gamma_2, \sigma_3, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}_2, \Box, \text{while } (\widehat{e})\ \widehat{s}) \Downarrow'_{\widehat{d_3}} (\widehat{\gamma}_2, \widehat{\sigma}_3, \Box, \text{skip})$ such that $(\gamma_2, \sigma_3, \text{acc}, \text{skip}) \cong_{\psi_3} (\widehat{\gamma}_2,\ \widehat{\sigma}_3, \Box, \text{skip})$ and $d_3 \cong \widehat{d_3}$. By Definition 3.2.20, we have $(\gamma_2, \sigma_3) \cong_{\psi_3} (\widehat{\gamma}_2,\ \widehat{\sigma}_3)$. By Lemma 3.2.17, we have $(\gamma,\ \sigma_3) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \text{while } (\widehat{e})\ \widehat{s})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{n})$, $\widehat{n} \neq 0$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{s}) \Downarrow'_{\widehat{d_2}} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \Box,\ \text{skip})$, and $(\widehat{\gamma}, \widehat{\sigma}_2, \Box, \text{while } (e)\ s) \Downarrow'_{\widehat{d_3}} (\widehat{\gamma}_2, \widehat{\sigma}_3, \Box, \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \Box, \text{while } (\widehat{e})\ \widehat{s}) \Downarrow'_{\widehat{wlc}} (\widehat{\gamma},\ \widehat{\sigma}_3, \Box, \text{skip})$ by Vanilla C rule While Continue.

Given $(\gamma,\ \sigma_3) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, \text{while } (e)\ s) \Downarrow_{wlc} (\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{while } (\widehat{e})\ \widehat{s}) \Downarrow'_{\widehat{wlc}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, \text{skip})$, $\Pi \cong_{\psi_3} \Sigma$, and $wlc \cong \widehat{wlc}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, s_1;s_2) \Downarrow_{ss} (\gamma_2, \sigma_2, \text{acc},\ v)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ s_1;s_2) \Downarrow_{ss} (\gamma_2, \sigma_2, \text{acc},\ v)$ by SMC^2 rule Statement Sequencing, we have $(\gamma,\ \sigma,\ \text{acc},\ s_1) \Downarrow_{d_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ and $(\gamma_1, \sigma_1, \text{acc},\ s_2) \Downarrow_{d_2} (\gamma_2,\ \sigma_2, \text{acc}, v)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{s}_1;\ \widehat{s}_2)$ and ψ such that $(\gamma,\ \sigma,\ \text{acc},\ s_1;s_2) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{s}_1;\ \widehat{s}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $s_1;s_2 \cong_{\psi} \widehat{s}_1;\ \widehat{s}_2$. Given $(\gamma,\ \sigma,\ \text{acc},\ s_1;s_2) \Downarrow_{ss} (\gamma_2, \sigma_2, \text{acc},\ v)$, by Lemma 3.2.2 we have $(l, \mu) \notin s_1;s_2$. Therefore, by Lemma 3.2.3 we have $s_1;s_2 \cong \widehat{s}_1;\ \widehat{s}_2$. By Definition 3.2.10 we have $\text{Erase}(s_1;s_2) = \text{Erase}(s_1); \text{Erase}(s_2)$, $\text{Erase}(s_1) = \widehat{s}_1$, and $\text{Erase}(s_2) = \widehat{s}_2$. Therefore, we have $s_1 \cong_{\psi} \widehat{s}_1$ and $s_2 \cong \widehat{s}_2$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, $s_1 \cong \widehat{s}_1$, and $(l, \mu) \notin s_1;s_2$, by Lemma 3.2.4 we have $(\gamma,\ \sigma,\ \text{acc},\ s_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{s}_1)$. Given $(\gamma, \sigma, \text{acc}, s_1) \Downarrow_{d_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{s}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \text{skip})$ and ψ_1 such that $(\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \text{skip})$ and $d_1 \cong \widehat{d}_1$. By Definition 3.2.20, we have $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, $s_2 \cong \widehat{s}_2$ and $(l, \mu) \notin s_1;s_2$, by Lemma 3.2.4 we have we have $(\gamma_1, \sigma_1, \text{acc}, s_2) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square, \widehat{s}_2)$. Given $(\gamma_1, \sigma_1, \text{acc},\ s_2) \Downarrow_{d_2} (\gamma_2, \sigma_2, \text{acc},\ v)$, by the inductive hypothesis, we have $(\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \widehat{s}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2, \square, \widehat{v})$ such that $(\gamma_2, \sigma_2, \text{acc},\ v) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2, \square, \widehat{v})$ and $d_2 \cong \widehat{d}_2$. By Definition 3.2.20, we have $(\gamma_2, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2)$ and $v \cong_{\psi_2} \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{s}_1;\ \widehat{s}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$, and $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{s}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \widehat{s}_1; \widehat{s}_2) \Downarrow'_{\widehat{ss}} (\widehat{\gamma}_2,\ \widehat{\sigma}_2, \square, \widehat{v})$ by Vanilla C rule Statement Sequencing.

Given $(\gamma_2, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2)$ and $v \cong_{\psi_2} \widehat{v}$, by Definition 3.2.20 we have $(\gamma_2, \sigma_2, \text{acc},\ v) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2, \square, \widehat{v})$. Therefore, we have $(\gamma,\ \sigma, \text{acc}, s_1;\ s_2) \Downarrow_{ss} (\gamma_2, \sigma_2, \text{acc},\ v) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{s}_1;\ \widehat{s}_2) \Downarrow'_{\widehat{ss}} (\widehat{\gamma}_2,\ \widehat{\sigma}_2, \square, \widehat{v})$, $\Pi \cong_{\psi_2} \Sigma$, and $ss \cong \widehat{ss}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, (e)) \Downarrow_{ep} (\gamma, \sigma_1, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, (e)) \Downarrow_{ep} (\gamma, \sigma_1, \text{acc}, v)$ by SMC^2 rule Parentheses, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, v)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \Box, (\widehat{e}))$ and ψ such that $(\gamma, \sigma, \text{acc}, (e)) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}, \Box, (\widehat{e}))$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})$ and $(e) \cong_{\psi} (\widehat{e})$. Given $(\gamma, \sigma, \text{acc}, (e)) \Downarrow_{ep} (\gamma, \sigma_1, \text{acc}, v)$, by Lemma 3.2.2 we have $(l, \mu) \notin (e)$. Therefore, by Lemma 3.2.3 we have $(e) \cong (\widehat{e})$. By Definition 3.2.10 we have $\text{Erase}((e)) = (\text{Erase}(e))$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma, \sigma) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})$, $(l, \mu) \notin (e)$, and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{e})$ such that $(\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{e}) \cong_{\psi} (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{v})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, v) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{v})$ and $d_1 \cong \widehat{d_1}$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$ and $v \cong_{\psi_1} \widehat{v}$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \Box, (\widehat{e}))$ and $(\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{v})$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \Box, (\widehat{e})) \Downarrow'_{\widehat{ep}} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{v})$ by Vanilla C rule Parentheses.

Given $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$ and $v \cong_{\psi_1} \widehat{v}$, by Definition 3.2.20 we have $(\gamma, \sigma_1, \text{acc}, v) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{v})$. Therefore, we have $(\gamma, \sigma, \text{acc}, (e)) \Downarrow_{ep} (\gamma, \sigma_1, \text{acc}, v) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}, \Box, (\widehat{e})) \Downarrow'_{\widehat{ep}} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{v})$, $\Pi \cong_{\psi_1} \Sigma$, and $ep \cong \widehat{ep}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \{s\}) \Downarrow_{sb} (\gamma, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \{s\}) \Downarrow_{sb} (\gamma, \sigma_1, \text{acc}, \text{skip})$ by SMC^2 rule Statement Block, we have $(\gamma, \sigma, \text{acc}, s) \Downarrow_{d_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \Box, \{\widehat{s}\})$ and ψ such that $(\gamma, \sigma, \text{acc}, \{s\}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}, \Box, \{\widehat{s}\})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})$ and $\{s\} \cong_{\psi} \{\widehat{s}\}$. Given $(\gamma, \sigma, \text{acc}, \{s\}) \Downarrow_{sb} (\gamma, \sigma_1, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \{s\}$. Therefore, by Lemma 3.2.3 we have $\{s\} \cong \{\widehat{s}\}$. By Definition 3.2.10 we have $\text{Erase}(\{s\}) = \{\text{Erase}(s)\}$ and $\text{Erase}(s) = \widehat{s}$. Therefore, we have $s \cong \widehat{s}$.

Given $(\gamma, \sigma) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})$, $(l, \mu) \notin \{s\}$, and $s \cong \widehat{s}$, by Lemma 3.2.4 we have $(\gamma, \sigma, \text{acc}, s) \cong (\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{s})$. Given $(\gamma, \sigma, \text{acc}, s) \Downarrow_{d_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{s}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \text{skip})$ and ψ_1 such that $(\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_{\psi_1} (\widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \text{skip})$ and $d_1 \cong \widehat{d_1}$. By Definition 3.2.20, we have $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1, \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \{\widehat{s}\})$ and $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \{\widehat{s}\}) \Downarrow'_{\widehat{sb}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \text{skip})$ by Vanilla C rule Statement Block.

Given $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_1,\ \text{acc},\ \text{skip}) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc},\ \{s\}) \Downarrow_{sb} (\gamma,\ \sigma_1,\ \text{acc},\ \text{skip}) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \{\widehat{s}\}) \Downarrow'_{\widehat{sb}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \text{skip})$, $\Pi \cong_{\psi_1} \Sigma$, and $sb \cong \widehat{sb}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl} (\gamma,\ \sigma_3, \text{acc},\ (l, 0))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl} (\gamma,\ \sigma_3, \text{acc},\ (l, 0))$ by SMC^2 rule Cast Public Location, we have $(\gamma,\ \sigma,\ \text{acc},\ e) \Downarrow_{d_1} (\gamma,\ \sigma_1,\ \text{acc},\ (l, 0))$, $(ty = \text{public}\ bty*) \vee (ty = \text{char}*)$, $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega,\ \text{void}, n,\ \text{PermL}(\text{Freeable},\ \text{void},\ \text{public}, n)\big)\big]$, and $\sigma_3 = \sigma_2\big[l \rightarrow \big(\omega,\ ty,\ \frac{n}{\tau(ty)},\ \text{PermL}\big(\text{Freeable},\ ty,\ \text{public},\ \frac{n}{\tau(ty)}\big)\big)\big]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \text{acc},\ (ty)\ e) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $(ty)\ e \cong_{\psi} (\widehat{ty})\ \widehat{e}$. Given $(\gamma,\ \sigma,\ \text{acc},\ (ty)\ e) \Downarrow_{cl} (\gamma,\ \sigma_3, \text{acc},\ (l, 0))$, by Lemma 3.2.2 we have $(l, \mu) \notin (ty)\ e$. Therefore, by Lemma 3.2.3 we have $(ty)\ e \cong (\widehat{ty})\ \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}((ty)\ e) = (\text{Erase}(ty))\ \text{Erase}(e)$, $\text{Erase}(ty) = \widehat{ty}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, $(l, \mu) \notin (ty)\ e$, and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_{\psi} (\gamma, \sigma, \text{acc}, e)$ by Definition 3.2.20. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, (l, 0))$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}, 0))$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, (l, 0)) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}, 0))$ and $d_1 \cong \widehat{d_1}$. Given $(l, 0) \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l, 0) \cong_{\psi} (\widehat{l}, 0)$. By Definition 3.2.13 we have $l \cong_{\psi} \widehat{l}$.

Given $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega,\ \text{void}, n,\ \text{PermL}(\text{Freeable},\ \text{void},\ \text{public}, n)\big)\big]$, $l \cong_{\psi} \widehat{l}$, and $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, by Lemma 3.2.36 we have $\widehat{\sigma}_1 = \widehat{\sigma}_2\big[\widehat{l} \rightarrow \big(\widehat{\omega}, \text{void},\ \widehat{n},\ \text{PermL}(\text{Freeable}, \text{void}, \text{public}, \widehat{n})\big)\big]$ such that $(\gamma,\ \sigma_2) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2)$, $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$, $ty \cong \widehat{ty}$, and $\omega \cong_{\psi} \widehat{\omega}$.

Given $\sigma_3 = \sigma_2\big[l \rightarrow \big(\omega,\ ty,\ \frac{n}{\tau(ty)},\ \text{PermL}\big(\text{Freeable},\ ty,\ \text{public},\ \frac{n}{\tau(ty)}\big)\big)\big]$, $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$, $l \cong_{\psi} \widehat{l}$, $ty \cong \widehat{ty}$, $(\gamma,\ \sigma_2) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $\omega \cong_{\psi} \widehat{\omega}$, by Lemma 3.2.35 we have $\widehat{\sigma}_3 = \widehat{\sigma}_2\big[\widehat{l} \rightarrow \big(\widehat{\omega},\ \widehat{ty},\ \frac{\widehat{n}}{\tau(\widehat{ty})},\ \text{PermL}\big(\text{Freeable},\ \widehat{ty}, \text{public},\ \frac{\widehat{n}}{\tau(\widehat{ty})}\big)\big)\big]$ such that $(\gamma,\ \sigma_3) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}), (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})\ \Downarrow'_{\widehat{d}_1}\ (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}, 0)), \widehat{\sigma}_1 = \widehat{\sigma}_2\big[\widehat{l} \rightarrow \big(\widehat{\omega}, \text{void},\ \widehat{n},$ $\text{PermL}(\text{Freeable}, \text{void}, \text{public}, \widehat{n})\big)\big]$, $(\widehat{ty} = \widehat{bty*})$, and $\widehat{\sigma}_3 = \widehat{\sigma}_2\big[\widehat{l} \rightarrow \big(\widehat{\omega},\ \widehat{ty},\ \frac{\widehat{n}}{\tau(\widehat{ty})},$ $\text{PermL}\big(\text{Freeable},\ \widehat{ty}, \text{public},\ \frac{\widehat{n}}{\tau(\widehat{ty})}\big)\big)\big]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}) \Downarrow'_{\widehat{cl}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, (\widehat{l}, 0))$ by Vanilla C rule Cast Location.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_3, \text{acc}, (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, (\widehat{l}, 0))$. Therefore, we have $(\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl} (\gamma,\ \sigma_3, \text{acc},\ (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}) \Downarrow'_{\widehat{cl}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, (\widehat{l}, 0))$, $\Pi \cong_\psi \Sigma$, and $cl1 \cong \widehat{cl}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma,\ \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl1} (\gamma,\ \sigma_3, \text{acc},\ (l, 0))$

Given $\Pi \triangleright (\gamma,\ \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl1} (\gamma,\ \sigma_3, \text{acc},\ (l, 0))$ by SMC^2 rule Case Private Location, we have $(\gamma, \sigma, \text{acc},$ $e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, (l, 0))$, $(ty = \text{private int}*) \vee (ty = \text{private float}*) \vee (ty = \text{int}*) \vee (ty = \text{float}*)$, $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega, \text{void}, n, \text{PermL}(\text{Freeable}, \text{void}, \text{private}, n)\big)\big]$, and $\sigma_3 = \sigma_2\big[l \rightarrow \big(\omega, ty, \frac{n}{\tau(ty)}, \text{PermL}\big(\text{Freeable},$ $ty, \text{private}, \frac{n}{\tau(ty)}\big)\big)\big]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e})$ and ψ such that $(\gamma, \sigma, \text{acc}, (ty)\ e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $(ty)\ e \cong_\psi (\widehat{ty})\ \widehat{e}$. Given $(\gamma,\ \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl1} (\gamma,\ \sigma_3, \text{acc},\ (l, 0))$, by Lemma 3.2.2 we have $(l, \mu) \notin (ty)\ e$. Therefore, by Lemma 3.2.3 we have $(ty)\ e \cong (\widehat{ty})\ \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}((ty)\ e) = (\text{Erase}(ty))\ \text{Erase}(e)$, $\text{Erase}(ty) = \widehat{ty}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, $(l, \mu) \notin (ty)\ e$, and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square,$ $\widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, (l, 0))$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square,$ $\widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square, (\widehat{l}, 0))$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, (l, 0)) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}, 0))$ and $d_1 \cong \widehat{d}_1$. Given $(l, 0) \neq \text{skip}$, by Lemma 3.2.1 we have $(\gamma, \sigma_1, \text{acc}, (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}, 0))$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$.

Given $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega, \text{void}*, n, \text{PermL}(\text{Freeable}, ty, \text{private}, n)\big)\big]$, $l \cong_\psi \widehat{l}$, $ty \cong \widehat{ty}$, and $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, by Lemma 3.2.36 we have $\widehat{\sigma}_1 = \widehat{\sigma}_2\big[\widehat{l} \rightarrow \big(\widehat{\omega},\ \text{void}*,\ \widehat{n}, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, \widehat{n})\big)\big]$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},$ $\widehat{\sigma}_2)$, $\omega \cong_\psi \widehat{\omega}$, and $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$.

Given $\sigma_3 = \sigma_2\big[l \rightarrow \big(\omega,\ ty,\ \frac{n}{\tau(ty)}, \text{PermL}\big(\text{Freeable},\ ty,\ \text{private}, \frac{n}{\tau(ty)}\big)\big)\big]$, $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$, $l \cong_\psi \widehat{l}$, $ty \cong \widehat{ty}$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.35 we have $\widehat{\sigma}_3 = \widehat{\sigma}_2\big[\widehat{l} \rightarrow \big(\widehat{\omega},\ \widehat{ty},\ \frac{\widehat{n}}{\tau(\widehat{ty})}, \text{PermL}\big(\text{Freeable},$

$\widehat{ty}, \text{public}, \frac{\widehat{n}}{\tau(\widehat{ty})}))]$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}), (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})\ \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}, 0)), \widehat{\sigma}_1 = \widehat{\sigma}_2[\widehat{l} \to (\widehat{\omega},\ \text{void},\ \widehat{n}, \text{PermL}(\text{Freeable}, \text{void}, \text{public}, \widehat{n}))]$, $(\widehat{ty} = \widehat{bty*})$, and $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l} \to (\widehat{\omega}, \widehat{ty}, \frac{\widehat{n}}{\tau(\widehat{ty})}, \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, \frac{\widehat{n}}{\tau(\widehat{ty})}))]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}) \Downarrow'_{\widehat{cl}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, (\widehat{l}, 0))$ by Vanilla C rule Cast Location.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_3, \text{acc},\ (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, (\widehat{l}, 0))$. Therefore, we have $(\gamma,\ \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl1} (\gamma,\ \sigma_3, \text{acc},\ (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}) \Downarrow'_{\widehat{cl}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, (\widehat{l}, 0))$, $\Pi \cong_\psi \Sigma$ and $cl \cong \widehat{cl}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv} (\gamma, \sigma_1, \text{acc}, n_1)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv} (\gamma, \sigma_1, \text{acc}, n_1)$ by SMC^2 rule Cast Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e)\ \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, $(ty = \text{public int}) \vee (ty = \text{public float})$, and $n_1 = \text{Cast}(\text{public}, ty, n)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \text{acc},\ (ty)\ e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $(ty)\ e \cong_\psi (\widehat{ty})\ \widehat{e}$. Given $(\gamma,\ \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv} (\gamma,\ \sigma_3, \text{acc},\ n_1)$, by Lemma 3.2.2 we have $(l, \mu) \notin (ty)\ e$. Therefore, by Lemma 3.2.3 we have $(ty)\ e \cong (\widehat{ty})\ \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}((ty)\ e) = (\text{Erase}(ty))\ \text{Erase}(e)$, $\text{Erase}(ty) = \widehat{ty}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, $(l, \mu) \notin (ty)\ e$, and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$ and $d_1 \cong \widehat{d}_1$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_\psi \widehat{n}$. By Definition 3.2.17 we have $n \cong \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.10.

Given $n_1 = \text{Cast}(\text{public}, ty, n)$, $ty \cong \widehat{ty}$, and $n = \widehat{n}$, by Lemma 3.2.24 we have $\widehat{n}_1 = \text{Cast}(\text{public},\ \widehat{ty},\ \widehat{n})$ such that $n_1 = \widehat{n}_1$. By Definition 3.2.10 we have $n_1 \cong \widehat{n}_1$, and by Definition 3.2.17 we have $n_1 \cong_\psi \widehat{n}_1$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}), (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})\ \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$, and $\widehat{n}_1 = \text{Cast}(\text{public},\ \widehat{ty},\ \widehat{n})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}) \Downarrow'_{\widehat{cv}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n}_1)$ by Vanilla C rule Cast Value.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$, by Definition 3.2.20 we have $(\gamma, \sigma_1, \text{acc}, n_1) \cong (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n}_1)$. Therefore,

we have $(\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv} (\gamma, \sigma_1, \text{acc}, n_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}) \Downarrow'_{\widehat{cv}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$, $\Pi \cong_\psi \Sigma$ and $cv \cong \widehat{cv}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv1} (\gamma, \sigma_1, \text{acc}, n_1)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv1} (\gamma, \sigma_1, \text{acc}, n_1)$ by SMC^2 rule Cast Private Value, we have $\text{Label}(e, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, $(ty = \text{private int}) \vee (ty = \text{private float}) \vee (ty = \text{int}) \vee (ty = \text{float})$, and $n_1 = \text{Cast}(\text{private}, ty, n)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e})$ such that $(\gamma, \sigma, \text{acc}, (ty)\ e) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $(ty)\ e \cong_\psi (\widehat{ty})\ \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv1} (\gamma, \sigma_3, \text{acc}, n_1)$, by Lemma 3.2.2 we have $(l, \mu) \notin (ty)\ e$. Therefore, by Lemma 3.2.3 we have $(ty)\ e \cong (\widehat{ty})\ \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}((ty)\ e) = (\text{Erase}(ty))\ \text{Erase}(e)$, $\text{Erase}(ty) = \widehat{ty}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, \mu) \notin (ty)\ e$, and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \cong (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, n) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n})$ and $d_1 \cong \widehat{d}_1$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $n \cong_\psi \widehat{n}$.

Given $n_1 = \text{Cast}(\text{private}, ty, n)$, $ty \cong \widehat{ty}$, and $n \cong_\psi \widehat{n}$, by Lemma 3.2.25 we have $\widehat{n}_1 = \text{Cast}(\text{public}, \widehat{ty}, \widehat{n})$ such that $n_1 \cong_\psi \widehat{n}_1$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e})$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n})$, and $\widehat{n}_1 = \text{Cast}(\text{public}, \widehat{ty}, \widehat{n})$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}) \Downarrow'_{\widehat{cv}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$ by Vanilla C rule Cast Value.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$, by Definition 3.2.20 we have $(\gamma, \sigma_1, \text{acc}, n_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$. Therefore, we have $(\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv1} (\gamma, \sigma_1, \text{acc}, n_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}) \Downarrow'_{\widehat{cv}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$, $\Pi \cong_\psi \Sigma$ and $cv1 \cong \widehat{cv}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow_{inp} (\gamma, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow_{inp} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by SMC^2 rule SMC Input Public Value, we have $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, $\text{acc} = 0$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$,

$\gamma(x) = (l, \text{public } bty)$, $\text{InputValue}(x, n) = n_1$, and $(\gamma,\ \sigma_2,\ \text{acc},\ x = n_1) \Downarrow_{d_3} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2))$ and ψ such that $(\gamma,\ \sigma,\ \text{acc}, \text{smcinput}(e_1,\ e_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2))$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{smcinput}(e_1, e_2) \cong_\psi \text{mcinput}(\widehat{e}_1, \widehat{e}_2)$. Given $(\gamma,\ \sigma,\ \text{acc},\ \text{smcinput}(e_1,e_2)) \Downarrow_{inp} (\gamma,\ \sigma_3,\ \text{acc},\ \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{smcinput}(e_1,e_2)$. Therefore, by Lemma 3.2.3 we have $\text{smcinput}(e_1,e_2) \cong \text{mcinput}(\widehat{e}_1, \widehat{e}_2)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcinput}(e_1,\ e_2)) = \text{mcinput}(\text{Erase}(e_1,\ e_2))$. By Definition 3.2.8, we have $\text{Erase}(e_1, e_2) = \text{Erase}(e_1), \text{Erase}(e_2))$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$ and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$, and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x})$ and $d_1 \cong \widehat{d}_1$. Given $x \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma\ \sigma_1, \text{acc},\ e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, n) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n})$ and $d_2 \cong \widehat{d}_2$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. By Definition 3.2.17 we have $n \cong \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.10.

Given $\gamma(x) = (l, \text{public } bty)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by $\text{public } bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\text{InputValue}(x, n) = n_1$, $x = \widehat{x}$, and $n = \widehat{n}$, by Axiom 3.2.4 and Lemma 3.2.26 we have $\text{InputValue}(\widehat{x},\ \widehat{n}) = \widehat{n}_1$ such that $n_1 \cong \widehat{n}_1$. By Definition 3.2.17 we have $n_1 \cong_\psi \widehat{n}_1$.

Given $x = \widehat{x}$ and $n_1 \cong \widehat{n}_1$, by Definition 3.2.10 we have $x = n_1 \cong \widehat{x} = \widehat{n}_1$, and by Definition 3.2.18 we have $x = n_1 \cong_\psi \widehat{x} = \widehat{n}_1$. Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, we have $(\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{x} = \widehat{n}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{x} = \widehat{n}_1) \cong_\psi (\gamma,\ \sigma_2, \text{acc},\ x = n_1)$ by Definition 3.2.20. Given $(\gamma, \sigma_2, \text{acc}, x = n_1) \Downarrow_{d_3} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{x} = \widehat{n}_1) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ and ψ_3 such that $(\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ and $d_3 \cong \widehat{d}_3$. Given $\text{pfree}(e) \notin x = n_1$, by Definition 3.2.11 we have $\psi_3 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)\ \Downarrow'_{\widehat{d}_1}\ (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)\ \Downarrow'_{\widehat{d}_2}\ (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty})$, $\text{InputValue}(\widehat{x},\ \widehat{n}) = \widehat{n}_1$, and $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{x} = \widehat{v})\ \Downarrow'_{\widehat{d}_3}\ (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2)) \Downarrow'_{\widehat{inp}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, \text{skip})$ by Vanilla C rule Input Value.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma, \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, \text{smcinput}(e_1,e_2)) \Downarrow_{inp} (\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2)) \Downarrow'_{\widehat{inp}} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$ and $inp \cong \widehat{inp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1,e_2)) \Downarrow_{inp3} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma,\ \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow_{inp3} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$ by SMC^2 rule Input Private Variable, we have $\text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, $(\gamma\ \sigma_1, \text{acc},\ e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$, $\gamma(x) = (l, \text{private}\ bty)$, $\text{InputValue}(x, n) = n_1$, and $(\gamma, \sigma_2, \text{acc}, x = n_1) \Downarrow_{d_3} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2))$ and ψ such that $(\gamma,\ \sigma,\ \text{acc}, \text{smcinput}(e_1, e_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2))$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{smcinput}(e_1, e_2) \cong_\psi \text{mcinput}(\widehat{e}_1, \widehat{e}_2)$. Given $(\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow_{inp3} (\gamma, \sigma_3, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{smcinput}(e_1,\ e_2)$. Therefore, by Lemma 3.2.3 we have $\text{smcinput}(e_1, e_2) \cong \text{mcinput}(\widehat{e}_1, \widehat{e}_2)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcinput}(e_1, e_2)) = \text{mcinput}(\text{Erase}(e_1, e_2))$. By Definition 3.2.8, we have $\text{Erase}(e_1, e_2) = \text{Erase}(e_1), \text{Erase}(e_2))$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$ and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$, and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x})$ and $d_1 \cong \widehat{d}_1$. Given $x \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma\ \sigma_1, \text{acc},\ e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, n) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n})$ and $d_2 \cong \widehat{d}_2$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.17 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{private } bty)$, $(\gamma,\ \sigma) \cong (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by $\text{private } bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\text{InputValue}(x, n) = n_1$, $x = \widehat{x}$, and $n = \widehat{n}$, by Axiom 3.2.4 and Lemma 3.2.26 we have $\text{InputValue}(\widehat{x},\ \widehat{n}) = \widehat{n}_1$ such that $n_1 \cong \widehat{n}_1$.

Given $x = \widehat{x}$ and $n_1 \cong \widehat{n}_1$, by Definition 3.2.10 we have $x = n_1 \cong \widehat{x} = \widehat{n}_1$. Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{x} = \widehat{n}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{x} = \widehat{n}_1) \cong_\psi (\gamma,\ \sigma_2, \text{acc},\ x = n_1)$. Given $(\gamma, \sigma_2, \text{acc}, x = n_1) \Downarrow_{d_3} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{x} = \widehat{n}_1) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ and ψ_3 such that $(\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ and $d_3 \cong \widehat{d}_3$. Given $\text{pfree}(e) \notin$, we have $\psi_3 = \psi_1$. By Definition 3.2.20 we have $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty})$, $\text{InputValue}(\widehat{x},\ \widehat{n}) = \widehat{n}_1$, and $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{x} = \widehat{v}) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2)) \Downarrow'_{\widehat{inp}} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, \text{skip})$ by Vanilla C rule Input Value.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow_{inp3} (\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2)) \Downarrow'_{\widehat{inp}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$ and $inp3 \cong \widehat{inp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp1} (\gamma,\ \sigma_4, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp1} (\gamma,\ \sigma_4, \text{acc}, \text{skip})$ by SMC^2 rule SMC Input Public 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma,\ \sigma,\ \text{acc},\ e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$, $(\gamma, \sigma_2, \text{acc}, e_3) \Downarrow_{d_3} (\gamma,\ \sigma_3, \text{acc},\ n_1)$, $\gamma(x) = (l, \text{public const } bty*)$, $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, and $(\gamma,\ \sigma_3, \text{acc},\ x = [m_0, ..., m_{n_1}]) \Downarrow_{d_4} (\gamma,\ \sigma_4, \text{acc}, \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2, e_3))$ and ψ such that $(\gamma,\ \sigma,\ \text{acc}, \text{smcinput}(e_1, e_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{smcinput}(e_1, e_2, e_3) \cong_\psi \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)$. Given $(\gamma,\ \sigma,\ \text{acc},\ \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp1} (\gamma,\ \sigma_4, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{smcinput}(e_1,\ e_2, e_3)$. Therefore, by Lemma 3.2.3 we have $\text{smcinput}(e_1,\ e_2,\ e_3) \cong \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcoutput}(e_1, e_2, e_3)) = \text{mcinput}(\text{Erase}(e_1, e_2, e_3))$. By Definition 3.2.8, we have $\text{Erase}(e_1,\ e_2, e_3) = \text{Erase}(e_1)$,

$\text{Erase}(e_2)$, $\text{Erase}(e_3)$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$, $\text{Erase}(e_2) = \widehat{e}_2$, and $\text{Erase}(e_3) = \widehat{e}_3$. Therefore, we have $e_1 \cong \widehat{e}_1$, $e_2 \cong \widehat{e}_2$, and $e_3 \cong \widehat{e}_3$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x})$ and $d_1 \cong \widehat{d}_1$. Given $x \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma\ \sigma_1, \text{acc},\ e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, n) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n})$ and $d_2 \cong \widehat{d}_2$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $e_3 \cong \widehat{e}_3$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{e}_3)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{e}_3) \cong_\psi (\gamma, \sigma_2, \text{acc}, e_3)$. Given $(\gamma, \sigma_2, \text{acc}, e_3) \Downarrow_{d_3} (\gamma,\ \sigma_3, \text{acc},\ n_1)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{e}_3) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \widehat{n}_1)$ and ψ_3 such that $(\gamma,\ \sigma_3, \text{acc},\ n_1) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \widehat{n}_1)$ and $d_3 \cong \widehat{d}_3$. Given $n_1 \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_3 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$ and $n_1 \cong_\psi \widehat{n}_1$. Given $\text{Label}(e_3, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$ and therefore $n_1 = \widehat{n}_1$ by Definition 3.2.18 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, $x = \widehat{x}$, $n = \widehat{n}$, $n_1 = \widehat{n}_1$, by Axiom 3.2.4 and Lemma 3.2.27 we have $\text{InputArray}(\widehat{x}, \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$ such that $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$.

Given $x = \widehat{x}$ and $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, by Definition 3.2.10 we have $x = [m_0, ..., m_{n_1}] \cong \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$. Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, we have $(\widehat{\gamma},\ \widehat{\sigma}_3, \square, \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}])$ such that $(\widehat{\gamma},\ \widehat{\sigma}_3, \square, \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \cong_\psi (\gamma,\ \sigma_3, \text{acc}, x = [m_0, ..., m_{n_1}])$ by Lemma 3.2.4. Given $(\gamma,\ \sigma_3, \text{acc},\ x = [m_0, ..., m_{n_1}]) \Downarrow_{d_4} (\gamma,\ \sigma_4, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_3, \square, \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \Downarrow'_{\widehat{d}_4} (\widehat{\gamma},\ \widehat{\sigma}_4\ \square, \text{skip})$ and ψ_4 such that $(\gamma,\ \sigma_4, \text{acc}, \text{skip}) \cong_{\psi_4} (\widehat{\gamma},\ \widehat{\sigma}_4, \square, \text{skip})$ and $d_4 \cong \widehat{d}_4$. Given $\text{pfree}(e) \notin x = [m_0, ..., m_{n_1}]$, we have $\psi_4 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_4) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_4)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$, $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty*})$, $\text{InputArray}(\widehat{x}, \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, and $(\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \Downarrow'_{\widehat{d}_4} (\widehat{\gamma}, \widehat{\sigma}_4, \square, \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3)) \Downarrow'_{\widehat{out1}} (\widehat{\gamma},\ \widehat{\sigma}_4,\ \square, \text{skip})$ by Vanilla C rule Input 1D Array.

Given $(\gamma,\ \sigma_4) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_4)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_4, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_4, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp1} (\gamma,\ \sigma_4, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3)) \Downarrow'_{\widehat{out1}} (\widehat{\gamma},\ \widehat{\sigma}_4, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $inp1 \cong \widehat{inp1}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp4} (\gamma,\ \sigma_4, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp4} (\gamma,\ \sigma_4, \text{acc}, \text{skip})$ by SMC^2 rule SMC Input Private 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$, $(\gamma,\ \sigma_2, \text{acc}, e_3) \Downarrow_{d_3} (\gamma,\ \sigma_3, \text{acc},\ n_1)$, $\gamma(x) = (l, \text{private const } bty*)$, $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, and $(\gamma,\ \sigma_3, \text{acc},\ x = [m_0, ..., m_{n_1}]) \Downarrow_{d_4} (\gamma,\ \sigma_4, \text{acc}, \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2, e_3))$ such that $(\gamma,\ \sigma,\ \text{acc}, \text{smcinput}(e_1, e_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{smcinput}(e_1, e_2, e_3) \cong_\psi \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3)$. Given $(\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp4} (\gamma,\ \sigma_4, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{smcinput}(e_1,\ e_2,\ e_3)$. Therefore, by Lemma 3.2.3 we have $\text{smcinput}(e_1,\ e_2,\ e_3) \cong \text{mcinput}(\widehat{e}_1, \widehat{e}_2, e_3)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcoutput}(e_1,\ e_2,\ e_3)) = \text{mcinput}(\text{Erase}(e_1,\ e_2,\ e_3))$. By Definition 3.2.8, we have $\text{Erase}(e_1, e_2, e_3) = \text{Erase}(e_1), \text{Erase}(e_2), \text{Erase}(e_3)$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$, $\text{Erase}(e_2) = \widehat{e}_2$, and $\text{Erase}(e_3) = \widehat{e}_3$. Therefore, we have $e_1 \cong \widehat{e}_1$, $e_2 \cong \widehat{e}_2$, and $e_3 \cong \widehat{e}_3$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x})$ and $d_1 \cong \widehat{d}_1$. Given $x \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma\ \sigma_1, \text{acc},\ e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,$

$\square, \widehat{n})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, n) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n})$ and $d_2 \cong \widehat{d}_2$. Given $\neq$ skip, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $e_3 \cong \widehat{e}_3$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{e}_3)$ such that $(\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{e}_3) \cong (\gamma, \sigma_2, \text{acc}, e_3)$. Given $(\gamma, \sigma_2, \text{acc}, e_3) \Downarrow_{d_3} (\gamma, \sigma_3, \text{acc}, n_1)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{e}_3) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{n}_1)$ and ψ_3 such that $(\gamma, \sigma_3, \text{acc}, n_1) \cong_{\psi_3} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{n}_1)$ and $d_3 \cong \widehat{d}_3$. Given $n_1 \neq$ skip, by Lemma 3.2.1 we have $\psi_3 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$ and $n_1 \cong_\psi \widehat{n}_1$. Given $\text{Label}(e_3, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$ and therefore $n_1 = \widehat{n}_1$ by Definition 3.2.18 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty*})$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty*}$ by Lemma 3.2.14.

Given $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, $x = \widehat{x}$, $n = \widehat{n}$, $n_1 = \widehat{n}_1$, by Axiom 3.2.4 and Lemma 3.2.27 we have $\text{InputArray}(\widehat{x}, \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$ such that $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$.

Given $x = \widehat{x}$ and $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, by Definition 3.2.10 we have $x = [m_0, ..., m_{n_1}] \cong \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$. Given $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}])$ such that $(\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \cong_\psi (\gamma, \sigma_3, \text{acc}, x = [m_0, ..., m_{n_1}])$. Given $(\gamma, \sigma_3, \text{acc}, x = [m_0, ..., m_{n_1}]) \Downarrow_{d_4} (\gamma, \sigma_4, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \Downarrow'_{\widehat{d}_4} (\widehat{\gamma}, \widehat{\sigma}_4\, \square, \text{skip})$ and ψ_4 such that $(\gamma, \sigma_4, \text{acc}, \text{skip}) \cong_{\psi_4} (\widehat{\gamma}, \widehat{\sigma}_4, \square, \text{skip})$ and $d_4 \cong \widehat{d}_4$. Given $\text{pfree}(e) \notin x = [m_0, ..., m_{n_1}]$, we have $\psi_4 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_4) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_4)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{x})$, $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n})$, $(\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{e}_3) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{n}_1)$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty*})$, $\text{InputArray}(\widehat{x}, \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, and $(\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \Downarrow'_{\widehat{d}_4} (\widehat{\gamma}, \widehat{\sigma}_4, \square, \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)) \Downarrow'_{\widehat{out1}} (\widehat{\gamma}, \widehat{\sigma}_4, \square, \text{skip})$ by Vanilla C rule Input 1D Array.

Given $(\gamma, \sigma_4) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_4)$, by Definition 3.2.20 we have $(\gamma, \sigma_4, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_4, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp4} (\gamma, \sigma_4, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)) \Downarrow'_{\widehat{out1}} (\widehat{\gamma}, \widehat{\sigma}_4, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $inp4 \cong \widehat{inp1}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2)) \Downarrow_{out} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2)) \Downarrow_{out} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule SMC Output Public Value, we have $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$, $\gamma(x) = (l, \text{public } bty)$, $\sigma_2(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1))$, $\text{DecodeVal}(\text{public } bty, 1, \omega) = n_1$, and $\text{OutputValue}(x, n, n_1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \text{mcoutput}(\widehat{e}_1, \widehat{e}_2))$ and ψ such that $(\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2))$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $\text{smcoutput}(e_1, e_2) \cong_\psi \text{mcinput}(\widehat{e}_1, \widehat{e}_2)$. Given $(\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2)) \Downarrow_{out} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{smcoutput}(e_1, e_2)$. Therefore, by Lemma 3.2.3 we have $\text{smcoutput}(e_1, e_2) \cong \text{mcoutput}(\widehat{e}_1, \widehat{e}_2)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcoutput}(e_1, e_2)) = \text{mcoutput}(\text{Erase}(e_1, e_2))$. By Definition 3.2.8, we have $\text{Erase}(e_1, e_2) = \text{Erase}(e_1), \text{Erase}(e_2))$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$ and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$, and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{x})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, x) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{x})$ and $d_1 \cong \widehat{d}_1$. Given $x \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma\ \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, n) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n})$ and $d_2 \cong \widehat{d}_2$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{public } bty)$, $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by $\text{public } bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1))$, $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, bty, \text{public}, 1))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$.

Given $\text{DecodeVal}(\text{public } bty, 1, \omega) = n_1$, $\text{public } bty \cong \widehat{bty}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, 1, \widehat{\omega}) = \widehat{n}_1$ and $n_1 \cong \widehat{n}_1$.

Given $\mathrm{OutputValue}(x, n, n_1)$, $x = \widehat{x}$, $n = \widehat{n}$, and $n_1 \cong \widehat{n}_1$, by Lemma 3.2.28 we have $\mathrm{OutputValue}(\widehat{x},\ \widehat{n},\ \widehat{n}_1)$ such that the corresponding output files are congruent.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \mathrm{mcoutput}(\widehat{e}_1,\ \widehat{e}_2))$, $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ \widehat{n})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty})$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega},\ \widehat{bty},\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, 1))$, $\mathrm{DecodeVal}(\widehat{bty}, 1, \widehat{\omega}) = \widehat{v}$, and $\mathrm{OutputValue}(\widehat{x},\ \widehat{n},\ \widehat{v})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \Box, \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2)) \Downarrow'_{\widehat{out}} (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \mathrm{skip})$ by Vanilla C rule Output Value.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_2, \mathrm{acc}, \mathrm{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \mathrm{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \mathrm{acc}, \mathrm{smcoutput}(e_1, e_2)) \Downarrow_{out} (\gamma, \sigma_2, \mathrm{acc}, \mathrm{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \Box, \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2)) \Downarrow'_{\widehat{out}} (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \mathrm{skip})$, $\Pi \cong_\psi \Sigma$, and $out \cong \widehat{out}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, \mathrm{smcoutput}(e_1, e_2)) \Downarrow_{out3} (\gamma,\ \sigma_2, \mathrm{acc}, \mathrm{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, \mathrm{smcoutput}(e_1, e_2)) \Downarrow_{out3} (\gamma,\ \sigma_2, \mathrm{acc}, \mathrm{skip})$ by SMC^2 rule SMC Output Private Value, we have $\mathrm{Label}(e_2, \gamma) = \mathrm{public}$, $(\gamma, \sigma, \mathrm{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \mathrm{acc}, x)$, $(\gamma, \sigma_1, \mathrm{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \mathrm{acc}, n)$, $\gamma(x) = (l,\ \mathrm{private}\ bty)$, $\sigma_2(l) = (\omega, \mathrm{private}\ bty,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty, \mathrm{private}, 1))$, $\mathrm{DecodeVal}(\mathrm{private}\ bty, 1, \omega) = n_1$, and $\mathrm{OutputValue}(x, n, n_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2))$ and ψ such that $(\gamma,\ \sigma,\ \mathrm{acc}, \mathrm{smcoutput}(e_1, e_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \Box, \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2))$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $\mathrm{smcoutput}(e_1, e_2) \cong_\psi \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2)$. Given $(\gamma, \sigma, \mathrm{acc}, \mathrm{smcoutput}(e_1, e_2)) \Downarrow_{out3} (\gamma, \sigma_2, \mathrm{acc}, \mathrm{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \mathrm{smcoutput}(e_1, e_2)$. Therefore, by Lemma 3.2.3 we have $\mathrm{smcoutput}(e_1, e_2) \cong \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2)$. By Definition 3.2.10 we have $\mathrm{Erase}(\mathrm{smcoutput}(e_1, e_2)) = \mathrm{mcoutput}(\mathrm{Erase}(e_1, e_2))$. By Definition 3.2.8, we have $\mathrm{Erase}(e_1, e_2) = \mathrm{Erase}(e_1), \mathrm{Erase}(e_2))$. By Definition 3.2.10 we have $\mathrm{Erase}(e_1) = \widehat{e}_1$ and $\mathrm{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$, and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \mathrm{acc}, e_1)$. Given $(\gamma, \sigma, \mathrm{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \mathrm{acc}, x)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{x})$ and ψ_1 such that $(\gamma, \sigma_1, \mathrm{acc}, x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{x})$ and $d_1 \cong \widehat{d}_1$. Given $x \neq \mathrm{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1,$

acc, e_2). Given (γ σ_1, acc, e_2) $\Downarrow_{d_2}$ (γ, σ_2, acc, n), by the inductive hypothesis we have ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{e}_2$) $\Downarrow'_{\widehat{d}_2}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, $\widehat{n}$) and ψ_2 such that (γ, σ_2, acc, n) $\cong_{\psi_2}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, $\widehat{n}$) and $d_2 \cong \widehat{d}_2$. Given $n \neq$ skip, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have (γ, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$) and $n \cong_\psi \widehat{n}$. Given Label(e_2, γ) = public, we have Label(n, γ) = public and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $\gamma(x)$ = (l, private bty), (γ, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$), and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x})$ = ($\widehat{l}$, $\widehat{bty}$) such that $l = \widehat{l}$ by private $bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\sigma_2(l)$ = (ω, private bty, 1, PermL(Freeable, private bty, private, 1)), (γ, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$), and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l})$ = ($\widehat{\omega}$, $\widehat{bty}$, 1, PermL(Freeable, bty, public, 1)) where $\omega_1 \cong_\psi \widehat{\omega}_1$.

Given DecodeVal(private bty, 1, ω) = n_1, private bty $\cong_\psi$ $\widehat{bty}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.41 we have DecodeVal(bty, 1, $\widehat{\omega}$) = $\widehat{n}_1$ and $n_1 \cong \widehat{n}_1$.

Given OutputValue(x, n, n_1), $x = \widehat{x}$, $n = \widehat{n}$, and $n_1 \cong \widehat{n}_1$, by Lemma 3.2.28 we have OutputValue($\widehat{x}$, $\widehat{n}$, $\widehat{n}_1$) such that the corresponding output files are congruent.

Given ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, mcoutput($\widehat{e}_1$, $\widehat{e}_2$)), ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}_1$) $\Downarrow'_{\widehat{d}_1}$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{x}$), ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{e}_2$) $\Downarrow'_{\widehat{d}_2}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, $\widehat{n}$), $\widehat{\gamma}(\widehat{x})$ = ($\widehat{l}$, $\widehat{bty}$), $\widehat{\sigma}_2(\widehat{l})$ = ($\widehat{\omega}$, $\widehat{bty}$, 1, PermL(Freeable, $\widehat{bty}$, public, 1)), DecodeVal($\widehat{bty}$, 1, $\widehat{\omega}$) = $\widehat{n}_1$, and OutputValue($\widehat{x}$, $\widehat{n}$, $\widehat{n}_1$), we have $\Sigma \triangleright$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, mcoutput($\widehat{e}_1$, $\widehat{e}_2$)) $\Downarrow'_{\widehat{out}}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, skip) by Vanilla C rule Output Value.

Given (γ, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$), by Definition 3.2.20 we have (γ, σ_2, acc, skip) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, skip). Therefore, we have (γ, σ, acc, smcoutput(e_1, e_2)) $\Downarrow_{out3}$ (γ, σ_2, acc, skip) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, mcoutput($\widehat{e}_1$, $\widehat{e}_2$)) $\Downarrow'_{\widehat{out}}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, skip), $\Pi \cong_\psi \Sigma$, and $out3 \cong \widehat{out}$ by Definition 3.2.21.

Case $\Pi \triangleright$ (γ, σ, acc, smcoutput(e_1, e_2, e_3)) $\Downarrow_{out1}$ (γ, σ_3, acc, skip)

Given $\Pi \triangleright$ (γ, σ, acc, smcoutput(e_1, e_2, e_3)) $\Downarrow_{out1}$ (γ, σ_3, acc, skip) by SMC^2 rule SMC Output Public 1D Array, we have Label(e_2, γ) = Label(e_3, γ) = public, (γ, σ, acc, e_1) $\Downarrow_{d_1}$ (γ, σ_1, acc, x), (γ, σ_1, acc, e_2) $\Downarrow_{d_2}$ (γ, σ_2, acc, n), (γ, σ_2, acc, e_3) $\Downarrow_{d_3}$ (γ, σ_3, acc, n_1), $\gamma(x)$ = (l, public const $bty*$), $\sigma_3(l)$ = (ω, public const $bty*$, 1, PermL(Freeable, public const $bty*$, public, 1), DecodePtr(public const $bty*$, 1, ω) = [1, [(l_1, 0)], [1], 1], $\sigma_3(l_1)$ = (ω_1, public bty, n_1, PermL(Freeable, public bty, public, n_1)), DecodeVal(public bty, n_1, ω_1) = [m_0, ...,

$m_{n_1}]$, and $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$ and ψ such that $(\gamma,\ \sigma, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{smcoutput}(e_1, e_2, e_3) \cong_\psi \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)$. Given $(\gamma,\ \sigma, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow_{out1} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{smcoutput}(e_1, e_2, e_3)$. Therefore, by Lemma 3.2.3 we have $\text{smcoutput}(e_1, e_2, e_3) \cong \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcoutput}(e_1,\ e_2,\ e_3)) = \text{mcoutput}(\text{Erase}(e_1,\ e_2,\ e_3))$. By Definition 3.2.8, we have $\text{Erase}(e_1,\ e_2,\ e_3) = \text{Erase}(e_1), \text{Erase}(e_2), \text{Erase}(e_3)$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$, $\text{Erase}(e_2) = \widehat{e}_2$, and $\text{Erase}(e_3) = \widehat{e}_3$. Therefore, we have $e_1 \cong \widehat{e}_1$, $e_2 \cong \widehat{e}_2$, and $e_3 \cong \widehat{e}_3$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x})$ and $d_1 \cong \widehat{d}_1$. Given $x \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma\ \sigma_1, \text{acc},\ e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, n) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n})$ and $d_2 \cong \widehat{d}_2$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $e_3 \cong \widehat{e}_3$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{e}_3)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{e}_3) \cong_\psi (\gamma, \sigma_2, \text{acc}, e_3)$ by Definition 3.2.20. Given $(\gamma, \sigma_2, \text{acc}, e_3) \Downarrow_{d_3} (\gamma,\ \sigma_3, \text{acc},\ n_1)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{e}_3) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, \widehat{n}_1)$ and ψ_3 such that $(\gamma,\ \sigma_3, \text{acc},\ n_1) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, \widehat{n}_1)$ and $d_3 \cong \widehat{d}_3$. Given $n_1 \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_3 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$ and $n_1 \cong_\psi \widehat{n}_1$. Given $\text{Label}(e_3, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$ and therefore $n_1 = \widehat{n}_1$ by Definition 3.2.18 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty*})$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma_3(l) = (\omega,\ \text{public const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{public const } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$,

and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_3(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable, const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{public const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$. By Definition 3.2.13 we have $l_1 \cong_\psi \widehat{l}_1$.

Given $\sigma_3(l_1) = (\omega_1, \text{public } bty, n_1, \text{PermL}(\text{Freeable, public } bty, \text{public}, n_1))$, $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, and $l_1 \cong_\psi \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_3(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}_1, \text{PermL}(\text{Freeable, } bty, \text{public}, \widehat{n}_1))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n_1 = \widehat{n}_1$.

Given $\text{DecodeVal}(\text{public } bty, n_1, \omega_1) = [m_0, ..., m_{n_1}]$, $\text{public } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}_1, \widehat{\omega}_1) = [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}]$ and $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$.

Given $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$, $x = \widehat{x}$, $n = \widehat{n}$, and $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, by Lemma 3.2.29 we have $\text{OutputArray}(\widehat{x},\ \widehat{n}, [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}])$ such that the corresponding output files are congruent.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcoutput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$, $(\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{e}_3) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{n}_1)$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_3(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable, const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)],\ [1],\ 1]$, $\widehat{\sigma}_3(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n}_2, \text{PermL}(\text{Freeable, } \widehat{bty}, \text{public}, \widehat{n}_1))$, $\text{DecodeVal}(\widehat{bty},\ \widehat{n}_1,\ \widehat{\omega}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, and $\text{OutputArray}(\widehat{x},\ \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)) \Downarrow'_{\widehat{out1}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule Output 1D Array.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow_{out1} (\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcoutput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3)) \Downarrow'_{\widehat{out1}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $out1 \cong \widehat{out1}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow_{out4} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow_{out4} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$ by SMC^2 rule SMC Output Private 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$, $(\gamma,\ \sigma_2, \text{acc}, e_3) \Downarrow_{d_3} (\gamma,\ \sigma_3, \text{acc},\ n_1)$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_3(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable, private const } bty*, \text{private}, 1)$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1,\ [(l_1, 0)], [1], 1]$,

$\sigma_3(l_1) = (\omega_1, \text{ private } bty, n_1, \text{PermL}(\text{Freeable}, \text{ private } bty, \text{ private}, n_1))$, $\text{DecodeVal}(\text{private } bty, n_1, \omega_1) = [m_0, ..., m_{n_1}]$, and $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$ and ψ such that $(\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $\text{smcoutput}(e_1, e_2, e_3) \cong_\psi \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)$. Given $(\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow_{out4} (\gamma, \sigma_3, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{smcoutput}(e_1, e_2, e_3)$. Therefore, by Lemma 3.2.3 we have $\text{smcoutput}(e_1, e_2, e_3) \cong \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcoutput}(e_1, e_2, e_3)) = \text{mcoutput}(\text{Erase}(e_1, e_2, e_3))$. By Definition 3.2.8, we have $\text{Erase}(e_1, e_2, e_3) = \text{Erase}(e_1), \text{Erase}(e_2), \text{Erase}(e_3)$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$, $\text{Erase}(e_2) = \widehat{e}_2$, and $\text{Erase}(e_3) = \widehat{e}_3$. Therefore, we have $e_1 \cong \widehat{e}_1$, $e_2 \cong \widehat{e}_2$, and $e_3 \cong \widehat{e}_3$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{x})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, x) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{x})$ and $d_1 \cong \widehat{d}_1$. Given $x \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma\ \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, n) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n})$ and $d_2 \cong \widehat{d}_2$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $e_3 \cong \widehat{e}_3$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{e}_3)$ such that $(\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{e}_3) \cong_\psi (\gamma, \sigma_2, \text{acc}, e_3)$. Given $(\gamma, \sigma_2, \text{acc}, e_3) \Downarrow_{d_3} (\gamma, \sigma_3, \text{acc}, n_1)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{e}_3) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{n}_1)$ and ψ_3 such that $(\gamma, \sigma_3, \text{acc}, n_1) \cong_{\psi_3} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{n}_1)$ and $d_3 \cong \widehat{d}_3$. Given $n_1 \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_3 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$ and $n_1 \cong_\psi \widehat{n}_1$. Given $\text{Label}(e_3, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$ and therefore $n_1 = \widehat{n}_1$ by Definition 3.2.18 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_3(l) = (\omega, \text{ private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$,

and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_3(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable, const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0), [1], 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$. By Definition 3.2.13 we have $l_1 \cong_\psi \widehat{l}_1$.

Given $\sigma_3(l_1) = (\omega_1, \text{private } bty, n_1, \text{PermL}(\text{Freeable, private } bty, \text{private}, n_1))$, $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, and $l_1 \cong_\psi \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_3(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}_1, \text{PermL}(\text{Freeable, } bty, \text{public}, \widehat{n}_1))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n_1 = \widehat{n}_1$.

Given $\text{DecodeVal}(\text{private } bty, n_1, \omega_1) = [m_0, ..., m_{n_1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}_1, \widehat{\omega}_1) = [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}]$ and $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}]$.

Given $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$, $x = \widehat{x}$, $n = \widehat{n}$, and $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, by Lemma 3.2.29 we have $\text{OutputArray}(\widehat{x},\ \widehat{n}, [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}])$ such that the corresponding output files are congruent.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcoutput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$, $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \Downarrow'_{\widehat{d}_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_3(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable, const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)], [1],\ 1]$, $\widehat{\sigma}_3(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n}_2, \text{PermL}(\text{Freeable, } \widehat{bty}, \text{public}, \widehat{n}_1))$, $\text{DecodeVal}(\widehat{bty},\ \widehat{n}_1,\ \widehat{\omega}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, and $\text{OutputArray}(\widehat{x}, \widehat{n}, [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}])$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcoutput}(\widehat{e}_1, \widehat{e}_2,\ \widehat{e}_3)) \Downarrow'_{\widehat{out1}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule Output 1D Array.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow_{out4} (\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcoutput}(\widehat{e}_1, \widehat{e}_2,\ \widehat{e}_3)) \Downarrow'_{\widehat{out1}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $out4 \cong \widehat{out1}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x(\overline{p})) \Downarrow_{df} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ ty\ x(\overline{p})) \Downarrow_{df} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ by SMC^2 rule Function Declaration, we have $\text{acc} = 0$, $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, and $\sigma_1 = \sigma[l \rightarrow (\text{NULL}, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun}(\text{public}))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}}))$ and ψ such that $(\gamma,\ \sigma,\ \mathrm{acc},\ ty\ x(\overline{p})) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}}))$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x(\overline{p}) \cong_{\psi} \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})$. Given $(\gamma,\ \sigma,\ \mathrm{acc},\ ty\ x(\overline{p})) \Downarrow_{df} (\gamma_1, \sigma_1, \mathrm{acc}, \mathrm{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin ty\ x(\overline{p})$. Therefore, by Lemma 3.2.3 we have $ty\ x(\overline{p}) \cong \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})$. By Definition 3.2.10 we have $\mathrm{Erase}(ty\ x(\overline{p})) = \mathrm{Erase}(ty)\ \widehat{x}(\mathrm{Erase}(\overline{p}))$ where $x = \widehat{x}$, $\mathrm{Erase}(ty) = \widehat{ty}$, and $\mathrm{Erase}(\overline{p}) = \widehat{\overline{p}}$ by Definition 3.2.9. Therefore, we have $ty \cong \widehat{ty}$ and $\overline{p} \cong \widehat{\overline{p}}$.

Given $\mathrm{GetFunTypeList}(\overline{p}) = \overline{ty}$ and $\overline{p} \cong \widehat{\overline{p}}$, by Lemma 3.2.30 we have $\mathrm{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}}$ where $\overline{ty} \cong \widehat{\overline{ty}}$. Therefore, we have $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$ by Definition 3.2.6.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, $x = \widehat{x}$, $l = \widehat{l}$, $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$, and $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{\overline{ty}} \rightarrow \widehat{ty})]$ such that $(\gamma_1, \sigma) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given $\sigma_1 = \sigma[l \rightarrow (\mathrm{NULL}, \overline{ty} \rightarrow ty, 1, \mathrm{PermL_Fun}(\mathrm{public}))]$, $l = \widehat{l}$, $(\gamma_1, \sigma) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma})$, and $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\mathrm{NULL},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1, \mathrm{PermL_Fun}(\mathrm{public}))]$ such that $(\gamma_1, \sigma_1) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}}))$, $\widehat{l} = \phi()$, $\mathrm{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}}$, $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{\overline{ty}} \rightarrow \widehat{ty})]$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\mathrm{NULL},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \mathrm{PermL_Fun}(\mathrm{public}))]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})) \Downarrow'_{\widehat{df}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \mathrm{skip})$ by Vanilla C rule Function Declaration.

Given $(\gamma_1, \sigma_1) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, by Definition 3.2.20 we have $(\gamma_1, \sigma_1, \mathrm{acc}, \mathrm{skip}) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \mathrm{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \mathrm{acc},\ ty\ x(\overline{p})) \Downarrow_{df} (\gamma_1, \sigma_1, \mathrm{acc}, \mathrm{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})) \Downarrow'_{\widehat{df}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \mathrm{skip})$, $\Pi \cong_{\psi} \Sigma$, and $df \cong \widehat{df}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, ty\ x(\overline{p})\{s\}) \Downarrow_{fpd} (\gamma, \sigma_2, \mathrm{acc}, \mathrm{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, ty\ x(\overline{p})\{s\}) \Downarrow_{fpd} (\gamma, \sigma_2, \mathrm{acc}, \mathrm{skip})$ by SMC^2 rule Pre-Declared Function Definition, we have $\mathrm{acc} = 0$, $x \in \gamma$, $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $\mathrm{CheckPublicEffects}(s, x, \gamma, \sigma) = n$, $\mathrm{EncodeFun}(s, n, \overline{p}) = \omega$, $\sigma = \sigma_1[l \rightarrow (\mathrm{NULL}, \overline{ty} \rightarrow ty,\ 1, \mathrm{PermL_Fun}(\mathrm{public}))]$, and $\sigma_2 = \sigma_1[l \rightarrow (\omega,\ \overline{ty} \rightarrow ty,\ 1, \mathrm{PermL_Fun}(\mathrm{public}))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\})$ and ψ such that $(\gamma, \sigma, \mathrm{acc}, ty\ x\ (\overline{p})\{s\}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x(\overline{p})\{s\} \cong_{\psi} \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\}$. Given $(\gamma, \sigma, \mathrm{acc}, ty\ x(\overline{p})\{s\}) \Downarrow_{fpd} (\gamma, \sigma_2, \mathrm{acc}, \mathrm{skip})$, by

Lemma 3.2.2 we have $(l, \mu) \notin ty\ x(\overline{p})\{s\}$. Therefore, by Lemma 3.2.3 we have $ty\ x(\overline{p})\{s\} \cong \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x(\overline{p})\{s\}) = \text{Erase}(ty\ x(\overline{p}))\ \{\text{Erase}(s)\}$, $\text{Erase}(ty\ x(\overline{p})) = \text{Erase}(ty)\ \widehat{x}(\text{Erase}(\overline{p}))$ where $x = \widehat{x}$, $\text{Erase}(ty) = \widehat{ty}$, $\text{Erase}(s) = \widehat{s}$, and $\text{Erase}(\overline{p}) = \widehat{\overline{p}}$ by Definition 3.2.9. Therefore, we have $ty \cong \widehat{ty}$ and $\overline{p} \cong \widehat{\overline{p}}$.

Given $x \in \gamma$, $\gamma(x) = (l, \overline{ty} \to ty)$, $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{\overline{ty}} \to \widehat{ty})$ such that $l = \widehat{l}$ by $\overline{ty} \to ty \cong \widehat{\overline{ty}} \to \widehat{ty}$ by Lemma 3.2.14 and $\widehat{x} \in \widehat{\gamma}$ by Lemma 3.2.33.

Given $\text{EncodeFun}(s, n, \overline{p}) = \omega$, $s \cong \widehat{s}$, and $\overline{p} \cong \widehat{\overline{p}}$, by Lemma 3.2.46 we have $\text{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}}) = \widehat{\omega}$ such that $\omega \cong_{\psi} \widehat{\omega}$.

Given $\sigma = \sigma_1[l \to (\text{NULL},\ \overline{ty} \to ty,\ 1, \text{PermL_Fun}(\text{public}))]$, $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.36 we have $\widehat{\sigma} = \widehat{\sigma}_1[\widehat{l} \to (\text{NULL},\ \widehat{\overline{ty}} \to \widehat{ty},\ 1, \text{PermL_Fun}(\text{public}))]$ and $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given $\sigma_2 = \sigma_1[l \to (\omega,\ \overline{ty} \to ty,\ 1, \text{PermL_Fun}(\text{public}))]$, $(\gamma,\ \sigma_1) \cong (\widehat{\gamma},\ \widehat{\sigma}_1)$, $l = \widehat{l}$, $\omega \cong_{\psi} \widehat{\omega}$, and $\overline{ty} \to ty \cong \widehat{\overline{ty}} \to \widehat{ty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \to (\widehat{\omega}, \widehat{\overline{ty}} \to \widehat{ty}, 1, \text{PermL_Fun}(\text{public}))]$ such that $(\gamma,\ \sigma_2) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\})$, $x \in \widehat{\gamma}$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{\overline{ty}} \to \widehat{ty})$, $\widehat{\sigma} = \widehat{\sigma}_1[\widehat{l} \to (\text{NULL},\ \widehat{\overline{ty}} \to \widehat{ty},\ 1, \text{PermL_Fun}(\text{public}))]$, $\text{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}}) = \widehat{\omega}$, and $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \to (\widehat{\omega}, \widehat{\overline{ty}} \to \widehat{ty}, 1, \text{PermL_Fun}(\text{public}))]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}) \Downarrow'_{\widehat{fpd}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Pre-Declared Function Definition.

Given $(\gamma,\ \sigma_2) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_2, \text{acc}, \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma, \text{acc},\ ty\ x(\overline{p})\{s\}) \Downarrow_{fpd} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}) \Downarrow'_{\widehat{fpd}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_{\psi} \Sigma$, and $fpd \cong \widehat{fpd}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc},\ ty\ x(\overline{p})\{s\}) \Downarrow_{fd} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ ty\ x(\overline{p})\{s\}) \Downarrow_{fd} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ by SMC^2 rule Function Definition, we have $l = \phi()$, $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$, $x \notin \gamma$, $\gamma_1 = \gamma[x \to (l, \overline{ty} \to ty)]$, $\text{acc} = 0$, $\text{CheckPublicEffects}(s, x, \gamma, \sigma) = n$, $\text{EncodeFun}(s, n, \overline{p}) = \omega$, and $\sigma_1 = \sigma[l \to (\omega, \overline{ty} \to ty, 1, \text{PermL_Fun}(\text{public}))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\})$ such that $(\gamma, \sigma, \text{acc},\ ty\ x(\overline{p})\{s\}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x(\overline{p})\{s\} \cong_{\psi} \widehat{ty}\ \widehat{x}(\widehat{\overline{e}})\ \{\widehat{s}\}$. Given $(\gamma, \sigma, \text{acc},\ ty\ x(\overline{p})\{s\}) \Downarrow_{fd} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$,

by Lemma 3.2.2 we have $(l, \mu) \notin ty\ x(\overline{p})\{s\}$. Therefore, by Lemma 3.2.3 we have $ty\ x(\overline{p})\{s\} \cong \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\})$. By Definition 3.2.10 we have $\text{Erase}(ty\ x(\overline{p})\{s\}) = \text{Erase}(ty\ x(\overline{p}))\ \{\text{Erase}(s)\}$, $\text{Erase}(ty\ x(\overline{p})) = \text{Erase}(ty)\ \widehat{x}(\text{Erase}(\overline{p}))$ where $x = \widehat{x}$, $\text{Erase}(ty) = \widehat{ty}$, $\text{Erase}(\overline{p}) = \widehat{\overline{p}}$, and $\text{Erase}(s) = \widehat{s}$ by Definition 3.2.9. Therefore, we have $ty \cong \widehat{ty}$, $\overline{p} \cong \widehat{\overline{p}}$, and $s \cong \widehat{s}$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$ and $\overline{p} \cong \widehat{\overline{p}}$, by Lemma 3.2.30 we have $\text{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}}$ where $\overline{ty} \cong \widehat{\overline{ty}}$.

Given $x \notin \gamma$, $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, by Lemma 3.2.32 we have $\widehat{x} \notin \widehat{\gamma}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, $x = \widehat{x}$, $l = \widehat{l}$, $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$, and $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{\overline{ty}} \rightarrow \widehat{ty})]$ such that $(\gamma_1, \sigma) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given $\text{EncodeFun}(s, n, \overline{p}) = \omega$, $s \cong \widehat{s}$, and $\overline{p} \cong \widehat{\overline{p}}$, by Lemma 3.2.46 we have $\text{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}}) = \widehat{\omega}$ such that $\omega \cong_{\psi} \widehat{\omega}$.

Given $\sigma_1 = \sigma[l \rightarrow (\text{NULL}, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun}(\text{public}))]$, $l = \widehat{l}$, $(\gamma_1, \sigma) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma})$, and $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\text{NULL},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1, \text{PermL_Fun}(\text{public}))]$ such that $(\gamma_1, \sigma_1) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\})$, $x \notin \widehat{\gamma}$, $\widehat{l} = \phi()$, $\text{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}}$, $\widehat{\gamma}_1 = \widehat{\gamma}[x \rightarrow (\widehat{l},\ \widehat{\overline{ty}} \rightarrow \widehat{ty})]$, $\text{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}}) = \widehat{\omega}$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \text{PermL_Fun}(\text{public}))]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}) \Downarrow'_{\widehat{fd}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \text{skip})$ by Vanilla C rule Function Definition.

Given $(\gamma_1, \sigma_1) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, by Definition 3.2.20 we have $(\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \text{acc}, ty\ x(\overline{p})\{s\}) \Downarrow_{fd} (\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}) \Downarrow'_{\widehat{fd}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \text{skip})$, $\Pi \cong_{\psi} \Sigma$, and $fd \cong \widehat{fd}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma,\ \sigma, \text{acc}, x(\overline{e})) \Downarrow_{fc} (\gamma, \sigma_2, \text{acc}, \text{NULL})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x(\overline{e})) \Downarrow_{fc} (\gamma, \sigma_2, \text{acc}, \text{NULL})$ by SMC^2 rule Function Call No Return With Public Side Effects, we have $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $\sigma(l) = (\omega, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun}(\text{public}))$, $\text{DecodeFun}(\omega) = (s, 1, \overline{p})$, $\text{acc} = 0$, $\text{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$, $(\gamma,\ \sigma,\ \text{acc},\ s_1) \Downarrow_{d_1} (\gamma_1,\ \sigma_1,\ \text{acc},\ \text{skip})$, and $(\gamma_1, \sigma_1, \text{acc},\ s) \Downarrow_{d_2} (\gamma_2, \sigma_2,\ \text{acc},$

skip).

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}(\widehat{\overline{e}}))$ and ψ such that $(\gamma, \sigma, \text{acc}, x(\overline{e})) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}(\widehat{\overline{e}}))$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x(\overline{e}) \cong_\psi \widehat{x}(\widehat{\overline{e}})$. Given $(\gamma, \sigma, \text{acc}, x(\overline{e})) \Downarrow_{fc} (\gamma,\ \sigma_2, \text{acc}, \text{NULL})$, by Lemma 3.2.2 we have $(l, \mu) \notin x(\overline{e})$. Therefore, by Lemma 3.2.3 we have $x(\overline{e}) \cong \widehat{x}(\widehat{\overline{e}})$. By Definition 3.2.10 we have $\text{Erase}(x(e)) = \widehat{x}(\text{Erase}(\overline{e}))$ where $x = \widehat{x}$ and $\text{Erase}(\overline{e}) = \widehat{\overline{e}}$ by Definition 3.2.8. Therefore, we have $\overline{e} \cong \widehat{\overline{e}}$.

Given $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{\overline{ty}} \rightarrow \widehat{ty})$ such that $l = \widehat{l}$ by $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun}(\text{public}))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \text{PermL_Fun}(\text{public}))$ where $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodeFun}(\omega) = (s, 1, \overline{p})$ and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.47 we have $\text{DecodeFun}(\widehat{\omega}) = (\widehat{s},\ \square,\ \widehat{\overline{p}})$ such that $s \cong \widehat{s}$ and $\overline{p} \cong \widehat{\overline{p}}$.

Given $\text{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$, $\overline{p} \cong \widehat{\overline{p}}$, and $\overline{e} \cong \widehat{\overline{e}}$, by Lemma 3.2.31 we have $\text{GetFunParamAssign}(\widehat{\overline{p}}, \widehat{\overline{e}}) = \widehat{s}_1$ where $s_1 \cong_\psi \widehat{s}_1$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $s_1 \cong_\psi \widehat{s}_1$, we have $(\gamma, \sigma, \text{acc}, s_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{s}_1)$ by Definition 3.2.20. Given $(\gamma, \sigma, \text{acc}, s_1) \Downarrow_{d_1} (\gamma_1,\ \sigma_1, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{s}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \text{skip})$ and ψ_1 such that $(\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \text{skip})$ and $d_1 \cong \widehat{d}_1$. By Definition 3.2.20, we have $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$. By Lemma 3.2.17, we have $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$ and $s \cong \widehat{s}$, BY Lemma 3.2.4 we have $(\gamma_1, \sigma_1, \text{acc},\ s) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \widehat{s})$ by Definition 3.2.20. Given $(\gamma_1, \sigma_1, \text{acc},\ s) \Downarrow_{d_2} (\gamma_2, \sigma_2, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \widehat{s}) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2, \square, \text{skip})$ and ψ_2 such that $(\gamma_2, \sigma_2, \text{acc}, \text{skip}) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2, \square, \text{skip})$ and $d_2 \cong \widehat{d}_2$. By Definition 3.2.20, we have $(\gamma_2, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2)$. By Lemma 3.2.17, we have $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}(\widehat{\overline{e}}))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{\overline{ty}} \rightarrow \widehat{ty})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \text{PermL_Fun}(\text{public}))$, $\text{DecodeFun}(\widehat{\omega}) = (\widehat{s},\ \square,\ \widehat{\overline{p}})$, $\text{GetFunParamAssign}(\widehat{\overline{p}}, \widehat{\overline{e}}) = \widehat{s}_1$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$, and $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{s}) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}(\widehat{\overline{e}})) \Downarrow'_{\widehat{fc}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \text{NULL})$ by Vanilla C rule Function Call.

Given $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $\mathrm{NULL} = \mathrm{NULL}$, by Definition 3.2.17 we have $\mathrm{NULL} \cong_{\psi_2} \mathrm{NULL}$ and by Definition 3.2.20 we have $(\gamma, \sigma_2, \mathrm{acc}, \mathrm{NULL}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \mathrm{NULL})$. Therefore, we have $(\gamma, \sigma, \mathrm{acc}, x(\overline{e})) \Downarrow_{fc} (\gamma, \sigma_2, \mathrm{acc}, \mathrm{NULL}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{x}(\widehat{\overline{e}})) \Downarrow'_{\widehat{fc}} (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \mathrm{NULL})$, $\Pi \cong_{\psi_2} \Sigma$, and $fc \cong \widehat{fc}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, x(\overline{e})) \Downarrow_{fc1} (\gamma, \sigma_2, \mathrm{acc}, \mathrm{NULL})$

Given $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, x(\overline{e})) \Downarrow_{fc1} (\gamma, \sigma_2, \mathrm{acc}, \mathrm{NULL})$ by SMC^2 rule Function Call No Return Without Public Side Effects, we have $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $\sigma(l) = (\omega, \overline{ty} \rightarrow ty, 1,$ $\mathrm{PermL_Fun(public)})$, $\mathrm{DecodeFun}(\omega) = (s, 1, \overline{p})$, $\mathrm{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$, $(\gamma, \sigma, \mathrm{acc}, s_1) \Downarrow_{d_1} (\gamma_1,\ \sigma_1,$ $\mathrm{acc}, \mathrm{skip})$, and $(\gamma_1, \sigma_1, \mathrm{acc},\ s) \Downarrow_{d_2} (\gamma_2, \sigma_2, \mathrm{acc}, \mathrm{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{x}(\widehat{\overline{e}}))$ and ψ such that $(\gamma, \sigma, \mathrm{acc}, x(\overline{e})) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{x}(\widehat{\overline{e}}))$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_{\psi}$ $(\widehat{\gamma},\ \widehat{\sigma})$ and $x(\overline{e}) \cong_{\psi} \widehat{x}(\widehat{\overline{e}})$. Given $(\gamma, \sigma, \mathrm{acc}, x(\overline{e})) \Downarrow_{fc1} (\gamma,\ \sigma_2, \mathrm{acc}, \mathrm{NULL})$, by Lemma 3.2.2 we have $(l, \mu) \notin x(\overline{e})$. Therefore, by Lemma 3.2.3 we have $x(\overline{e}) \cong \widehat{x}(\widehat{\overline{e}})$. By Definition 3.2.10 we have $\mathrm{Erase}(x(e)) = \widehat{x}(\mathrm{Erase}(\overline{e}))$ where $x = \widehat{x}$ and $\mathrm{Erase}(\overline{e}) = \widehat{\overline{e}}$ by Definition 3.2.8. Therefore, we have $\overline{e} \cong \widehat{\overline{e}}$.

Given $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{\overline{ty}} \rightarrow \widehat{ty})$ such that $l = \widehat{l}$ by $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \overline{ty} \rightarrow ty, 1, \mathrm{PermL_Fun(public)})$, $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \mathrm{PermL_Fun(public)})$ where $\omega \cong_{\psi} \widehat{\omega}$.

Given $\mathrm{DecodeFun}(\omega) = (s, 1, \overline{p})$ and $\omega \cong_{\psi} \widehat{\omega}$, by Lemma 3.2.47 we have $\mathrm{DecodeFun}(\widehat{\omega}) = (\widehat{s},\ \Box,\ \widehat{\overline{p}})$ such that $s \cong \widehat{s}$ and $\overline{p} \cong \widehat{\overline{p}}$.

Given $\mathrm{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$, $\overline{p} \cong \widehat{\overline{p}}$, and $\overline{e} \cong \widehat{\overline{e}}$, by Lemma 3.2.31 we have $\mathrm{GetFunParamAssign}(\widehat{\overline{p}}, \widehat{\overline{e}}) = \widehat{s}_1$ where $s_1 \cong_{\psi} \widehat{s}_1$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $s_1 \cong_{\psi} \widehat{s}_1$, we have $(\gamma, \sigma, \mathrm{acc}, s_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{s}_1)$ by Definition 3.2.20. Given $(\gamma, \sigma,$ $\mathrm{acc}, s_1) \Downarrow_{d_1} (\gamma_1,\ \sigma_1, \mathrm{acc}, \mathrm{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{s}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \Box, \mathrm{skip})$ and ψ_1 such that $(\gamma_1, \sigma_1, \mathrm{acc}, \mathrm{skip}) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \Box, \mathrm{skip})$ and $d_1 \cong \widehat{d}_1$. By Definition 3.2.20, we have $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$. By Lemma 3.2.17, we have $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$ and $s \cong \widehat{s}$, by Lemma 3.2.4 we have $(\gamma_1, \sigma_1, \text{acc},\ s) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \widehat{s})$ by Definition 3.2.20. Given $(\gamma_1, \sigma_1, \text{acc},\ s) \Downarrow_{d_2} (\gamma_2, \sigma_2, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \widehat{s}) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2, \square, \text{skip})$ and ψ_2 such that $(\gamma_2, \sigma_2, \text{acc}, \text{skip}) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2, \square, \text{skip})$ and $d_2 \cong \widehat{d}_2$. By Definition 3.2.20, we have $(\gamma_2, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2)$. By Lemma 3.2.17, we have $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}(\widehat{\overline{e}}))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{\overline{ty}} \rightarrow \widehat{ty})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \text{PermL_Fun(public)})$, $\text{DecodeFun}(\widehat{\omega}) = (\widehat{s},\ \square,\ \widehat{\overline{p}})$, $\text{GetFunParamAssign}(\widehat{\overline{p}}, \widehat{\overline{e}}) = \widehat{s}_1$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$, and $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{s}) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}(\widehat{\overline{e}})) \Downarrow'_{\widehat{fc}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{NULL})$ by Vanilla C rule Function Call.

Given $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $\text{NULL} \cong \text{NULL}$, by Lemma 3.2.4 we have $(\gamma, \sigma_2, \text{acc}, \text{NULL}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{NULL})$. Therefore, we have $(\gamma, \sigma, \text{acc}, x(\overline{e})) \Downarrow_{fc1} (\gamma, \sigma_2, \text{acc}, \text{NULL}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}(\widehat{\overline{e}})) \Downarrow'_{\widehat{fc}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{NULL})$, $\Pi \cong_{\psi_2} \Sigma$, and $fc1 \cong \widehat{fc}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{malloc}(e)) \Downarrow_{mal} (\gamma, \sigma_2, \text{acc}, (l, 0))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{malloc}(e)) \Downarrow_{mal} (\gamma, \sigma_2, \text{acc}, (l, 0))$ by SMC^2 rule Public Malloc, we have $\text{Label}(e, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma,\ \sigma,\ \text{acc},\ e) \Downarrow_{d_1} (\gamma,\ \sigma_1,\ \text{acc},\ n)$, $l = \phi()$, and $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, n)\big]\big)\big]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{malloc}(\widehat{e}))$ and ψ such that $(\gamma,\ \sigma,\ \text{acc}, \text{malloc}(e)) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{malloc}(\widehat{e}))$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{malloc}(e) \cong_{\psi} \text{malloc}(\widehat{e})$. Given $(\gamma, \sigma, \text{acc}, \text{malloc}(e)) \Downarrow_{mal} (\gamma, \sigma_2, \text{acc}, (l, 0))$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{malloc}(e)$. Therefore, by Lemma 3.2.3 we have $\text{malloc}(e) \cong \text{malloc}(\widehat{e})$. By Definition 3.2.10 we have $\text{Erase}(\text{malloc}(e)) = \text{malloc}(\text{Erase}(e))$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_{\psi} (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$ and $d_1 \cong \widehat{d}_1$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_{\psi} \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$, and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$, $l = \widehat{l}$, and $(l, 0) \cong_{\psi} (\widehat{l}, 0)$.

Given $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, n)\big)\big]$, $n = \widehat{n}$, $l = \widehat{l}$, $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, by

Lemma 3.2.7 we have $\widehat{\sigma}_2 = \widehat{\sigma}_1\big[\widehat{l} \rightarrow \big(\text{NULL},\ \text{void}*,\ \hat{n},\ \text{PermL}(\text{Freeable},\ \text{void}*,\ \text{public},\ \widehat{n})\big)\big]$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{malloc}(\widehat{e}))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \hat{n})$, $\widehat{l} = \phi()$, and $\widehat{\sigma}_2 = \widehat{\sigma}_1\big[\widehat{l} \rightarrow \big(\text{NULL},\ \text{void}*,\ \hat{n},\ \text{PermL}(\text{Freeable},\ \text{void}*,\ \text{public}, \widehat{n})\big)\big]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{malloc}(\widehat{e})) \Downarrow'_{\widehat{mal}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ (\widehat{l}, 0))$ by Vanilla C rule Malloc.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, (\widehat{l}, 0))$. Therefore, we have $(\gamma, \sigma, \text{acc}, \text{malloc}(e)) \Downarrow_{mal} (\gamma,\ \sigma_2, \text{acc}, (l,\ 0)) \cong (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{malloc}(\widehat{e})) \Downarrow'_{\widehat{mal}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, (\widehat{l}, 0))$, $\Pi \cong_\psi \Sigma$, and $mal \cong \widehat{mal}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{pmalloc}(e,\ ty)) \Downarrow_{malp} (\gamma, \sigma_2, \text{acc}, (l, 0))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{pmalloc}(e,\ ty)) \Downarrow_{malp} (\gamma, \sigma_2, \text{acc}, (l, 0))$ by SMC^2 rule Private Malloc, we have $(ty = \text{private int}) \vee (ty = \text{private float})$, $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma,\ \sigma_1,\ \text{acc},\ n)$, $\text{acc} = 0$, $l = \phi()$, and $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL},\ \text{void}*,\ n,\ \text{PermL}(\text{Freeable}, ty, \text{private}, n)\big]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{malloc}(\widehat{e}'))$ such that $(\gamma, \sigma, \text{acc}, \text{pmalloc}(e,\ ty)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{malloc}(\widehat{e}'))$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{pmalloc}(e, ty) \cong_\psi \text{malloc}(\widehat{e}')$. Given $(\gamma, \sigma, \text{acc}, \text{pmalloc}(e,\ ty)) \Downarrow_{malp} (\gamma, \sigma_2, \text{acc}, (l, 0))$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{pmalloc}(e, ty)$. Therefore, by Lemma 3.2.3 we have $\text{pmalloc}(e, ty) \cong \text{malloc}(\widehat{e}')$. By Lemma 3.2.6, we have that $\widehat{e}' = \text{sizeof}(\text{Erase}(ty)) \cdot \text{Erase}(e)$. By Definition 3.2.6, we have $\text{Erase}(ty) = \widehat{ty}$ such that $ty \cong \widehat{ty}$. By Definition 3.2.10, we have $\text{Erase}(e) = \widehat{e}$ such that $e \cong \widehat{e}$. Therefore, we have $\widehat{e}' = \text{sizeof}(\widehat{ty}) \cdot (\widehat{e})$.

Given $\widehat{e}' = \text{sizeof}(\widehat{ty}) \cdot (\widehat{e})$, we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{sizeof}(\widehat{ty}) \cdot (\widehat{e}))$.

Given $\text{sizeof}(\widehat{ty})$, we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{sizeof}(\widehat{ty})) \Downarrow'_{\widehat{ty}} (\widehat{\gamma},\ \widehat{\sigma},\ \square, \widehat{n}_1)$ by Vanilla C rule Size of Type such that $\widehat{n}_1 = \tau(\widehat{ty})$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\widehat{e} \cong e$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$ and $d_1 \cong \widehat{d}_1$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$, and therefore by Definition 3.2.18 and Definition 3.2.10 we have $n = \widehat{n}$.

Given $\widehat{n}_1$ and $\widehat{n}$, we have $\widehat{n}' = \widehat{n}_1 \cdot \widehat{n}$. Therefore, by Vanilla C rule Multiplication we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{sizeof}(\widehat{ty}) \cdot (\widehat{e}))$ $\Downarrow'_{\widehat{bm}}$ $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n}')$.

Given $\widehat{e}' = \text{sizeof}(\widehat{ty}) \cdot (\widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{sizeof}(\widehat{ty}))$ $\Downarrow'_{\widehat{ty}}$ $(\widehat{\gamma},\ \widehat{\sigma},\ \square, \widehat{n}_1)$, $\widehat{n}_1 = \tau(\widehat{ty})$, and $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ $\Downarrow'_{\widehat{d}_1}$ $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$, we have $\widehat{e}' = \tau(\widehat{ty}) \cdot \widehat{n} = \widehat{n}'$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$, $l = \widehat{l}$, and $(l, 0) \cong (\widehat{l}, 0)$.

Given $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n \cdot \tau(ty), \text{PermL}(\text{Freeable}, \text{void}*, \text{private}, n \cdot \tau(ty))\big]$, $n = \widehat{n}$, $\tau(\widehat{ty}) \cdot \widehat{n} = \widehat{n}'$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, $l = \widehat{l}$, $ty \cong \widehat{ty}$, and $\frac{n \cdot \tau(ty)}{\tau(ty)} = \frac{\widehat{n}'}{\tau(\widehat{ty})}$, by Lemma 3.2.8 we have $\widehat{\sigma}_2 = \widehat{\sigma}_1\big[\widehat{l} \rightarrow \big(\text{NULL},\ \text{void}*,\ \widehat{n}',$ $\text{PermL}(\text{Freeable}, \text{void}, \text{public}, \widehat{n}')\big)\big]$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{malloc}(\widehat{e}'))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}')$ $\Downarrow'_{\widehat{d}_1}$ $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}')$, $\widehat{l} = \phi()$, $\text{EncodePtr}(\text{void}*, [1,\ [(\widehat{l}_{default},$ $0)], [1], 1]) = \widehat{\omega}$, and $\widehat{\sigma}_2 = \widehat{\sigma}_1\big[\widehat{l} \rightarrow \big(\text{NULL},\ \text{void}*,\ \widehat{n}', \text{PermL}(\text{Freeable}, \text{void}, \text{public}, \widehat{n}')\big)\big]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square,$ $\text{malloc}(\widehat{e}))$ $\Downarrow'_{\widehat{mal}}$ $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ (\widehat{l}, 0))$ by Vanilla C rule Malloc.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ (\widehat{l}, 0))$. Therefore, we have $(\gamma, \sigma, \text{acc}, \text{pmalloc}(e,\ ty))$ $\Downarrow_{malp}$ $(\gamma, \sigma_2, \text{acc}, (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{malloc}(\widehat{e}))$ $\Downarrow'_{\widehat{mal}}$ $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,$ $(\widehat{l}, 0))$, $\Pi \cong_\psi \Sigma$, and $[malp, d_1] \cong [mal, bm, ty, \widehat{d}_1]$ by Definition 3.2.22.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{free}(e)) \Downarrow_{fre} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{free}(e)) \Downarrow_{fre} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Public Free, we have $(\gamma, \sigma, \text{acc}, e)$ $\Downarrow_{d_1}$ $(\gamma,$ $\sigma_1, \text{acc}, x)$, $\gamma(x) = (l, \text{public } bty*)$, $(bty = \text{int}) \vee (bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{void})$, $\text{acc} = 0$, and $\text{Free}(\sigma_1, l, \gamma) = \sigma_2$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \text{free}(\widehat{e}))$ such that $(\gamma, \sigma, \text{acc}, \text{free}(e)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{free}(\widehat{e}))$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi$ $(\widehat{\gamma},\ \widehat{\sigma})$ and $\text{free}(e) \cong_\psi \text{free}(\widehat{e})$. Given $(\gamma, \sigma, \text{acc}, \text{free}(e))$ $\Downarrow_{fre}$ $(\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{free}(e)$. Therefore, by Lemma 3.2.3 we have $\text{free}(e) \cong \text{free}(\widehat{e})$. By Definition 3.2.10 we have $\text{Erase}(\text{free}(e)) =$ $\text{free}(\text{Erase}(e))$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{x})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, x) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{x})$ and $d_1 \cong \widehat{d_1}$. Given $x \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{public } bty*)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, and $\text{public } bty* \cong \widehat{bty*}$ by Lemma 3.2.14. By Definition 3.2.13 we have $l \cong_\psi \widehat{l}$.

Given $\text{Free}(\sigma_1, l, \gamma) = \sigma_2$ and $l \cong_\psi \widehat{l}$, by Lemma 3.2.38 we have $\text{Free}(\widehat{\sigma}_1, \widehat{l}, \widehat{\gamma}) = \widehat{\sigma}_2$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \Box, \text{free}(\widehat{e}))$, $(\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$, and $\text{Free}(\widehat{\sigma}_1, \widehat{l}, \widehat{\gamma}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \Box, \text{free}(\widehat{e})) \Downarrow'_{\widehat{fre}} (\widehat{\gamma}, \widehat{\sigma}_2, \Box, \text{skip})$ by Vanilla C rule Free.

Given $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \Box, \text{skip})$. Therefore, we have $(\gamma, \sigma, \text{acc}, \text{free}(e)) \Downarrow_{fre} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \Box, \text{free}(\widehat{e})) \Downarrow'_{\widehat{fre}} (\widehat{\gamma}, \widehat{\sigma}_2, \Box, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $fre \cong \widehat{fre}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{pfree}(e)) \Downarrow_{frep} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{pfree}(e)) \Downarrow_{frep} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Private Free, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, $\gamma(x) = (l, \text{private } bty*)$, $\text{acc} = 0$, $(bty = \text{int}) \vee (bty = \text{float})$, and $\text{PFree}(\sigma_1, l) = \sigma_2$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \Box, \text{free}(\widehat{e}))$ such that $(\gamma, \sigma, \text{acc}, \text{pfree}(e)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \Box, \text{free}(\widehat{e}))$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $\text{pfree}(e) \cong_\psi \text{free}(\widehat{e})$. Given $(\gamma, \sigma, \text{acc}, \text{pfree}(e)) \Downarrow_{frep} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin \text{pfree}(e)$. Therefore, by Lemma 3.2.3 we have $\text{pfree}(e) \cong \text{free}(\widehat{e})$. By Definition 3.2.10 we have $\text{Erase}(\text{pfree}(e)) = \text{free}(\text{Erase}(e))$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{e})$ such that $(\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, x)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{x})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, x) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \Box, \widehat{x})$ and $d_1 \cong \widehat{d_1}$. Given $x \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, and $\text{private } bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given $\text{PFree}(\gamma,\ \sigma_1, l) = (\sigma_2, \bar{l}, \bar{j})$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, $l \cong_\psi \widehat{l}$, $\text{GetLocationSwap}(\bar{l}, \bar{j}) = \bar{l}'$, $\psi_2 = \psi[\bar{l}']$, and $\text{SwapMemory}(\widehat{\sigma}_2, \psi_2) = \widehat{\sigma}_3$, by Lemma 3.2.39 we have $\text{Free}(\widehat{\sigma}_1, \widehat{l}, \widehat{\gamma}) = \widehat{\sigma}_2$, such that $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{free}(\widehat{e}))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*})$, and $\text{Free}(\widehat{\sigma}_1,\ \widehat{l}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{free}(\widehat{e})) \Downarrow'_{\widehat{fre}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Free.

Given $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_2, \text{acc}, \text{skip}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, \text{pfree}(e)) \Downarrow_{frep} (\gamma,\ \sigma_2, \text{acc}, \text{skip}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{free}(\widehat{e})) \Downarrow'_{\widehat{fre}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_{\psi_2} \Sigma$, and $frep \cong \widehat{fre}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x = e) \Downarrow_{ds} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x = e) \Downarrow_{ds} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Declaration Assignment, we have $(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{d_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$, and $(\gamma_1, \sigma_1, \text{acc},\ x = e) \Downarrow_{d_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x} = \widehat{e})$ and ψ such that $(\gamma, \sigma, \text{acc}, ty\ x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x = e \cong_\psi \widehat{ty}\ \widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, ty\ x = e) \Downarrow_{ds} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin ty\ x = e$. Therefore, by Lemma 3.2.3 we have $ty\ x = e \cong \widehat{ty}\ \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x = e) = \text{Erase}(ty)\ \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(ty) = \widehat{ty}$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $ty \cong \widehat{ty}$ and $x = \widehat{x}$, by Definition 3.2.10 we have $ty\ x \cong \widehat{ty}\ \widehat{x}$. Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}) \cong_\psi (\gamma, \sigma, \text{acc}, ty\ x)$ by Definition 3.2.20. Given $(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{d_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ x) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$ and ψ_1 such that $(\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$. Given $\text{pfree}(e) \notin ty\ x$, we have $\psi_1 = \psi$ by Definition 3.2.11. By Definition 3.2.20 we have $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $x = \widehat{x}$ and $e \cong \widehat{e}$, by Definition 3.2.10 we have $x = e \cong \widehat{x} = \widehat{e}$. Given $(\gamma_1,\ \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, by Lemma 3.2.4 we have $(\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \widehat{x} = \widehat{e})$ such that $(\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \widehat{x} = \widehat{e}) \cong_\psi (\gamma_1, \sigma_1, \text{acc},\ x = e)$. Given $(\gamma_1, \sigma_1, \text{acc},\ x = e) \Downarrow_{d_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$, by the inductive hypothesis we have $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ x = \widehat{e}) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ and ψ_2 such that $(\gamma_1, \sigma_2, \text{acc}, \text{skip}) \cong_{\psi_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Given $\text{pfree}(e) \notin x = e$, we have $\psi_2 = \psi$ by Definition 3.2.11. By Definition 3.2.20, we have $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ x) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$, and $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ x = \widehat{e}) \Downarrow'_{\widehat{d_2}} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{ds}} (\widehat{\gamma}_1,\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Declaration Assignment.

Given $(\gamma_1,\ \sigma_2) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \text{acc},\ ty\ x = e) \Downarrow_{ds} (\gamma_1, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{ds}} (\widehat{\gamma}_1,\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $ds \cong \widehat{ds}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc},\ ty\ x[e_1]\ = e_2) \Downarrow_{das} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc},\ ty\ x = e) \Downarrow_{ds} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc},\ ty\ x) \Downarrow_d (\gamma_1, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_d (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ by SMC^2 rule Public Declaration, we have $(ty = \text{public}\ bty) \vee (ty = \text{char})$, $\text{acc} = 0$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\omega = \text{EncodeVal}(ty, \text{NULL})$, and $\sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$ such that $(\gamma,\ \sigma,\ \text{acc},\ ty\ x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \widehat{ty}\ \widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x \cong_\psi \widehat{ty}\ \widehat{x}$. Given $(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_d (\gamma_1, \sigma_1, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin ty\ x$. Therefore, by Lemma 3.2.3 we have $ty\ x \cong \widehat{ty}\ \widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x) = \text{Erase}(ty)\ \text{Erase}(x)$, $\text{Erase}(ty) = \widehat{ty}$ and $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$. Therefore, we have $ty \cong \widehat{ty}$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $x = \widehat{x}$, $l = \widehat{l}$, $ty \cong \widehat{ty}$, and $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty})]$ such that $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given $\omega = \text{EncodeVal}(ty, \text{NULL})$, and $ty \cong \widehat{ty}$, by Lemma 3.2.40 we have $\widehat{\omega} = \text{EncodeVal}(\widehat{bty}, \text{NULL})$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega,\ ty,\ 1,\ \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$, $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and $ty \cong \widehat{ty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{bty}, 1, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))]$ such that $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{bty}\ \widehat{x}), \widehat{l} = \phi(), \widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty})], \widehat{\omega} = \text{EncodeVal}(\widehat{bty}, \text{NULL})$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{bty},\ 1,$ $\text{PermL}(\text{Freeable},\ \widehat{bty}, \text{public}, 1))]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{bty}\ \widehat{x}) \Downarrow'_{\widehat{d}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square, \text{skip})$ by Vanilla C rule Declaration.

Given $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, by Definition 3.2.20 we have $(\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_d (\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong (\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{bty}\ \widehat{x}) \Downarrow'_{\widehat{d}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $d \cong \widehat{d}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_{d1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_{d1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ by SMC^2 rule Private Declaration, we have $((ty = bty) \vee (ty = \text{private}\ bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \text{private}\ bty)]$, $\omega = \text{EncodeVal}(ty, \text{NULL})$, and $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private}\ bty, 1, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, 1))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{ty}\ \widehat{x})$ such that $(\gamma,\ \sigma,\ \text{acc},\ ty\ x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x \cong_\psi \widehat{ty}\ \widehat{x}$. Given $(\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_{d1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin ty\ x$. Therefore, by Lemma 3.2.3 we have $ty\ x \cong \widehat{ty}\ \widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x) = \text{Erase}(ty)\ \text{Erase}(x)$, $\text{Erase}(ty) = \widehat{ty}$ and $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$. Therefore, we have $ty \cong \widehat{ty}$. Given $ty \cong \widehat{ty}$ and $((ty = \text{private}\ bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, by Definition 3.2.6 we have $bty \cong \widehat{bty}$. Therefore, we have $\text{private}\ bty \cong \widehat{bty}$ by Definition 3.2.6.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{private}\ bty)]$, $x = \widehat{x}$, $l = \widehat{l}$, $\text{private}\ bty$, and $(\gamma,\ \sigma) \cong (\widehat{\gamma},\ \widehat{\sigma})$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty})]$ such that $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given $\omega = \text{EncodeVal}(\text{private}\ bty, \text{NULL})$, and $\text{private}\ bty \cong \widehat{bty}$, by Lemma 3.2.40 we have $\widehat{\omega} = \text{EncodeVal}(\widehat{bty}, \text{NULL})$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private}\ bty, 1, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, 1))]$, $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{private}\ bty \cong \widehat{bty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{bty},\ 1, \text{PermL}(\text{Freeable},\ \widehat{bty}, \text{public}, 1))]$ such that $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{bty}\ \widehat{x}), \widehat{l} = \phi(), \widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty})], \widehat{\omega} = \text{EncodeVal}(\widehat{bty}, \text{NULL})$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{bty},\ 1,$

PermL(Freeable, $\widehat{bty}$, public, 1))], we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{bty}\ \widehat{x}) \Downarrow'_{\widehat{d}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square, \text{skip})$ by Vanilla C rule Declaration.

Given $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, by Definition 3.2.20 we have $(\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{d1} (\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{bty}\ \widehat{x}) \Downarrow'_{\widehat{d}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $d1 \cong \widehat{d}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{dp} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{dp} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ by SMC^2 rule Public Pointer Declaration, we have $(ty = \text{public}\ bty*) \vee ((ty = bty*) \wedge ((bty = \text{char}) \vee (bty = \text{void})))$, GetIndirection$(*) = i$, $\text{acc} = 0$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \text{public}\ bty*)]$, $\omega = \text{EncodePtr}(\text{public}\ bty*, [1, [(l_{default}, 0)], [1], i])$, and $\sigma_1 = \sigma[l \rightarrow (\omega, \text{public}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{ty}\ \widehat{x})$ such that $(\gamma, \sigma, \text{acc}, ty\ x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x \cong_\psi \widehat{ty}\ \widehat{x}$. Given $(\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_{dp} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin ty\ x$. Therefore, by Lemma 3.2.3 we have $ty\ x \cong \widehat{ty}\ \widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x) = \text{Erase}(ty)\ \text{Erase}(x)$, $\text{Erase}(ty) = \widehat{ty}$ and $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$. Therefore, we have $ty \cong \widehat{ty}$ such that $* = *$.

Given GetIndirection$(*) = i$ and $* = *$, by Lemma 3.2.49 we have GetIndirection$(*) = \widehat{i}$ such that $i = \widehat{i}$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{public}\ bty*)]$, $x = \widehat{x}$, $l = \widehat{l}$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $\text{public}\ bty* \cong \widehat{bty}*$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty}*)]$ such that $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given $\omega = \text{EncodePtr}(\text{public}\ bty*, [1, [(l_{default}, 0)], [1], i])$, $\text{public}\ bty* \cong \widehat{bty}*$, $i = \widehat{i}$, and $[1, [(l_{default}, 0)], [1], i] \cong_\psi [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}]$, by Lemma 3.2.42 we have $\widehat{\omega} = \text{EncodePtr}(\widehat{ty}*, [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}])$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, \text{public}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1))]$, $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{public}\ bty* \cong \widehat{bty}*$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{ty},\ 1, \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, 1))]$ such that $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x})$, $(\widehat{ty} = \widehat{bty}*)$, $\text{GetIndirection}(*) = \widehat{i}$, $\widehat{l} = \phi()$, $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{ty})]$, $\widehat{\omega} = \text{EncodePtr}(\widehat{ty}*, [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}])$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{ty},\ 1, \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, 1))]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}) \Downarrow'_{\widehat{dp}} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$ by Vanilla C rule Pointer Declaration.

Given $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, by Definition 3.2.20 we have $(\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_{dp} (\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{ty}\ \widehat{x}) \Downarrow'_{\widehat{dp}} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $dp \cong \widehat{dp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_{dp1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_{dp1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ by SMC^2 rule Private Pointer Declaration, we have $((ty = bty*) \vee (ty = \text{private}\ bty*)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $\text{GetIndirection}(*) = i$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \text{private}\ bty*)]$, $\omega = \text{EncodePtr}(\text{private}\ bty*, [1, [(l_{default}, 0)], [1], i])$, and $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, 1))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{ty}\ \widehat{x})$ such that $(\gamma,\ \sigma,\ \text{acc},\ ty\ x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x \cong_\psi \widehat{ty}\ \widehat{x}$. Given $(\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_{dp1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin ty\ x$. Therefore, by Lemma 3.2.3 we have $ty\ x \cong \widehat{ty}\ \widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x) = \text{Erase}(ty)\ \text{Erase}(x)$, $\text{Erase}(ty) = \widehat{ty}$ and $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$. Therefore, we have $ty \cong \widehat{ty}$ such that $* = *$.

Given $((ty = bty*) \vee (ty = \text{private}\ bty*)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$ and $ty \cong \widehat{ty}$, we have $\text{private}\ bty* \cong \widehat{bty}*$ and $bty \cong \widehat{bty}$ by Definition 3.2.6.

Given $\text{GetIndirection}(*) = i$ and $* = *$, by Lemma 3.2.49 we have $\text{GetIndirection}(*) = \widehat{i}$ such that $i = \widehat{i}$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{private}\ bty*)]$, $x = \widehat{x}$, $l = \widehat{l}$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $\text{private}\ bty* \cong \widehat{bty}*$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty}*)]$ such that $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given $\omega = \text{EncodePtr}(\text{private}\ bty*, [1, [(l_{default}, 0)], [1], i])$, $i = \widehat{i}$, $\text{private}\ bty* \cong \widehat{bty}*$, and $[1, [(l_{default}, 0)], [1], i] \cong_\psi [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}]$, by Lemma 3.2.42 we have $\widehat{\omega} = \text{EncodePtr}(\widehat{bty}*, [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}])$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))]$, $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma})$, private $bty* \cong \widehat{bty*}$, $l = \widehat{l}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, 1))]$ such that $(\gamma_1, \sigma_1) \cong (\widehat{\gamma}_1, \widehat{\sigma}_1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$, $(\widehat{ty} = \widehat{bty*})$, $\text{GetIndirection}(*) = \widehat{i}, \widehat{l} = \phi()$, $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \widehat{bty*})]$, $\widehat{\omega} = \text{EncodePtr}(\widehat{bty*}, [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}])$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, 1))]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}) \Downarrow'_{\widehat{dp}} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$ by Vanilla C rule Pointer Declaration.

Given $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$, by Definition 3.2.20 we have $(\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{dp1} (\gamma_1, \sigma_1, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}) \Downarrow'_{\widehat{dp}} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $dp1 \cong \widehat{dp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_r (\gamma, \sigma, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_r (\gamma, \sigma, \text{acc}, v)$ by SMC^2 rule Read Public Variable, we have $\gamma(x) = (l, \text{public } bty)$, $\sigma(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1))$, and
$\text{DecodeVal}(\text{public } bty, 1, \omega) = v$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$ such that $(\gamma, \sigma, \text{acc}, x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{public } bty)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by public $bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, bty, \text{public}, 1))$ where $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodeVal}(\text{public } bty, 1, \omega) = v$, public $bty \cong \widehat{bty}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, 1, \widehat{\omega}) = \widehat{v}$ and $v \cong_\psi \widehat{v}$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))$, and $\text{DecodeVal}(\widehat{bty}, 1, \widehat{\omega}) = \widehat{v}$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{r}} (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{v})$ by Vanilla C rule Read Variable.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.20 we have $(\gamma, \sigma, \text{acc}, v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{v})$. Therefore, we have $(\gamma, \sigma, \text{acc}, x) \Downarrow_r (\gamma, \sigma, \text{acc}, v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{x}) \Downarrow'_{\widehat{r}} (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{v})$, $\Pi \cong_\psi \Sigma$, and $r \cong \widehat{r}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{r1} (\gamma, \sigma, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{r1} (\gamma, \sigma, \text{acc}, v)$ by SMC^2 rule Read Public Variable, we have $\gamma(x) = (l,\ \text{private}\ \ bty)$, $\sigma(l) = (\omega,\ \text{private}\ bty,\ 1,\ \text{PermL}(\text{Freeable}, \text{private}\ \ bty,\ \text{private},\ 1))$, and $\text{DecodeVal}(\text{private}\ bty, 1, \omega) = v$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{x})$ such that $(\gamma, \sigma, \text{acc}, x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private}\ bty)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by private $bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{private}\ bty, 1, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, 1))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable},\ bty, \text{public}, 1))$ where $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodeVal}(\text{private}\ bty, 1, \omega) = v$, private $bty \cong \widehat{bty}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, 1, \widehat{\omega}) = \widehat{v}$ and $v \cong_\psi \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty},\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty},\ \text{public},\ 1))$, and $\text{DecodeVal}(\widehat{bty}, 1,\ \widehat{\omega}) = \widehat{v}$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{x}) \Downarrow'_{\widehat{r}} (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{v})$ by Vanilla C rule Read Variable.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.20 we have $(\gamma, \sigma, \text{acc}, v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{v})$. Therefore, we have $(\gamma, \sigma, \text{acc}, x) \Downarrow_{r1} (\gamma, \sigma, \text{acc}, v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{x}) \Downarrow'_{\widehat{r}} (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{v})$, $\Pi \cong_\psi \Sigma$, and $r1 \cong \widehat{r}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_w (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_w (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Public Write Variable, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, v)$, $v \neq \text{skip}$, $\text{acc} = 0$, $\gamma(x) = (l, \text{public}\ bty)$, and $\text{UpdateVal}(\sigma_1, l, v, \text{public}\ bty) = \sigma_2$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$ such that $(\gamma, \sigma, \text{acc}, x = e) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_{\psi} \widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$ and $d_1 \cong \widehat{d_1}$.

Given $v \neq \text{skip}$ and $v \cong_{\psi} \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$ and by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_{\psi} \widehat{v}$.

Given $\gamma(x) = (l, \text{public } bty)$, $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by public $bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\text{UpdateVal}(\sigma_1, l, v, \text{public } bty) = \sigma_2$, $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$, $l = \widehat{l}$, $v \cong_{\psi} \widehat{v}$, and public $bty \cong \widehat{bty}$, by Lemma 3.2.52 we have $\text{UpdateVal}(\widehat{\sigma}_1,\ \widehat{l},\ \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$ such that $(\gamma,\ \sigma_2) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty})$, and $\text{UpdateVal}(\widehat{\sigma}_1, \widehat{l}, \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{w}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Write.

Given $(\gamma,\ \sigma_2) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_2, \text{acc}, \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, x = e) \Downarrow_{w} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{w}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \text{skip})$, $\Pi \cong_{\psi} \Sigma$, and $w \cong \widehat{w}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w2} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w2} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Private Write Variable, we have $\text{Label}(e, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private } bty)$, and $\text{UpdateVal}(\sigma_1, l, v, \text{private } bty) = \sigma_2$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$ and ψ such that $(\gamma, \sigma, \text{acc}, x = e) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_{\psi} \widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w2} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2

we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$ by Definition 3.2.20. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$ and $d_1 \cong \widehat{d}_1$. Given $v \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private } bty)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by private $bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\text{UpdateVal}(\sigma_1, l, v, \text{private } bty) = \sigma_2$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l = \widehat{l}$, $v \cong_\psi \widehat{v}$, and private $bty \cong \widehat{bty}$, by Lemma 3.2.52 we have $\text{UpdateVal}(\widehat{\sigma}_1,\ \widehat{l},\ \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty})$, and $\text{UpdateVal}(\widehat{\sigma}_1, \widehat{l}, \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{w}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Write.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc},\ x = e) \Downarrow_{w2} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{w}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $w2 \cong \widehat{w}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w1} (\gamma,\ \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w1} (\gamma,\ \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Write Private Variable Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma,\ \sigma_1,\ \text{acc},\ v)$, $v \neq \text{skip}$, $\gamma(x) = (l,\ \text{private } bty)$, and $\text{UpdateVal}(\sigma_1,\ l,\ \text{encrypt}(v), \text{private } bty) = \sigma_2$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$ and ψ such that $(\gamma, \sigma, \text{acc}, x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w1} (\gamma,\ \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e)\ \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$ and $d_1 \cong \widehat{d_1}$. Given $v \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l,\ \text{private}\ bty)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by private $bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\text{UpdateVal}(\sigma_1,\ l,\ \text{encrypt}(v), \text{private}\ bty) = \sigma_2$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l = \widehat{l}$, $v \cong_\psi \widehat{v}$, and private $bty \cong \widehat{bty}$, by Lemma 3.2.52 we have $\text{UpdateVal}(\widehat{\sigma}_1,\ \widehat{l},\ \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})\ \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty})$, and $\text{UpdateVal}(\widehat{\sigma}_1, \widehat{l}, \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{w}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Write.

Given $\Pi \triangleright (\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w1} (\gamma,\ \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{w}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $w1 \cong \widehat{w}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x)\ \Downarrow_{rp} (\gamma, \sigma, \text{acc}, (l_1, \mu_1))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x)\ \Downarrow_{rp} (\gamma, \sigma, \text{acc}, (l_1, \mu_1))$ by SMC^2 rule Public Pointer Read Single Location, we have $\gamma(x) = (l,\ \text{public}\ bty*)$, $\sigma(l) = (\omega,\ \text{public}\ bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{public}\ bty*,\ \text{public},\ 1))$, and $\text{DecodePtr}(\text{public}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x})$ and ψ such that $(\gamma, \sigma, \text{acc}, x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l,\ \text{public}\ bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by public $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$ $\text{public } bty* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*), \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$, and $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{rp}} (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{l}_1, \widehat{\mu}_1))$ by Vanilla C rule Pointer Read Location.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, by Definition 3.2.20 we have $(\gamma, \sigma, \text{acc}, (l_1, \mu_1)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{l}_1, \widehat{\mu}_1))$. Therefore, we have $(\gamma, \sigma, \text{acc}, x) \Downarrow_{rp} (\gamma, \sigma, \text{acc}, (l_1, \mu_1)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{rp}} (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{l}_1, \widehat{\mu}_1))$, $\Pi \cong_\psi \Sigma$, and $rp \cong \widehat{rp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{rp2} (\gamma, \sigma, \text{acc}, (l_1, \mu_1))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{rp2} (\gamma, \sigma, \text{acc}, (l_1, \mu_1))$ by SMC^2 rule Private Pointer Read Single Location, we have $\gamma(x) = (l, \text{private } bty*)$, $\sigma(l) = (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))$, and $\text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$ and ψ such that $(\gamma, \sigma, \text{acc}, x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private } bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{private } bty*, 1\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, $\text{private } bty* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*), \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$, and $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{rp}} (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{l}_1, \widehat{\mu}_1))$ by Vanilla C rule Pointer Read Location.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu}_1)$, by Definition 3.2.20 we have $(\gamma, \sigma, \text{acc}, (l_1, \mu_1)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{l_1}, \widehat{\mu}_1))$. Therefore, we have $(\gamma, \sigma, \text{acc}, x) \ \Downarrow_{rp2} \ (\gamma, \sigma, \text{acc}, (l_1, \mu_1)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{rp}} (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{l_1}, \widehat{\mu}_1))$, $\Pi \cong_\psi \Sigma$, and $rp2 \cong \widehat{rp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma,\ \sigma, \text{acc},\ x) \Downarrow_{rp1} (\gamma, \sigma, \text{acc}, [\alpha, \bar{l}, \bar{j}, i])$

Given $\Pi \triangleright (\gamma,\ \sigma, \text{acc},\ x) \Downarrow_{rp1} (\gamma, \sigma, \text{acc}, [\alpha, \bar{l}, \bar{j}, i])$ by SMC^2 rule Private Pointer Read Multiple Locations, we have $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, and $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x})$ and ψ such that $(\gamma, \sigma, \text{acc}, x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]$, $\text{private } bty* \cong \widehat{bty}*$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha,\ \bar{l}, \bar{j},\ i], \text{private } bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1, (\widehat{l_1}, \widehat{\mu}_1),\ [1],\ 1]$ where $[\alpha, \bar{l}, \bar{j}, i] \cong_\psi [1, (\widehat{l_1}, \widehat{\mu}_1), [1], i]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*), \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$, and $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l_1}, \widehat{\mu}_1)],\ [1],\ \widehat{i}]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{rp}} (\widehat{\gamma},\ \widehat{\sigma},\ \square, (\widehat{l_1}, \widehat{\mu}_1))$ by Vanilla C rule Pointer Read Location.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $[\alpha, \bar{l}, \bar{j}, i] \cong_\psi (\widehat{l_1}, \widehat{\mu}_1)$, by Definition 3.2.20 we have $(\gamma, \sigma, \text{acc}, [\alpha, \bar{l}, \bar{j}, i]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{l_1}, \widehat{\mu}_1))$. Therefore, we have $(\gamma,\ \sigma, \text{acc},\ x) \Downarrow_{rp1} (\gamma, \sigma, \text{acc}, [\alpha, \bar{l}, \bar{j}, i]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{rp}} (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{l_1}, \widehat{\mu}_1))$, $\Pi \cong_\psi \Sigma$, and $rp1 \cong \widehat{rp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Public Pointer Write Single Location, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, (l_e, \mu_e))$, $\gamma(x) = (l, \text{public } bty*)$, $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\text{acc} = 0$, $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, and $\text{UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \text{public } bty*) = (\sigma_2, 1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$ such that $(\gamma, \sigma, \text{acc}, x = e) \cong (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong (\widehat{\gamma}, \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, (l_e, \mu_e))$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, (l_e, \mu_e)) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$ and $d_1 \cong \widehat{d}_1$. Given $(l_e, \mu_e) \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $(l_e, \mu_e) \cong_\psi (\widehat{l}_e, \widehat{\mu}_e)$.

Given $\gamma(x) = (l, \text{public } bty*)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by public $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, public $bty* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $\text{UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \text{public } bty*) = (\sigma_2, 1)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, public $bty* \cong \widehat{bty}*$, and $[1, [(l_e, \mu_e)], [1], i] \cong_\psi [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}]$, by Lemma 3.2.54 we have $\text{UpdatePtr}(\widehat{\sigma}, (\widehat{l}, 0), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, and $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}, 0), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wp}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Pointer Assign Location.

Given $(\gamma,\ \sigma_2) \cong_\psi (\gamma,\ \sigma_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\gamma, \sigma_2, \text{acc}, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, x = e) \Downarrow_{wp} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp} (\gamma, \sigma_2, \text{acc}, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wp \cong \widehat{wp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x\ = e) \Downarrow_{wp2} (\gamma, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ x = e)\ \Downarrow_{wp2}\ (\gamma,\ \sigma_1,\ \text{acc},\ \text{skip})$ by SMC^2 rule Private Pointer Write Multiple Locations, we have $(\gamma,\ \sigma,\ \text{acc},\ e)\ \Downarrow_{d_1}\ (\gamma,\ \sigma_1,\ \text{acc},\ [\alpha, \bar{l}, \bar{j}, i])$, $\gamma(x) = (l,\ \text{private}\ bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, and $\text{UpdatePtr}(\sigma_1, (l, 0), [\alpha, \bar{l}, \bar{j}, i], \text{private}\ bty*) = (\sigma_2, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \text{acc},\ x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \text{acc},\ x = e)\ \Downarrow_{wp2}\ (\gamma,\ \sigma_1,\ \text{acc},\ \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \widehat{x} = \widehat{e}$ where $x = \widehat{x}$ and $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}, \bar{j}, i])$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}, \bar{j}, i]) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$ and $d_1 \cong \widehat{d}_1$. Given $[\alpha, \bar{l}, \bar{j}, i] \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $[\alpha, \bar{l}, \bar{j}, i] \cong_\psi (\widehat{l}_e, \widehat{\mu}_e)$.

Given $\gamma(x) = (l, \text{private}\ bty*)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\text{UpdatePtr}(\sigma, (l, 0), [\alpha, \bar{l}, \bar{j}, i]) = (\sigma_1, 1)$, $[\alpha, \bar{l}, \bar{j}, i] \cong_\psi (\widehat{l}_e, \widehat{\mu}_e)$, private $bty* \cong \widehat{bty}*$, and $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, by Lemma 3.2.54 we have $\text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_e,\ \widehat{\mu}_e)],\ [1],\ \widehat{i}],\ \widehat{bty}*) = (\widehat{\sigma}_2, 1)$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$, $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ \widehat{i}]$, and $\text{UpdatePtr}(\widehat{\sigma}_1,\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_e, \widehat{\mu}_e)],\ [1],\ \widehat{i}], \widehat{bty}*) = (\widehat{\sigma}_2,\ 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Pointer Assign Location.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we

have $(\gamma,\ \sigma,\ \text{acc},\ x = [\alpha, \bar{l}, \bar{j}, i]) \Downarrow_{wp2} (\gamma, \sigma_1, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wp2 \cong \widehat{wp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp1} (\gamma,\ \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp1} (\gamma,\ \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Private Pointer Assign Single Location, we have $\gamma(x) = (l,\ \text{private}\ bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma,\ \sigma_1,\ \text{acc},\ (l_e, \mu_e))$, $\sigma_1(l) = (\omega, \text{private}\ bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private}\ bty*, \alpha, \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]$, $\text{Label}(e, \gamma) = \text{public}$, and $\text{UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1],\ i], \text{private}\ bty*) = (\sigma_2, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \text{acc},\ x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \text{acc},\ x = e) \Downarrow_{wp1} (\gamma,\ \sigma_2,\ \text{acc},\ \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l,\ \text{private}\ bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private}\ bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, (l_e, \mu_e))$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, (l_e, \mu_e)) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$ and $d_1 \cong \widehat{d}_1$. Given $(l_e, \mu_e) \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l_e, \mu_e) \cong_\psi (\widehat{l}_e, \widehat{\mu}_e)$.

Given $\sigma_1(l) = (\omega, \text{private}\ bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*, 1, \text{PermL}(\text{Freeable},\ \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private}\ \ bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]$, $\text{private}\ bty* \cong \widehat{bty}*$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha, \bar{l}, \bar{j}, i], \text{private}\ bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha, \bar{l}, \bar{j}, i] \cong_\psi [1, (\widehat{l}_1, \widehat{\mu}_1), [1], i]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $\text{UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \text{private}\ bty*) = (\sigma_2, 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, $\text{private}\ bty* \cong \widehat{bty}*$, and $[1, [(l_e, \mu_e)], [1], i] \cong_\psi [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}]$, by Lemma 3.2.54 we have $\text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0), [1,\ [(\widehat{l}_e, \widehat{\mu}_e)],\ [1],\ \widehat{i}], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l_e}, \widehat{\mu}_e))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*})$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,$ $\mathrm{PermL}(\mathrm{Freeable},\ \widehat{bty*},\ \mathrm{public},\ 1))$, $\mathrm{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ \widehat{i}]$, and $\mathrm{UpdatePtr}(\widehat{\sigma}_1,$ $(\widehat{l}, 0),\ [1,\ [(\widehat{l_e}, \widehat{\mu}_e)],\ [1],\ \widehat{i}], \widehat{bty*}) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wp}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \mathrm{skip})$ by Vanilla C rule Pointer Assign Location.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_2, \mathrm{acc}, \mathrm{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \mathrm{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \mathrm{acc}, x = e) \Downarrow_{wp1} (\gamma,\ \sigma_2, \mathrm{acc}, \mathrm{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \mathrm{skip})$, $\Pi \cong_\psi \Sigma$, and $wp1 \cong \widehat{wp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, *x) \Downarrow_{rdp} (\gamma, \sigma, \mathrm{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, *x) \Downarrow_{rdp} (\gamma, \sigma, \mathrm{acc}, v)$ by SMC^2 rule Private Pointer Dereference Single Location, we have $\gamma(x) = (l,\ \mathrm{public}\ bty*)$, $\sigma(l) = (\omega, \mathrm{public}\ bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{public}\ bty*, \mathrm{public}, 1))$, $\mathrm{DecodePtr}(\mathrm{public}\ bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, and $\mathrm{DerefPtr}(\sigma, \mathrm{public}\ bty, (l_1, \mu_1)) = (v, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x})$ and ψ such that $(\gamma, \sigma, \mathrm{acc}, *x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x \cong_\psi *\widehat{x}$. Given $(\gamma, \sigma, \mathrm{acc}, *x) \Downarrow_{rdp} (\gamma, \sigma, \mathrm{acc}, v)$, by Lemma 3.2.2 we have $(l, \mu) \notin *x$. Therefore, by Lemma 3.2.3 we have $*x \cong *\widehat{x}$. By Definition 3.2.10 we have $\mathrm{Erase}(*x) = *\widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \mathrm{public}\ bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by $\mathrm{public}\ bty* \cong \widehat{bty*}$ by Lemma 3.2.14. Therefore, by Definition 3.2.6 we have $bty \cong \widehat{bty}$.

Given $\sigma(l) = (\omega, \mathrm{public}\ bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{public}\ bty*, \mathrm{public}, 1))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty*}, \mathrm{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\mathrm{DecodePtr}(\mathrm{public}\ bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, $\mathrm{public}\ bty* \cong \widehat{bty*}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\mathrm{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \mu_1)],\ [1],\ 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \mu_1)$.

Given $\mathrm{DerefPtr}(\sigma, \mathrm{public}\ bty, (l_1, \mu_1)) = (v, 1)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, $bty \cong \widehat{bty}$, and $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$, by Lemma 3.2.58 we have $\mathrm{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{v}, 1)$ such that $v \cong_\psi \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*}), \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \mathrm{PermL}(\mathrm{Freeable},\ \widehat{bty*},\ \mathrm{public},\ 1)), \mathrm{DecodeVal}(\widehat{bty*},$

$1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$, and $\text{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{v}, 1)$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x}) \Downarrow'_{\widehat{rdp}} (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{v})$ by Vanilla C rule Pointer Dereference.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.20 we have $(\gamma, \sigma, \text{acc}, v) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{v})$. Therefore, we have $(\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp} (\gamma, \sigma, \text{acc}, v) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x}) \Downarrow'_{\widehat{rdp}} (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{v})$, $\Pi \cong_\psi \Sigma$, and $rdp \cong \widehat{rdp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp1} (\gamma, \sigma, \text{acc}, (l_2, \mu_2))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp1} (\gamma, \sigma, \text{acc}, (l_2, \mu_2))$ by SMC^2 rule Public Pointer Dereference Single Location Higher Level Indirection, we have $\gamma(x) = (l, \text{public } bty*)$, $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, $i > 1$, and $\text{DerefPtrHLI}(\sigma, \text{public } bty*, (l_1, \mu_1)) = ([1, [(l_2, \mu_2)], [1], i-1], 1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x})$ and ψ such that $(\gamma, \sigma, \text{acc}, *x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $*x \cong_\psi *\widehat{x}$. Given $(\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp1} (\gamma, \sigma, \text{acc}, (l_2, \mu_2))$, by Lemma 3.2.2 we have $(l, \mu) \notin *x$. Therefore, by Lemma 3.2.3 we have $*x \cong *\widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(*x) = *\widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{public } bty*)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by public $bty* \cong \widehat{bty}*$ by Lemma 3.2.14. Therefore, by Definition 3.2.6 we have $bty \cong \widehat{bty}$.

Given $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, 0)], [1], i]$, public $bty* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], \widehat{i}]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $i > 1$ and $i = \widehat{i}$, we have $\widehat{i} > 1$.

Given $\text{DerefPtrHLI}(\sigma, \text{public } bty*, (l_1, \mu_1)) = ([1, [(l_2, \mu_2)], [1], i-1], 1)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, $bty \cong \widehat{bty}$, and $i = \widehat{i}$, by Lemma 3.2.59, we have $\text{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty}*, (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1], 1)$ such that $[1, [(l_2, \mu_2)], [1], i-1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1]$ and $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*), \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1)), \text{DecodePtr}(\widehat{bty}*,$

$1, \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ \widehat{i}], \widehat{i} > 1$, and $\text{DerefPtrHLI}(\widehat{\sigma},\ \widehat{bty}*,\ (\widehat{l}_1,\ \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i} - 1], 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x}) \Downarrow'_{\widehat{rdp1}} (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{l}_2, \widehat{\mu}_2))$ by Vanilla C rule Pointer Dereference Higher Level Indirection.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma, \text{acc}, (l_2, \mu_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{l}_2, \widehat{\mu}_2))$. Therefore, we have $(\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp1} (\gamma, \sigma, \text{acc}, (l_2, \mu_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x}) \Downarrow'_{\widehat{rdp1}} (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{l}_2, \widehat{\mu}_2))$, $\Pi \cong_\psi \Sigma$, and $rdp1 \cong \widehat{rdp1}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp2} (\gamma, \sigma, \text{acc}, v)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ *x) \Downarrow_{rdp2} (\gamma,\ \sigma,\ \text{acc},\ v)$ by SMC^2 rule Private Pointer Dereference, we have $\gamma(x) = (l, \text{private } bty*), (bty = \text{int}) \vee (bty = \text{float}), \sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, and $\text{Retrieve_vals}(\alpha, \bar{l}, \bar{j}, \text{private } bty, \sigma) = (v, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x})$ and ψ such that $(\gamma, \sigma, \text{acc}, *x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x \cong_\psi *\widehat{x}$. Given $(\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp2} (\gamma, \sigma, \text{acc}, v)$, by Lemma 3.2.2 we have $(l, \mu) \notin *x$. Therefore, by Lemma 3.2.3 we have $*x \cong *\widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(*x) = *\widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private } bty* \cong \widehat{bty}*$ by Lemma 3.2.14. Therefore, by Definition 3.2.6 we have $bty \cong \widehat{bty}$.

Given $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ 1]$, $\text{private } bty* \cong \widehat{bty}*$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha, \bar{l}, \bar{j}, 1], \text{private } bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha, \bar{l}, \bar{j}, 1] \cong_\psi [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$.

Given $\text{Retrieve_vals}(\alpha, \bar{l}, \bar{j}, \text{private } bty, \sigma) = (v, 1)$, $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ 1],\ \text{private } bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $\text{private } bty \cong \widehat{bty}$, by Lemma 3.2.60 we have $\text{DerefPtr}(\widehat{\sigma},\ \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{v}, 1)$ such that $v \cong_\psi \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*), \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$, $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1, \mu_1)],\ [1],\ 1]$, and $\text{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{v}, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x}) \Downarrow'_{\widehat{rdp}} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{v})$

by Vanilla C rule Pointer Dereference.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.20 we have $(\gamma,\ \sigma,\ \mathrm{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$. Therefore, we have $(\gamma,\ \sigma,\ \mathrm{acc},\ *x) \Downarrow_{rdp2} (\gamma,\ \sigma,\ \mathrm{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'_{\widehat{rdp}} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$, $\Pi \cong_\psi \Sigma$, and $rdp2 \cong \widehat{rdp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, *x) \Downarrow_{rdp3} (\gamma,\ \sigma, \mathrm{acc},\ [\alpha', \bar{l}', \bar{j}', i-1])$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \mathrm{acc},\ *x) \Downarrow_{rdp3} (\gamma,\ \sigma, \mathrm{acc},\ [\alpha', \bar{l}', \bar{j}', i-1])$ by SMC^2 rule Private Pointer Dereference Higher Level Indirection, we have $\gamma(x) = (l, \mathrm{private}\ bty*)$, $(bty = \mathrm{int}) \vee (bty = \mathrm{float})$, $\sigma(l) = (\omega,\ \mathrm{private}\ bty*,\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty*, \mathrm{private}, \alpha))$, $\mathrm{DecodePtr}(\mathrm{private}\ bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$, $i > 1$, and $\mathrm{DerefPrivPtr}(\alpha, \bar{l}, \bar{j}, \mathrm{private}\ bty*, \sigma) = ((\alpha', \bar{l}', \bar{j}'), 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x})$ and ψ such that $(\gamma, \sigma, \mathrm{acc}, *x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x \cong_\psi *\widehat{x}$. Given $(\gamma,\ \sigma,\ \mathrm{acc},\ *x) \Downarrow_{rdp3} (\gamma,\ \sigma, \mathrm{acc},\ [\alpha',\ \bar{l}',\ \bar{j}',\ i-1])$, by Lemma 3.2.2 we have $(l, \mu) \notin *x$. Therefore, by Lemma 3.2.3 we have $*x \cong *\widehat{x}$. By Definition 3.2.10 we have $\mathrm{Erase}(*x) = *\widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \mathrm{private}\ bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by $\mathrm{private}\ bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega,\ \mathrm{private}\ bty*,\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable},\ \mathrm{private}\ bty*,\ \mathrm{private},\ \alpha))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \mathrm{PermL}(\mathrm{Freeable},\ \widehat{bty*},\ \mathrm{public},\ 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\mathrm{DecodePtr}(\mathrm{private}\ \ bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]$, $\mathrm{private}\ bty* \cong \widehat{bty*}$, $\omega \cong_\psi \widehat{\omega}$, and $\mathrm{DeclassifyPtr}([\alpha, \bar{l}, \bar{j}, i], \mathrm{private}\ bty*) = (l_1, 0)$, by Lemma 3.2.45 we have $\mathrm{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[\alpha, \bar{l}, \bar{j}, i] \cong_\psi [1, (\widehat{l}_1, 0), [1], i]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$ and $i = \widehat{i}$.

Given $i > 1$ and $i = \widehat{i}$, we have $\widehat{i} > 1$.

Given $\mathrm{DerefPrivPtr}(\alpha,\ \bar{l},\ \bar{j},\ \mathrm{private}\ \ bty*,\ \sigma) = ((\alpha',\ \bar{l}',\ \bar{j}'),\ 1)$, $i = \widehat{i}$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $\mathrm{DeclassifyPtr}([\alpha, \bar{l}, \bar{j}, i], \mathrm{private}\ bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, $\mathrm{DeclassifyPtr}([\alpha',\ \bar{l}',\ \bar{j}',\ i-1],\ \mathrm{private}\ bty*) = (l_2, \mu_2)$, and $\mathrm{private}\ bty* \cong \widehat{bty*}$, by Lemma 3.2.57 we have $\mathrm{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty*}, (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1], 1)$ such that $[\alpha', \bar{l}', \bar{j}', i-1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1]$ and $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*}), \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*}, 1, \text{PermL}(\text{Freeable},\ \widehat{bty*}, \text{public}, 1)), \text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \mu_1)],\ [1],\ \widehat{i}], \widehat{i} > 1, \widehat{\sigma}(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty*},\ 1, \text{PermL}(\text{Freeable},\ \widehat{bty*},\ \text{public},\ 1))$, and $\text{DecodeVal}(\widehat{bty*},\ 1,\ \widehat{\omega}_1) = [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i} - 1]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x}) \Downarrow'_{\widehat{rdp1}} (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{l}_2, \widehat{\mu}_2))$ by Vanilla C rule Pointer Dereference Higher Level Indirection.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $[\alpha', \bar{l}', \bar{j}', i-1] \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma, \text{acc}, [\alpha', \bar{l}', \bar{j}', i-1]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, (\widehat{l}_2, \widehat{\mu}_2))$. Therefore, we have $(\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp3} (\gamma,\ \sigma, \text{acc}, [\alpha', \bar{l}', \bar{j}', i-1]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x}) \Downarrow'_{\widehat{rdp1}} (\widehat{\gamma},\ \widehat{\sigma},\ \square, (\widehat{l}_2, \widehat{\mu}_2))$, $\Pi \cong_\psi \Sigma$, and $rdp3 \cong \widehat{rdp1}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ *x =\ e) \Downarrow_{wdp} (\gamma,\ \sigma_2,\ \text{acc},\ \text{skip})$ by SMC^2 rule Public Pointer Dereference Write Public Value, we have $(\gamma,\ \sigma,\ \text{acc},\ e) \Downarrow_{d_1} (\gamma,\ \sigma_1,\ \text{acc},\ v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{public } bty*)$, $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1,\ \mu_1)],\ [1],\ 1]$, $\text{UpdateOffset}(\sigma_1, (l_1, \mu_1), v, \text{publ}$ $bty) = (\sigma_2, 1)$, $\text{Label}(e, \gamma) = \text{public}$, and $\text{acc} = 0$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e})$ and ψ such that $(\gamma, \sigma, \text{acc}, *x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x = e \cong_\psi *\widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, *x =\ e) \Downarrow_{wdp} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin *x =\ e$. Therefore, by Lemma 3.2.3 we have $*x =\ e \cong *\widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(*x = e) = *\widehat{x} = \text{Erase}(e)$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$ and $d_1 \cong \widehat{d}_1$. Given $v \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{public } bty*)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by public $bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by

Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, $\text{public } bty* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$.

Given $\text{UpdateOffset}(\sigma_1, (l_1, \mu_1), v, \text{public } bty) = (\sigma_2, 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$, $\text{public } bty \cong \widehat{bty}$, and $v \cong_\psi \widehat{v}$, by Lemma 3.2.53 we have $\text{UpdateOffset}(\widehat{\sigma}_1,\ (\widehat{l}_1,\ \widehat{\mu}_1),\ \widehat{v},\ \widehat{bty}) = (\widehat{\sigma}_2,\ 1)$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$. $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$, $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ 1]$, and $\text{UpdateOffset}(\widehat{\sigma}_1,\ (\widehat{l}_1,\ \widehat{\mu}_1),\ \widehat{v},\ \widehat{bty}) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wdp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Pointer Dereference Write Value.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc},\ *x =\ e) \Downarrow_{wdp} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wdp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wdp \cong \widehat{wdp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp1} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp1} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Public Pointer Dereference Write Higher Level Indirection, we have $\text{acc} = 0$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, (l_e, \mu_e))$, $\gamma(x) = (l, \text{public } bty*)$, $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, $i > 1$, $\text{Label}(e, \gamma) = \text{public}$, and $\text{UpdatePtr}(\sigma_1, (l_1, \mu_1), [1, [(l_e, \mu_e)], [1], i - 1], \text{public } bty*) = (\sigma_2, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$ and ψ such that $(\gamma, \sigma, \text{acc}, *x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x = e \cong_\psi *\widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp1} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin *x = e$. Therefore, by Lemma 3.2.3 we have $*x = e \cong *\widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(*x = e) = *\widehat{x} = \text{Erase}(e)$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, (l_e, \mu_e))$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, (l_e, \mu_e)) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$ and $d_1 \cong \widehat{d}_1$. Given $(l_e, \mu_e) \neq \text{skip}$, by

Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l_e, \mu_e) \cong_\psi (\widehat{l}_e, \widehat{\mu}_e)$.

Given $\gamma(x) = (l, \text{public } bty*)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by public $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable, public } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable, } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, public $bty* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $i > 1$ and $i = \widehat{i}$, we have $\widehat{i} > 1$.

Given $\text{UpdatePtr}(\sigma_1, (l_1, \mu_1), [1, [(l_e, \mu_e)], [1], i-1]) = (\sigma_2, 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, public $bty* \cong \widehat{bty}*$, and $[1, [(l_e,\ \mu_e)], [1], 1] \cong_\psi [1, [(\widehat{l}_e,\ \widehat{\mu}_e)], [1], 1]$, by Lemma 3.2.54 we have $\text{UpdatePtr}(\widehat{\sigma}_1,\ s(\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1],\ i], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable, } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ \widehat{i}]$, $\widehat{i} > 1$, and $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wdp1}} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \text{skip})$ by Vanilla C rule Pointer Dereference Write Higher Level Indirection.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, *x = e) \Downarrow_{wdp1} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wdp1}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wdp1 \cong \widehat{wdp1}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp2} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp2} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Private Pointer Dereference Write Higher Level Indirection, we have $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma,\ \text{acc},\ e) \Downarrow_{d_1} (\gamma,\ \sigma_1,\ \text{acc},\ (l_e, \mu_e))$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable, private } bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$, and $\text{UpdatePrivPtr}(\sigma_1, [\alpha, \bar{l}, \bar{j}, i], [1, [(l_e, \mu_e)], [1], i-1], \text{private } bty*) = (\sigma_2, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \text{acc},\ *x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x = e \cong_\psi *\widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \text{acc},\ *x = e) \Downarrow_{wdp2} (\gamma,\ \sigma_2,\ \text{acc},\ \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin *x = e$. Therefore, by Lemma 3.2.3 we have $*x = e \cong *\widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(*x = e) = *\widehat{x} = \text{Erase}(e)$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, (l_e, \mu_e))$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, (l_e, \mu_e)) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$ and $d_1 \cong \widehat{d}_1$. Given $(l_e, \mu_e) \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l_e, \mu_e) \cong_\psi (\widehat{l}_e, \widehat{\mu}_e)$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]$, $\text{private } bty* \cong \widehat{bty}*$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ i], \text{private } bty*) = (l_1, 0)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], i]$ where $[\alpha,\ \bar{l},\ \bar{j},\ i] \cong_\psi [1, (\widehat{l}_1, 0), [1], \widehat{i}]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$ and $i = \widehat{i}$.

Given $\text{UpdatePrivPtr}(\sigma_1,\ [\alpha,\ \bar{l},\ \bar{j},\ i],\ [1,\ [(l_e,\ \mu_e)],\ [1],\ i-1], \text{private } bty*) = (\sigma_2,\ 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $\text{private } bty* \cong \widehat{bty}*$, $\text{DeclassifyPtr}([\alpha, \bar{l}, \bar{j},\ i],\ \text{private } bty*) = (l_1, \mu_1)$, $(l_1,\ \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, and $[\alpha_e, \bar{l}_e, \bar{j}_e, i - 1] \cong [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1]$, by Lemma 3.2.56 we have $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e,\ \widehat{\mu}_e))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$, $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ \widehat{i}]$, $\widehat{i} > 1$, and $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1],\ \widehat{bty}*) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wdp1}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Pointer Dereference Write Higher Level Indirection.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_2,\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc},\ *x = e) \Downarrow_{wdp2} (\gamma,\ \sigma_2,\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wdp1}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wdp2 \cong \widehat{wdp1}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp3} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp3} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Private Pointer Dereference Write Private Value, we have $\gamma(x) = (l, \text{private } bty*)$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, v)$, $v \neq \text{skip}$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{Label}(e, \gamma) = \text{private}$, $(bty = \text{int}) \vee (bty = \text{float})$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, $\text{acc} = 0$, and $\text{UpdatePriv}(\sigma_1, \alpha, \bar{l}, \bar{j}, \text{private } bty, v) = (\sigma_2, 1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x} = \widehat{e})$ and ψ such that $(\gamma, \sigma, \text{acc}, *x = e) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $*x = e \cong_\psi *\widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp3} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin *x = e$. Therefore, by Lemma 3.2.3 we have $*x = e \cong *\widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(*x = e) = *\widehat{x} = \text{Erase}(e)$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, v) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v})$ and $d_1 \cong \widehat{d}_1$. Given $v \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, $\text{private } bty* \cong \widehat{bty}*$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha, \bar{l}, \bar{j}, 1], \text{private } bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha, \bar{l}, \bar{j}, 1] \cong_\psi [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$.

Given $\text{UpdatePriv}(\sigma_1, \alpha, \bar{l}, \bar{j}, \text{private } bty, v) = (\sigma_2, 1)$ $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $\text{private } bty \cong \widehat{bty}$, $\text{DeclassifyPtr}([\alpha, \bar{l}, \bar{j}, i], \text{private } bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, $\text{private } bty* \cong \widehat{ty}$, and $v \cong_\psi \widehat{v}$, by Lemma 3.2.55 we have $\text{UpdateOffset}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty*})$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty*},\ \text{public},\ 1))$, $\text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l_1},\ \widehat{\mu_1})],\ [1],\ 1]$, and $\text{UpdateOffset}(\widehat{\sigma}_1,\ (\widehat{l_1}, \widehat{\mu_1}),\ \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wdp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Pointer Dereference Write Value.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_2,\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc},\ *x = e) \Downarrow_{wdp3} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wdp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wdp3 \cong \widehat{wdp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp4} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp4} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Private Pointer Dereference Write Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma,\ \sigma_1, \text{acc}, v)$, $v \neq \text{skip}$, $\text{acc} = 0$, $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, and $\text{UpdatePriv}(\sigma_1, \alpha, \bar{l}, \bar{j}, \text{private } bty, \text{encrypt}(v)) = (\sigma_2, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e})$ and ψ such that $(\gamma, \sigma, \text{acc}, *x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x = e \cong_\psi *\widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp4} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin *x = e$. Therefore, by Lemma 3.2.3 we have $*x = e \cong *\widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(*x = e) = *\widehat{x} = \text{Erase}(e)$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$ by Definition 3.2.20. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$ and $d_1 \cong \widehat{d_1}$. Given $v \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(v, \gamma) = \text{public}$ and therefore $v = \widehat{v}$ by Definition 3.2.18 and Definition 3.2.10.

Given $v \neq \text{skip}$ and $v \cong \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ 1]$, $\text{private } bty* \cong \widehat{bty}*$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha,\ \bar{l}, \bar{j}, 1], \text{private } bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha, \bar{l}, \bar{j}, 1] \cong_\psi [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$.

Given $v = \widehat{v}$, by Definition 3.2.18 we have $\text{encrypt}(v) \cong_\psi \widehat{v}$. Given $\text{UpdatePriv}(\sigma_1, \alpha,\ \bar{l},\ \bar{j},\ \text{private } bty, \text{encrypt}(v)) = (\sigma_2, 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ i],\ \text{private } bty*) = (l_1,\ \mu_1)$, $(l_1,\ \mu_1) \cong_\psi (\widehat{l}_1,\ \mu_1)$, $\text{private } bty \cong \widehat{bty}$, and $v \cong_\psi \widehat{v}$, by Lemma 3.2.55 we have $\text{UpdateOffset}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1),\ \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \Box, *\widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1, \text{PermL}(\text{Freeable}, \widehat{bty}*,\ \text{public},\ 1))$, $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$, and $\text{UpdateOffset}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1),\ \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \Box, *\widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wdp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \text{skip})$ by Vanilla C rule Pointer Dereference Write Value.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \text{skip})$. Therefore, we have $(\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp4} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \Box, *\widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wdp}} (\widehat{\gamma},\ \widehat{\sigma}_2, \Box, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wdp4 \cong \widehat{wdp}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp5} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp5} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Private Pointer Dereference Write Higher Level Indirection Multiple Locations, we have $\gamma(x) = (l, \text{private } bty*)$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, [\alpha_e, \bar{l}_e, \bar{j}_e, i-1])$, $\sigma_1(l) = (\omega, \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable},\ \text{private } bty*,\ \text{private},\ \alpha))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{acc} = 0$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]$, $i > 1$, and $\text{UpdatePrivPtr}(\sigma_1,\ [\alpha,\ \bar{l},\ \bar{j},\ i], [\alpha_e, \bar{l}_e, \bar{j}_e, i-1], \text{private } bty*) = (\sigma_2, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \Box, *\widehat{x} = \widehat{e})$ and ψ such that $(\gamma, \sigma, \text{acc}, *x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \Box, *\widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x = e \cong_\psi *\widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp5} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin *x = e$. Therefore, by Lemma 3.2.3 we have $*x = e \cong *\widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(*x = e) = *\widehat{x} = \text{Erase}(e)$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ and private $bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, [\alpha_e, \bar{l}_e, \bar{j}_e, i-1])$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, [\alpha_e, \bar{l}_e, \bar{j}_e, i-1]) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$ and $d_1 \cong \widehat{d_1}$. Given $[\alpha_e, \bar{l}_e, \bar{j}_e, i-1] \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $[\alpha_e, \bar{l}_e, \bar{j}_e, i-1] \cong_\psi [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1]$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]$, private $bty* \cong \widehat{bty*}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ i], \text{private } bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha,\ \bar{l},\ \bar{j},\ i] \cong_\psi [1, (\widehat{l}_1, \widehat{\mu}_1), [1], i]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $i > 1$ and $i = \widehat{i}$, we have $\widehat{i} > 1$.

Given $\text{UpdatePrivPtr}(\sigma_1, [\alpha,\ \bar{l},\ \bar{j},\ i], [\alpha_e, \bar{l}_e, \bar{j}_e, i-1], \text{private } bty*) = (\sigma_2, 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, private $bty* \cong \widehat{bty*}$, $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ i], \text{private } bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, and $[\alpha_e, \bar{l}_e, \bar{j}_e, i-1] \cong_\psi [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1]$, by Lemma 3.2.56 we have $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1], \widehat{bty*}) = (\widehat{\sigma}_2, 1)$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e,\ \widehat{\mu}_e))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*})$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$, $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, $\widehat{i} > 1$, and $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1], \widehat{bty*}) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wdp1}} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Pointer Dereference Write Higher Level Indirection.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, *x = e) \Downarrow_{wdp5} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, *\widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wdp1}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wdp5 \cong \widehat{wdp1}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin} (\gamma, \sigma_1, \text{acc}, v_1)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin} (\gamma, \sigma_1, \text{acc}, v_1)$ by SMC^2 rule Pre-Increment Public Variable, we have $\text{acc} = 0$, $\gamma(x) = (l, \text{public}\ bty)$, $\sigma(l) = (\omega, \text{public}\ bty, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, 1))$, $\text{DecodeVal}(\text{public}\ bty, 1, \omega) = v$, $v_1 =_{\text{public}} v +_{\text{public}} 1$, and $\text{UpdateVal}(\sigma, l, v_1, \text{public}\ bty) = \sigma_1$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x})$ and ψ such that $(\gamma, \sigma, \text{acc}, ++x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $++\ x \cong_\psi ++\ \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(++x) = ++\ \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{public}\ bty)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public}\ bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{public}\ bty, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, 1))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, bty, \text{public}, 1))$ where $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodeVal}(\text{public}\ bty, 1, \omega) = v$, $\text{public}\ bty \cong \widehat{bty}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, 1, \widehat{\omega}) = \widehat{v}$ and $v \cong_\psi \widehat{v}$.

Given $v_1 =_{\text{public}} v +_{\text{public}} 1$ and $v \cong_\psi \widehat{v}$, by Lemma 3.2.22 we have $\widehat{v}_1 = \widehat{v} + 1$ such that $v_1 \cong_\psi \widehat{v}_1$.

Given $\text{UpdateVal}(\sigma, l, v_1, \text{public}\ bty) = \sigma_1$, $\text{public}\ bty \cong \widehat{bty}$ $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $l = \widehat{l}$, and $v_1 \cong_\psi \widehat{v}_1$, by Lemma 3.2.52 we have $\text{UpdateVal}(\widehat{\sigma}, \widehat{l}, \widehat{v}_1, \widehat{bty}) = \widehat{\sigma}_1$ such that $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))$, $\text{DecodeVal}(\widehat{bty}, 1, \widehat{\omega}) = \widehat{v}$, $\widehat{v}_1 = \widehat{v} + 1$, and $\text{UpdateVal}(\widehat{\sigma}, \widehat{l}, \widehat{v}_1, \widehat{bty}) = \widehat{\sigma}_1$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x}) \Downarrow'_{\widehat{pin}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v}_1)$ by Vanilla C rule Pre-Increment Variable.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $v_1 \cong_\psi \widehat{v}_1$, by Definition 3.2.20 we have $(\gamma, \sigma_1, \text{acc}, v_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v}_1)$. Therefore, we have $(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin} (\gamma, \sigma_1, \text{acc}, v_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x}) \Downarrow'_{\widehat{pin}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v}_1)$, $\Pi \cong_\psi \Sigma$, and $pin \cong \widehat{pin}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin1} (\gamma, \sigma_1, \text{acc}, v_1)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin1} (\gamma, \sigma_1, \text{acc}, v_1)$ by SMC^2 rule Pre-Increment Private Variable, we have $\text{acc} = 0$, $\gamma(x) = (l, \text{private } bty)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))$, $\text{DecodeVal}(\text{private } bty, 1, \omega) = v$, $v_1 =_{\text{private}} v +_{\text{private}} \text{encrypt}(1)$, and $\text{UpdateVal}(\sigma, l, v_1, \text{private } bty) = \sigma_1$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x})$ and ψ such that $(\gamma, \sigma, \text{acc}, ++x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $++\ x \cong_\psi ++\ \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(++x) = ++\ \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, bty, \text{public}, 1))$ where $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodeVal}(\text{private } bty, 1, \omega) = v$, private $bty \cong \widehat{bty}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.41 we have $\text{DecodeVal}(\widehat{bty}, 1, \widehat{\omega}) = \widehat{v}$ and $v \cong_\psi \widehat{v}$.

Given $\text{encrypt}(1)$, by Definition 3.2.18 we have $\text{encrypt}(1) \cong_\psi 1$. Given $v_1 =_{\text{private}} v +_{\text{private}} \text{encrypt}(1)$ and $v \cong_\psi \widehat{v}$, by Lemma 3.2.22 we have $\widehat{v}_1 = \widehat{v} + 1$ such that $v_1 \cong_\psi \widehat{v}_1$.

Given $\text{UpdateVal}(\sigma, l, v_1, \text{private } bty) = \sigma_1$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $l = \widehat{l}$, private $bty \cong \widehat{bty}$, and $v_1 \cong_\psi \widehat{v}_1$, by Lemma 3.2.52 we have $\text{UpdateVal}(\widehat{\sigma}, \widehat{l}, \widehat{v}_1, \widehat{bty}) = \widehat{\sigma}_1$ such that $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))$, $\text{DecodeVal}(\widehat{bty}, 1, \widehat{\omega}) = \widehat{v}$, $\widehat{v}_1 = \widehat{v} + 1$, and $\text{UpdateVal}(\widehat{\sigma}, \widehat{l}, \widehat{v}_1, \widehat{bty}) = \widehat{\sigma}_1$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x}) \Downarrow'_{\widehat{pin}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v}_1)$ by Vanilla C rule Pre-Increment Variable.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $v_1 \cong_\psi \widehat{v}_1$, by Definition 3.2.20 we have $(\gamma, \sigma_1, \text{acc}, v_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v}_1)$. Therefore, we have $(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin1} (\gamma, \sigma_1, \text{acc}, v_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x}) \Downarrow'_{\widehat{pin}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v}_1)$, $\Pi \cong_\psi \Sigma$, and $pin1 \cong \widehat{pin}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$ by SMC^2 rule Pre-Increment Public Pointer Single Location, we have $\text{acc} = 0$, $\gamma(x) = (l, \text{public } bty*)$, $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, $((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty), \sigma)$, and $\text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \text{public } bty*) = (\sigma_1, 1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x})$ and ψ such that $(\gamma, \sigma, \text{acc}, ++x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $++\ x \cong_\psi ++\ \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(++x) = ++\ \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{public } bty*)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by $\text{public } bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, $\text{public } bty* \cong \widehat{bty*}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$.

Given $((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty), \sigma)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, $\text{public } bty \cong \widehat{bty}$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, by Lemma 3.2.50 we have $\text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty}), \widehat{\sigma}) = ((\widehat{l}_2, \widehat{\mu}_2), 1)$ such that $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$. By Definition 3.2.14 we have $[\alpha, \bar{l}', \bar{j}, 1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1]$.

Given $\text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \text{public } bty*) = (\sigma_1, 1)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, $\text{public } bty* \cong \widehat{bty*}$, and $[1, [(l_2, \mu_2)], [1], 1] \cong_\psi [1, [(\widehat{l}_2, \mu_2)], [1], 1]$, by Lemma 3.2.54 we have $\text{UpdatePtr}(\widehat{\sigma}, (\widehat{l}, 0), [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], i], \widehat{bty*}) = (\widehat{\sigma}_1, 1)$ such that $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$, $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$, $((\widehat{l}_2, \widehat{\mu}_2), 1) = \text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty}))$, and $\text{UpdatePtr}(\widehat{\sigma}, (\widehat{l}, 0), [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1], \widehat{bty*}) = (\widehat{\sigma}_1, 1)$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x}) \Downarrow'_{\widehat{pin2}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, (\widehat{l}_2, \widehat{\mu}_2))$ by Vanilla C rule Pre-Increment Pointer.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l_2, \mu_2) \cong_\psi (\widehat{l_2}, \widehat{\mu}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma_1, \text{acc}, (l_2, \mu_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l_2}, \widehat{\mu}_2))$. Therefore, we have $(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin2} (\gamma,\ \sigma_1, \text{acc}, (l_2, \mu_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, ++\ \widehat{x}) \Downarrow'_{\widehat{pin2}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l_2}, \widehat{\mu}_2))$, $\Pi \cong_\psi \Sigma$, and $pin2 \cong \widehat{pin2}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin3} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin6} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin6} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$ by SMC^2 rule Pre-Increment Private Pointer Single Location, we have $\text{acc} = 0$, $\gamma(x) = (l, \text{private } bty*)$, $\sigma(l) = (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, $\text{GetLocation}((l_1, \mu_1), \tau(\text{private } bty), \sigma) = ((l_2, \mu_2), 1)$, and $\text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \text{private } bty*) = (\sigma_1, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, ++\ \widehat{x})$ and ψ such that $(\gamma,\ \sigma, \text{acc}, ++x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, ++\ \widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $++\ x \cong_\psi ++\ \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(++x) = ++\ \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ and $\text{private } bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, $\text{private } bty* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$.

Given $((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{private } bty), \sigma)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, $\text{private } bty \cong \widehat{bty}$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, by Lemma 3.2.50 we have $\text{GetLocation}((\widehat{l}_1,\ \widehat{\mu}_1),\ \tau(\widehat{bty}), \widehat{\sigma}) = ((\widehat{l}_2, \widehat{\mu}_2), 1)$ such that $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$. By Definition 3.2.14 we have $[1, [(l_2, \mu_2)], [1], 1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1]$.

Given $\text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \text{private } bty*) = (\sigma_1, 1)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, $\text{private } bty* \cong \widehat{bty}*$, and $[1, [(l_2, \mu_2)], [1], 1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1]$, by Lemma 3.2.54 we have $\text{UpdatePtr}(\widehat{\sigma}, (\widehat{l}, 0), [1,$

$[(\widehat{l}_2, \widehat{\mu}_2)], [1], i], \widehat{bty}*) = (\widehat{\sigma}_1, 1)$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*)$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$, $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$, $((\widehat{l}_2, \widehat{\mu}_2), 1) = \text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty}), \widehat{\sigma})$, and $\text{UpdatePtr}(\widehat{\sigma}, (\widehat{l}, 0), [1,\ [(\widehat{l}_2, \widehat{\mu}_2)],\ [1],\ 1], \widehat{bty}*) = (\widehat{\sigma}_1, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, ++\ \widehat{x}) \Downarrow'_{\widehat{pin2}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square, (\widehat{l}_2, \widehat{\mu}_2))$ by Vanilla C rule Pre-Increment Pointer.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma_1, \text{acc}, (l_2, \mu_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_2, \widehat{\mu}_2))$. Therefore, we have $(\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin6} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, ++\ \widehat{x}) \Downarrow'_{\widehat{pin2}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_2, \widehat{\mu}_2))$, $\Pi \cong_\psi \Sigma$, and $pin6 \cong \widehat{pin2}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin7} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin6} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin4} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1])$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin4} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1])$ by SMC^2 rule Pre-Increment Private Pointer Multiple Locations, we have $\text{acc} = 0$, $\gamma(x) = (l, \text{private } bty*)$, $\sigma(l) = (\omega,\ \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ 1]$, $\text{IncrementList}(\bar{l}, \tau(\text{private } bty), \sigma) = (\bar{l}', 1)$, and $\text{UpdatePtr}(\sigma, (l, 0), [\alpha, \bar{l}', \bar{j}, 1], \text{private } bty*) = (\sigma_1, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, ++\ \widehat{x})$ and ψ such that $(\gamma,\ \sigma,\ \text{acc},\ ++x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, ++\ \widehat{x})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $++\ x \cong_\psi ++\ \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(++x) = ++\ \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty}*$ by Lemma 3.2.14. By Definition 3.2.6, we have private $bty \cong bty$.

Given $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ 1]$, private $bty* \cong \widehat{bty}*$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha,\ \bar{l}, \bar{j}, i], \text{private } bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha, \bar{l}, \bar{j}, 1] \cong_\psi$

$[1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$.

Given $\text{IncrementList}(\bar{l}, \tau(\text{private } bty), \sigma) = (\bar{l}', 1)$, $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ 1],\ \text{private } bty*) = (l_1, \mu_1)$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $ty \cong \widehat{ty}$, and $\text{DeclassifyPtr}([\alpha,\ \bar{l}',\ \bar{j},\ 1],\ \text{private } bty*) = (l_2, \mu_2)$, by Lemma 3.2.51 we have $((\widehat{l}_2, \widehat{\mu}_2), 1) = \text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1),\ \tau(\widehat{bty*}))$ such that $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$. By Definition 3.2.14 we have $[\alpha, \bar{l}', \bar{j}, 1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1]$.

Given $\text{UpdatePtr}(\sigma, (l, 0), [\alpha, \bar{l}', \bar{j}, 1], \text{private } bty*) = (\sigma_1, 1)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, $\text{private } bty* \cong \widehat{bty*}$, and $[\alpha, \bar{l}', \bar{j}, 1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1]$, by Lemma 3.2.54 we have $\text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_2, \widehat{\mu}_2)],\ [1],\ 1],\ \widehat{bty*}) = (\widehat{\sigma}_1, 1)$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty*})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty*},\ \text{public},\ 1))$, $\text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ 1]$, $\widehat{i} > 1$, $((\widehat{l}_2, \widehat{\mu}_2),\ 1) = \text{GetLocation}((\widehat{l}_1,\ \widehat{\mu}_1),\ \tau(\widehat{bty*}))$, and $\text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_2, \widehat{\mu}_2)],\ [1],\ i],\ \widehat{bty*}) = (\widehat{\sigma}_1, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, ++\ \widehat{x}) \Downarrow'_{\widehat{pin3}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_2, \widehat{\mu}_2))$ by Vanilla C rule Pre-Increment Pointer Higher Level Indirection.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $[\alpha, \bar{l}', \bar{j}, 1] \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_2, \widehat{\mu}_2))$. Therefore, we have $(\gamma, \sigma,\ \text{acc},\ ++\ x) \Downarrow_{pin4} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, ++\ \widehat{x}) \Downarrow'_{\widehat{pin3}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, (\widehat{l}_2, \widehat{\mu}_2))$, $\Pi \cong_\psi \Sigma$, and $pin4 \cong \widehat{pin3}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin5} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}', \bar{j}, i])$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin4} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1])$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x[e]) \Downarrow_{da} (\gamma_1, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x[e]) \Downarrow_{da} (\gamma_1, \sigma_3, \text{acc}, \text{skip})$ by SMC^2 rule Public 1 Dimension Array Declaration, we have $((ty = \text{public } bty) \wedge ((bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{int}))) \vee (ty = \text{char})$, $l = \phi()$, $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, $\gamma_1 = \gamma[x \rightarrow (l, \text{public const } bty*)]$, $l_1 = \phi()$, $\omega = \text{EncodePtr}(\text{public const } bty*, [1, [(l_1, 0)], [1], 1])$, $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))]$, $\text{acc} = 0$, $n > 0$, $\omega_1 = \text{EncodeVal}(\text{public } bty, \text{NULL})$, and $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma, \sigma, \text{acc}, ty\ x[e]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}[\widehat{e}])$, by Definition 3.2.20 we have

$(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x[e] \cong_\psi \widehat{ty}\ \widehat{x}[\widehat{e}]$. Given $(\gamma, \sigma, \text{acc}, ty\ x[e]) \Downarrow_{da} (\gamma_1, \sigma_3, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin ty\ x[e]$. Therefore, by Lemma 3.2.3 we have $ty\ x[e] \cong \widehat{ty}\ \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(ty\ x[e]) = \text{Erase}(ty)\ \text{Erase}(x[e])$, $\text{Erase}(ty) = \widehat{ty}$, $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{n})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{n})$ and $d_1 \cong \widehat{d}_1$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $v = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $ty \cong \widehat{ty}$ and $((ty = \text{public}\ bty) \wedge ((bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{int}))) \vee (ty = \text{char})$, by Definition 3.2.6 we have $bty \cong \widehat{bty}$. Therefore, we have $\text{public const}\ bty* \cong \text{const}\ \widehat{bty}*$ and $\text{public}\ bty \cong \widehat{bty}$ by Definition 3.2.6.

Given $\gamma_1 = \gamma[x \to (l, \text{public const}\ bty*)]$, $x = \widehat{x}$, $l = \widehat{l}$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $\text{public const}\ bty* \cong \text{const}\ \widehat{bty}*$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \to (\widehat{l},\ \widehat{bty}*)]$ such that $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $l_1 = \phi()$, by Axiom 3.2.3 we have $\widehat{l}_1 = \phi()$ and $l_1 = \widehat{l}_1$.

Given $[1, [(l_1, 0)], [1], 1]$, by Definition 3.2.14 we have $[1,\ [(l_1, 0)],\ [1],\ 1] \cong_\psi [1,\ [(\widehat{l}_1, 0)],\ [1],\ 1]$. Given $\omega = \text{EncodePtr}(\text{public const}\ bty*, [1, [(l_1, 0)], [1], 1])$ and $\text{public const}\ bty* \cong \text{const}\ \widehat{bty}*$, by Lemma 3.2.42 we have $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_2 = \sigma_1[l \to (\omega, \text{public const}\ bty*, 1,$
$\text{PermL}(\text{Freeable}, \text{public const}\ bty*, \text{public}, 1))]$, $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{public const}\ bty* \cong \text{const}\ \widehat{bty}*$, by Lemma 3.2.35 we have $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \to (\widehat{\omega},\ \text{const}\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable}, \text{const}\ \widehat{bty}*, \text{public}, 1))]$ such that $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$.

Given $n > 0$ and $n = \widehat{n}$, we have $\widehat{n} > 0$.

Given $\omega_1 = \text{EncodeVal}(\text{public } bty, \text{NULL})$, and $\text{public } bty \cong \widehat{bty}$, by Lemma 3.2.40 we have $\text{EncodeVal}(\widehat{bty}, \text{NULL}) = \widehat{\omega}_1$ such that $\omega_1 \cong_\psi \widehat{\omega}_1$.

Given $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))]$, $l_1 = \widehat{l}_1$, $\omega_1 \cong_\psi \widehat{\omega}_1$, $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_2)$, $n = \widehat{n}$, and $\text{public } bty \cong \widehat{bty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l}_1 \rightarrow (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))]$ such that $(\gamma_1, \sigma_3) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ x[\widehat{e}])$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_e (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n})$, $\widehat{l} = \phi()$, $\widehat{l}_1 = \phi()$, $\widehat{\omega} = \text{EncodePtr}(\text{const } \widehat{bty}*, [1, [(\widehat{l}_1, 0)], [1], 1])$, $\widehat{\gamma}_1 = \widehat{\gamma}[x \rightarrow (\widehat{l}, \text{const } \widehat{bty}*)]$, $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))]$, $\text{EncodeVal}(\widehat{bty}, \text{NULL}) = \widehat{\omega}_1$, $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l}_1 \rightarrow (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))]$, and $\widehat{n} > 0$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ x[\widehat{e}]) \Downarrow'_{\widehat{da}} (\widehat{\gamma}_1, \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Declaration.

Given $(\gamma_1, \sigma_3) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma_1, \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \text{acc}, ty\ x[e]) \Downarrow_{da} (\gamma_1, \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ x[\widehat{e}]) \Downarrow'_{\widehat{da}} (\widehat{\gamma}_1, \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $da \cong \widehat{da}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x[e]) \Downarrow_{da1} (\gamma_1, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x[e]) \Downarrow_{da1} (\gamma_1, \sigma_3, \text{acc}, \text{skip})$ by SMC^2 rule Private 1 Dimension Array Declaration, we have $\text{Label}(e, \gamma) = \text{public}$, $((ty = \text{private } bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, $n > 0$, $l = \phi()$, $l_1 = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \text{private const } bty*)]$, $\omega = \text{EncodePtr}(\text{private const } bty*, [1, [(l_1, 0)], [1], 1])$, $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))]$, $\omega_1 = \text{EncodeVal}(\text{private } bty, \text{NULL})$, and $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma, \sigma, \text{acc}, ty\ x[e]) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}[\widehat{e}])$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $ty\ x[e] \cong_\psi \widehat{ty}\ \widehat{x}[\widehat{e}]$. Given $(\gamma, \sigma, \text{acc}, ty\ x[e]) \Downarrow_{da1} (\gamma_1, \sigma_3, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin ty\ x[e]$. Therefore, by Lemma 3.2.3 we have $ty\ x[e] \cong \widehat{ty}\ \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(ty\ x[e]) = \text{Erase}(ty)\ \text{Erase}(x[e])$, $\text{Erase}(ty) = \widehat{ty}$, $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $ty \cong \widehat{ty}$ and $((ty = \text{private } bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, by Definition 3.2.6 we have $bty \cong \widehat{bty}$. Therefore, we have $\text{private const } bty* \cong \text{const } \widehat{bty}*$ and $\text{private } bty \cong \widehat{bty}$ by Definition 3.2.6.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{n})$ and $d_1 \cong \widehat{d_1}$. Given $n \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $n > 0$ and $n = \widehat{n}$, we have $\widehat{n} > 0$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $l_1 = \phi()$, by Axiom 3.2.3 we have $\widehat{l_1} = \phi()$ and $l_1 = \widehat{l_1}$.

Given $\gamma_1 = \gamma[x \to (l, \text{private const } bty*)]$, $x = \widehat{x}$, $l = \widehat{l}$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $\text{private const } bty* \cong \text{const } \widehat{bty}*$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \to (\widehat{l},\ \widehat{bty}*)]$ such that $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $[1,\ [(l_1,\ 0)],\ [1],\ 1]$, by Definition 3.2.14 we have $[1,\ [(l_1,\ 0)],\ [1],\ 1] \cong_\psi [1,\ [(\widehat{l_1},\ 0)],\ [1],\ 1]$. Given $\omega = \text{EncodePtr}(\text{private const } bty*, [1, [(l_1, 0)], [1], 1])$ and $\text{private const } bty* \cong \text{const } \widehat{bty}*$, by Lemma 3.2.42 we have $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_2 = \sigma_1[l \to (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))]$, $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{private const } bty* \cong \text{const } \widehat{bty}*$, by Lemma 3.2.35 we have $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \to (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))]$ such that $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$.

Given $\omega_1 = \text{EncodeVal}(\text{private } bty, \text{NULL})$, and $\text{private } bty \cong \widehat{bty}$, by Lemma 3.2.40 we have $\text{EncodeVal}(\widehat{bty}, \text{NULL}) = \widehat{\omega}_1$ such that $\omega_1 \cong_\psi \widehat{\omega}_1$.

Given $\sigma_3 = \sigma_2[l_1 \to (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))]$, $l_1 = \widehat{l_1}$, $\omega_1 \cong_\psi \widehat{\omega}_1$, $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$, $n = \widehat{n}$, and $\text{private } bty \cong \widehat{bty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l_1} \to (\widehat{\omega}_1,\ \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))]$ such that $(\gamma_1, \sigma_3) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square,\ \widehat{ty}\ x[\widehat{e}])$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_e (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$, $\widehat{l} = \phi()$, $\widehat{l_1} = \phi()$, $\widehat{\omega} = \text{EncodePtr}(\text{const } \widehat{bty}*, [1,\ [(\widehat{l_1}, 0)],\ [1],\ 1])$, $\widehat{\gamma}_1 = \widehat{\gamma}[x \to (\widehat{l},\ \text{const } \widehat{bty}*)]$, $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \to (\widehat{\omega},\ \text{const } \widehat{bty}*,\ 1, \text{PermL}(\text{Freeable}, \text{const}$

$\widehat{bty}*$, public, 1))], EncodeVal($\widehat{bty}$, NULL) $= \widehat{\omega}_1$, $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l}_1 \rightarrow (\widehat{\omega}_1,\ \widehat{bty}, \widehat{n},$ PermL(Freeable, $\widehat{bty}$, public, $\widehat{n}$))], and $\widehat{n} > 0$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ x[\widehat{e}]) \Downarrow'_{\widehat{da}} (\widehat{\gamma}_1,\ \widehat{\sigma}_3, \square,$ skip) by Vanilla C rule 1D Array Declaration.

Given $(\gamma_1, \sigma_3) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma_1, \sigma_3,$ acc, skip$) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_3, \square,$ skip). Therefore, we have $(\gamma,\ \sigma,$ acc, $ty\ x[e]) \Downarrow_{da1} (\gamma_1, \sigma_3,$ acc, skip$) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{ty}\ x[\widehat{e}]) \Downarrow'_{\widehat{da}} (\widehat{\gamma}_1,\ \widehat{\sigma}_3, \square,$ skip), $\Pi \cong_\psi \Sigma$, and $da1 \cong \widehat{da}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma,$ acc, $x[e]) \Downarrow_{ra} (\gamma,\ \sigma_1,$ acc, $v_i)$

Given $\Pi \triangleright (\gamma, \sigma,$ acc, $x[e]) \Downarrow_{ra} (\gamma,\ \sigma_1,$ acc, $v_i)$ by SMC^2 rule Public 1D Array Read Public Index, we have Label(e, γ) $=$ public, $(\gamma,\ \sigma,$ acc, $e) \Downarrow_{d_1} (\gamma,\ \sigma_1,$ acc, $i)$, $\gamma(x) = (l,$ public const $bty*)$, $\sigma_1(l) = (\omega,$ public const $bty*$, 1, PermL(Freeable, public const $bty*$, public, 1)), DecodePtr(public const $bty*$, 1, $\omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1)$ $= (\omega_1,$ public $bty, n,$ PermL(Freeable, public $bty,$ public, n)), DecodeVal(public $bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, and $0 \leq i \leq n-1$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma, \sigma,$ acc, $x[e]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi$ $(\widehat{\gamma},\ \widehat{\sigma})$ and $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given $(\gamma, \sigma,$ acc, $x[e]) \Downarrow_{ra} (\gamma,\ \sigma_1,$ acc, $v_i)$, by Lemma 3.2.2 we have $(l, \mu) \notin x[e]$. Therefore, by Lemma 3.2.3 we have $x[e] \cong \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have Erase$(x[e]) = \widehat{x}[$Erase$(e)]$ where $x = \widehat{x}$ and Erase$(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma,$ acc, $e)$. Given $(\gamma, \sigma,$ acc, $e) \Downarrow_{d_1} (\gamma, \sigma_1,$ acc, $i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,$ $\square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1,$ acc, $i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{i})$ and $d_1 \cong \widehat{d}_1$. Given $i \neq$ skip, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given Label$(e_1, \gamma) =$ public, we have Label$(i, \gamma) =$ public and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $\gamma(x) = (l,$ public const $bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l},$ const $\widehat{bty}*)$ such that $l = \widehat{l}$ by public const $bty* \cong$ const $\widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega,$ public const $bty*$, 1, PermL(Freeable, public const $bty*$, public, 1)), $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},$ const $\widehat{bty}*, 1,$ PermL(Freeable, const $\widehat{bty}*$, public, 1)) such that $\omega \cong_\psi \widehat{\omega}$.

Given DecodePtr(public const $bty*$, 1, ω) = [1, $(l_1, 0)$, [1], 1], public const $bty* \cong$ const $\widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have DecodePtr(const $\widehat{bty}*$, 1, $\widehat{\omega}$) = [1, $(\widehat{l}_1, 0)$, [1], 1] where [1, $(l_1, 0)$, [1], 1] $\cong_\psi$ [1, $[(\widehat{l}_1, 0)]$, [1], 1] such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1,$ public $bty, n,$ PermL(Freeable, public bty, public, n)), $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n},$ PermL(Freeable, bty, public, $\widehat{n}$)) where $\omega_1 \cong_\psi \widehat{\omega}_1$, public $bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given DecodeVal(public bty, n, ω_1) = $[v_0, ..., v_{n-1}]$, public $bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have DecodeVal(bty, $\widehat{n}$, $\widehat{\omega}_1$) = $[\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$. By Definition 3.2.10 we have $\forall m \in \{0, ..., n-1\}$, $v_m \cong_\psi \widehat{v}_m$. Therefore, given $i = \widehat{i}$, we have $v_i \cong_\psi \widehat{v}_{\widehat{i}}$.

Given $0 \leq i \leq n-1$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $0 \leq \hat{i} \leq \widehat{n} - 1$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$, $\widehat{\gamma}(x) = (\widehat{l},$ const $\widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},$ const $\widehat{bty}*$, 1, PermL(Freeable, const $\widehat{bty}*$, public, 1)), DecodePtr(const $\widehat{bty}*$, 1, $\widehat{\omega}$) = [1, $[(\widehat{l}_1, 0)]$, [1], 1], $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n},$ PermL(Freeable, $\widehat{bty}$, public, $\widehat{n}$)), $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$, $0 \leq \widehat{i} \leq \widehat{n} - 1$, and DecodeVal($\widehat{bty}$, $\widehat{n}$, $\widehat{\omega}_1$) = $[\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{ra}} (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{v}_{\widehat{i}})$ by Vanilla C rule Array Read.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $v_i \cong_\psi \widehat{v}_{\widehat{i}}$, by Definition 3.2.20 we have $(\gamma, \sigma_1, \text{acc}, v_i) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v}_{\widehat{i}})$. Therefore, we have $(\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra} (\gamma, \sigma_1, \text{acc}, v_i) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{ra}} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v}_{\widehat{i}})$, $\Pi \cong_\psi \Sigma$, and $ra \cong \widehat{ra}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra3} (\gamma, \sigma_1, \text{acc}, v_i)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra3} (\gamma, \sigma_1, \text{acc}, v_i)$ by SMC^2 rule Private 1D Array Read Public Index, we have Label(e, γ) = public, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, $\gamma(x) = (l,$ private const $bty*)$, $\sigma_1(l) = (\omega,$ private const $bty*$, 1, PermL(Freeable, private const $bty*$, private, 1)), DecodePtr(private const $bty*$, 1, ω) = [1, $[(l_1, 0)]$, [1], 1], $\sigma_1(l_1) = (\omega_1,$ private bty, n, PermL(Freeable, private bty, private, n)), DecodeVal(private bty, n, ω_1) = $[v_0, ..., v_{n-1}]$, and $0 \leq i \leq n-1$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma, \sigma, \text{acc}, x[e]) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given $(\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra3} (\gamma, \sigma_1, \text{acc}, v_i)$, by Lemma 3.2.2 we have $(l, \mu) \notin x[e]$.

Therefore, by Lemma 3.2.3 we have $x[e] \cong \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{i})$ and $d_1 \cong \widehat{d}_1$. Given $i \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1, (l_1, 0), [1], 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$. By Definition 3.2.10 we have $\forall m \in \{0, ..., n-1\}$, $v_m \cong_\psi \widehat{v}_m$. Therefore, given $i = \widehat{i}$, we have $v_i \cong_\psi \widehat{v}_{\widehat{i}}$.

Given $0 \leq i \leq n-1$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $0 \leq \widehat{i} \leq \widehat{n} - 1$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x})$, $\widehat{\gamma}(x) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public},\ 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $0 \leq \widehat{i} \leq \widehat{n} - 1$, and $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square,$

$\widehat{x}) \Downarrow'_{\widehat{ra}} (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{v_{\widehat{i}}})$ by Vanilla C rule Array Read.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v_i \cong_\psi \widehat{v_{\widehat{i}}}$, by Definition 3.2.20 we have $(\gamma,\ \sigma_1, \text{acc}, v_i) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v_{\widehat{i}}})$. Therefore, we have $(\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra3} (\gamma,\ \sigma_1, \text{acc}, v_i) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{ra}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v_{\widehat{i}}})$, $\Pi \cong_\psi \Sigma$, and $ra3 \cong \widehat{ra}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra1} (\gamma, \sigma_1, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra1} (\gamma, \sigma_1, \text{acc}, v)$ by SMC^2 rule Private 1D Array Read Private Index we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, $\gamma(x) = (l, \text{private const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, \ldots, v_{n-1}]$, and $v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge v_m$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma, \sigma, \text{acc}, x[e]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given $(\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra1} (\gamma, \sigma_1, \text{acc}, v)$, by Lemma 3.2.2 we have $(l, \mu) \notin x[e]$. Therefore, by Lemma 3.2.3 we have $x[e] \cong \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$ by Definition 3.2.20. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{i})$ and $d_1 \cong \widehat{d_1}$. Given $i \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1, (l_1, 0), [1], 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$

such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$. By Axiom 3.2.1, we have $0 \leq \widehat{i} \leq \widehat{n} - 1$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge v_m$, by Axiom 3.2.1 and Lemma 3.2.9, we have $v \cong_\psi \widehat{v}_{\widehat{i}}$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x})$, $\widehat{\gamma}(x) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public},\ 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ 0)],\ [1],\ 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $0 \leq \widehat{i} \leq \widehat{n} - 1$, and $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{ra}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v}_{\widehat{i}})$ by Vanilla C rule Array Read.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}_{\widehat{i}}$, by Definition 3.2.20 we have $(\gamma, \sigma_1, \text{acc}, v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v}_{\widehat{i}})$. Therefore, we have $(\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra1} (\gamma, \sigma_1, \text{acc}, v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{ra}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v}_{\widehat{i}})$, $\Pi \cong_\psi \Sigma$, and $ra1 \cong \widehat{ra}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra2} (\gamma, \sigma_1, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra2} (\gamma, \sigma_1, \text{acc}, v)$ by SMC^2 rule Public 1D Array Read Private Index, we have $\gamma(x) = (l,\ \text{public const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $(\gamma,\ \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma,\ \sigma_1,\ \text{acc},\ i)$, $\sigma_1(l) = (\omega, \text{public const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$, $\sigma_1(l_1) = (\omega_1,\ \text{public } bty,\ n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $\text{DecodeVal}(\text{public } bty,\ n,\ \omega_1) = [v_0, ..., v_{n-1}]$, and $v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge \text{encrypt}(v_m)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma, \sigma, \text{acc}, x[e]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given $(\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra2} (\gamma, \sigma_1, \text{acc}, v)$, by Lemma 3.2.2 we have $(l, \mu) \notin x[e]$. Therefore, by Lemma 3.2.3 we have $x[e] \cong \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l,\ \text{public const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{i})$ and $d_1 \cong \widehat{d_1}$. Given $i \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$.

Given $\sigma_1(l) = (\omega,\ \text{public const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{public const } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0), [1], 1]$, $\text{public const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{public } bty \cong \widehat{bty}$, and $n = \widehat{n}$. By Axiom 3.2.1, we have $0 \leq \widehat{i} \leq \widehat{n} - 1$.

Given $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{public } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $\gamma(x) = (l,\ \text{public const } bty*)$, by Definition 3.2.17 we have $[v_0, ..., v_{n-1}] = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$. Given $v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge \text{encrypt}(v_m)$, by Axiom 3.2.1 and Lemma 3.2.10 we have $v \cong_\psi \widehat{v}_{\widehat{i}}$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{x})$, $\widehat{\gamma}(x) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ 0)],\ [1],\ 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n}, \text{PermL}(\text{Freeable},\ \widehat{bty}, \text{public}, \widehat{n}))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}) \Downarrow'_{\widehat{d_1}} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{i})$, $0 \leq \widehat{i} \leq \widehat{n} - 1$, and $\text{DecodeVal}(\widehat{bty},\ \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{x}) \Downarrow'_{\widehat{ra}} (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{v}_{\widehat{i}})$ by Vanilla C rule Array Read.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}_{\widehat{i}}$, by Definition 3.2.20 we have $(\gamma, \sigma_1, \text{acc}, v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{v}_{\widehat{i}})$. Therefore, we have $(\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra2} (\gamma, \sigma_1, \text{acc}, v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \Box, \widehat{x}) \Downarrow'_{\widehat{ra}} (\widehat{\gamma},\ \widehat{\sigma}_1, \Box, \widehat{v}_{\widehat{i}})$, $\Pi \cong_\psi \Sigma$, and $ra2 \cong \widehat{ra}$ by

Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa} (\gamma, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by SMC^2 rule Public 1D Array Write Public Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{v}{v_i}\right)$, $0 \leq i \leq n-1$, and $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \text{public } bty) = \sigma_3$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$ and ψ such that $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x[e_1] = e_2 \cong_\psi \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. Given $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa} (\gamma, \sigma_3, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x[e_1] = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1] = e_2 \cong \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. By Definition 3.2.10 we have $\text{Erase}(x[e_1] = e_2 = \text{Erase}(x[e_1]) = \text{Erase}(e_2)$, $\text{Erase}(x[e_1]) = \widehat{x}[\text{Erase}(e_1)]$ where $x = \widehat{x}$, $\text{Erase}(e_1) = \widehat{e}_1$, and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, i) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and $d_1 \cong \widehat{d}_1$. Given $i \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, v) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$ and $d_2 \cong \widehat{d}_2$. Given $v \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \text{public const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{public const } bty*,\ \text{public},\ 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{public const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)],\ [1],\ 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1,\ [(\widehat{l}_1, 0)],\ [1],\ 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{public } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{public } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{v}{v_i}\right)$, $v \cong_\psi \widehat{v}$, $i = \widehat{i}$, and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, by Lemma 3.2.63 we have $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$ such that $[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}]$.

Given $0 \leq i \leq n-1$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $0 \leq \hat{i} \leq \widehat{n} - 1$.

Given $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n_e-1}], \text{public } bty) = \sigma_3$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $l_1 = \widehat{l}_1$, $\text{public } bty \cong \widehat{bty}$, and $[v'_0, ..., v'_{n_e-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}_e-1}]$, by Lemma 3.2.52 we have $\text{UpdateVal}(\widehat{\sigma}_2,\ \widehat{l}_1,\ [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \hat{i})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{const } \widehat{bty}*,\ \text{public},\ 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ 0)],\ [1],\ 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty}, \widehat{n},\ \text{PermL}(\text{Freeable},\ \widehat{bty},\ \text{public}, \widehat{n}))$, $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0,\ ...,\ \widehat{v}_{\widehat{n}-1}]$, $0 \leq \hat{i} \leq \widehat{n} - 1$, $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$, and $\text{UpdateVal}(\widehat{\sigma}_2, \widehat{l}_1, [\widehat{v}'_0,\ ...,\ \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{\widehat{wa}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Write.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_3,\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc},\ x[e_1] = e_2) \Downarrow_{wa} (\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{\widehat{wa}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wa \cong \widehat{wa}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa4} (\gamma, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa4} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by SMC^2 rule Private 1D Array Write Private Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{public}$, $\text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_1} (\gamma, \sigma_2, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{v}{v_i}\right)$ $0 \leq i \leq n-1$, and $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \text{private } bty) = \sigma_3$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$ and ψ such that $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x[e_1] = e_2 \cong_\psi \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. Given $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa4} (\gamma, \sigma_3, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x[e_1] = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1] = e_2 \cong \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. By Definition 3.2.10 we have $\text{Erase}(x[e_1] = e_2 = \text{Erase}(x[e_1]) = \text{Erase}(e_2)$, $\text{Erase}(x[e_1]) = \widehat{x}[\text{Erase}(e_1)]$ where $x = \widehat{x}$, $\text{Erase}(e_1) = \widehat{e}_1$, and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, i) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and $d_1 \cong \widehat{d}_1$. Given $i \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, v) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$ and $d_2 \cong \widehat{d}_2$. Given $v \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty*})$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)],\ [1],\ 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1,\ [(\widehat{l}_1, 0)],\ [1],\ 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable},\ bty,\ \text{public},\ \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{v}{v_i}\right)$, $v \cong_\psi \widehat{v}$, $i = \widehat{i}$, and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, by Lemma 3.2.63 we have $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$ such that $[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}]$.

Given $0 \leq i \leq n-1$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $0 \leq \hat{i} \leq \widehat{n} - 1$.

Given $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n_e-1}], \text{private } bty) = \sigma_3$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $l_1 = \widehat{l}_1$, $\text{private } bty \cong \widehat{bty}$, and $[v'_0, ..., v'_{n_e-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}_e-1}]$, by Lemma 3.2.52 we have $\text{UpdateVal}(\widehat{\sigma}_2, \widehat{l}_1, [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \hat{i})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)],\ [1],\ 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable},\ \widehat{bty},\ \text{public},\ \widehat{n}))$, $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, $0 \leq \hat{i} \leq \widehat{n} - 1$, $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$, and $\text{UpdateVal}(\widehat{\sigma}_2, \widehat{l}_1, [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{\widehat{wa}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Write.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, x[e_1] = e_2) \Downarrow_{wa4} (\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{\widehat{wa}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wa4 \cong \widehat{wa}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa1} (\gamma, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa1} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by SMC^2 rule Private 1D Array Write Public Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{\text{encrypt}(v)}{v_i}\right)$, $0 \leq i \leq n-1$, and $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \text{private } bty) = \sigma_3$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$ and ψ such that $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x[e_1] = e_2 \cong_\psi \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. Given $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa1} (\gamma, \sigma_3, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x[e_1] = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1] = e_2 \cong \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. By Definition 3.2.10 we have $\text{Erase}(x[e_1] = e_2 = \text{Erase}(x[e_1]) = \text{Erase}(e_2)$, $\text{Erase}(x[e_1]) = \widehat{x}[\text{Erase}(e_1)]$ where $x = \widehat{x}$, $\text{Erase}(e_1) = \widehat{e}_1$, and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, i) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and $d_1 \cong \widehat{d}_1$. Given $i \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, v) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$ and $d_2 \cong \widehat{d}_2$. Given $v \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(v, \gamma) = \text{public}$ and therefore $v = \widehat{v}$ by Definition 3.2.17.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{\text{encrypt}(v)}{v_i}\right)$ and $v = \widehat{v}$, by Definition 3.2.18 and Definition 3.2.10 we have $\text{encrypt}(v) \cong_\psi \widehat{v}$. Given $i = \widehat{i}$ and $[v_0,\ ...,\ v_{n-1}] \cong_\psi [\widehat{v}_0,\ ...,\ \widehat{v}_{n-1}]$, by Lemma 3.2.63 we have $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$ such that $[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}]$.

Given $0 \leq i \leq n-1$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $0 \leq \hat{i} \leq \widehat{n} - 1$.

Given $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n_e-1}]) = \sigma_3$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $l_1 = \widehat{l}_1$, $\text{private } bty \cong \widehat{bty}$, and $[v'_0, ..., v'_{n_e-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}_e-1}]$, by Lemma 3.2.52 we have $\text{UpdateVal}(\widehat{\sigma}_2, \widehat{l}_1, [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \hat{i})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{const } \widehat{bty}*,\ \text{public},\ 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, $0 \leq \hat{i} \leq \widehat{n} - 1$, $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$, and $\text{UpdateVal}(\widehat{\sigma}_2, \widehat{l}_1, [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{\widehat{wa}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Write.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, x[e_1] = e_2) \Downarrow_{wa1} (\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{\widehat{wa}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and

$wa1 \cong \widehat{wa}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa2} (\gamma, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa2} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by SMC^2 rule Private 1D Array Write Public Value Private Index, we have $\text{Label}(e_1, \gamma) = \text{private}$, $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $(bty = \text{int}) \vee (bty = \text{float})$, $v' = \text{encrypt}(v)$, $\forall v_m \in [v_0, ..., v_{n-1}]. v'_m = ((i = \text{encrypt}(m)) \wedge v') \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, and $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \text{private } bty) = \sigma_3$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$ and ψ such that $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x[e_1] = e_2 \cong_\psi \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. Given $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa2} (\gamma, \sigma_3, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x[e_1] = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1] = e_2 \cong \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. By Definition 3.2.10 we have $\text{Erase}(x[e_1] = e_2 = \text{Erase}(x[e_1]) = \text{Erase}(e_2)$, $\text{Erase}(x[e_1]) = \widehat{x}[\text{Erase}(e_1)]$ where $x = \widehat{x}$, $\text{Erase}(e_1) = \widehat{e}_1$, and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, i) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and $d_1 \cong \widehat{d}_1$. Given $i \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, v) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$ and $d_2 \cong \widehat{d}_2$. Given $v \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(v, \gamma) = \text{public}$ and therefore $v = \widehat{v}$ by Definition 3.2.17.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$. By Axiom 3.2.1, we have $0 \leq \widehat{i} \leq \widehat{n} - 1$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0), [1], 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$. By Axiom 3.2.1, we have $0 \leq \widehat{i} \leq \widehat{n} - 1$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $v' = \text{encrypt}(v)$ and $v = \widehat{v}$, by Definition 3.2.17 we have $v' \cong_\psi \widehat{v}$.

Given $\forall v_m \in [v_0,\ ...,\ v_{n-1}].\ v'_m = ((i = \text{encrypt}(m)) \wedge v') \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, by Axiom 3.2.1 and Lemma 3.2.12, we have $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$ such that $[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}]$.

Given $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n_e-1}], \text{private } bty) = \sigma_3$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $l_1 = \widehat{l}_1$, $\text{private } bty \cong \widehat{bty}$ and $[v'_0, ..., v'_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Lemma 3.2.52 we have $\text{UpdateVal}(\widehat{\sigma}_2,\ \widehat{l}_1,\ [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}],\ \widehat{bty}) = \widehat{\sigma}_3$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)\ \Downarrow'_{\widehat{d}_1}\ (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, $0 \leq \widehat{i} \leq \widehat{n} - 1$, $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$, and $\text{UpdateVal}(\widehat{\sigma}_2,\ \widehat{l}_1,\ [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}],\ \widehat{bty}) = \widehat{\sigma}_3$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{\widehat{wa}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Write.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, x[e_1] = e_2) \Downarrow_{wa2} (\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{\widehat{wa}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and

$wa2 \cong \widehat{wa}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa3} (\gamma, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa3} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by SMC^2 rule Private 1D Array Write Private Value Private Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $(bty = \text{int}) \vee (bty = \text{float})$, $\forall v_m \in [v_0, ..., v_{n-1}]. v'_m = ((i = \text{encrypt}(m)) \wedge v) \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, and $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \text{private } bty) = \sigma_3$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$ and ψ such that $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x[e_1] = e_2 \cong_\psi \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. Given $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa3} (\gamma, \sigma_3, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x[e_1] = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1] = e_2 \cong \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. By Definition 3.2.10 we have $\text{Erase}(x[e_1] = e_2 = \text{Erase}(x[e_1]) = \text{Erase}(e_2)$, $\text{Erase}(x[e_1]) = \widehat{x}[\text{Erase}(e_1)]$ where $x = \widehat{x}$, $\text{Erase}(e_1) = \widehat{e}_1$, and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, i) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and $d_1 \cong \widehat{d}_1$. Given $i \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, v) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$ and $d_2 \cong \widehat{d}_2$. Given $v \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty*})$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$. By Axiom 3.2.1, we have $0 \leq \widehat{i} \leq \widehat{n} - 1$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1, (l_1, 0), [1], 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $\forall v_m \in [v_0, ..., v_{n-1}].\ v'_m = ((i = \text{encrypt}(m)) \wedge v) \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, and $i \cong_\psi \widehat{i}$, by Axiom 3.2.1 and Lemma 3.2.11 we have $[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$.

Given $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n_e-1}], \text{private } bty) = \sigma_3$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $l_1 = \widehat{l}_1$, $\text{private } bty \cong \widehat{bty}$, and $[v'_0, ..., v'_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Lemma 3.2.52 we have $\text{UpdateVal}(\widehat{\sigma}_2, \widehat{l}_1, [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)\ \Downarrow'_{\widehat{d}_1}\ (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \hat{i})$, $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, $0 \leq \hat{i} \leq \widehat{n} - 1$, $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\hat{i}}}\right)$, and $\text{UpdateVal}(\widehat{\sigma}_2,\ \widehat{l}_1,\ [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{\widehat{wa}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Write.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa3} (\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{\widehat{wa}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wa3 \cong \widehat{wa}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{rao} (\gamma, \sigma_1, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{rao} (\gamma, \sigma_1, \text{acc}, v)$ by SMC^2 rule Public 1D Array Read Out of Bounds Public Index, we have $\text{Label}(e, \gamma) = \text{public}$, $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(i < 0) \vee (i \geq n)$, and $\text{ReadOOB}(i, n, l_1, \text{public } bty, \sigma_1) = (v, 1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma, \sigma, \text{acc}, x[e]) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given $(\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{rao} (\gamma, \sigma_1, \text{acc}, v)$, by Lemma 3.2.2 we have $(l, \mu) \notin x[e]$. Therefore, by Lemma 3.2.3 we have $x[e] \cong \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, i) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and $d_1 \cong \widehat{d}_1$. Given $i \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, (l_1, 0), [1], 1]$, $\text{public const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{public } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $(i < 0) \vee (i \geq n)$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$.

Given $\text{ReadOOB}(i,\ n,\ l_1,\ \text{public } bty,\ \sigma_1) = (v,\ 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $i = \widehat{i}$, $n = \widehat{n}$, $l = \widehat{l}$, and $\text{public } bty \cong \widehat{bty}$, by Lemma 3.2.62 we have $\text{ReadOOB}(\widehat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty},\ \widehat{\sigma}_1) = (\widehat{v},\ 1)$ such that $v \cong_\psi \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$, and $\text{ReadOOB}(\widehat{i},\ \widehat{n},\ \widehat{l}_1, \widehat{bty}, \widehat{\sigma}_1) = (\widehat{v}, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{rao}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$ by Vanilla C rule 1D Array Read Out of Bounds.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.20 we have $(\gamma,\ \sigma_1,\ \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$. Therefore, we have $(\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{rao} (\gamma,\ \sigma_1,\ \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{rao}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$, $\Pi \cong_\psi \Sigma$, and $rao \cong \widehat{rao}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{rao1} (\gamma,\ \sigma_1,\ \text{acc},\ v)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{rao1} (\gamma,\ \sigma_1,\ \text{acc},\ v)$ by SMC^2 rule Private 1D Array Read Out of Bounds Public Index, we have $\text{Label}(e, \gamma) = \text{public}$, $\gamma(x) = (l,\ \text{private const } bty*)$, $(\gamma,\ \sigma,\ \text{acc},\ e) \Downarrow_{d_1} (\gamma,\ \sigma_1,\ \text{acc},\ i)$, $\sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ [(l_1,\ 0)],\ [1],\ 1]$, $\sigma_1(l_1) = (\omega_1,\ \text{private } bty,\ n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(i < 0) \vee (i \geq n)$, and $\text{ReadOOB}(i,\ n,\ l_1, \text{private } bty, \sigma_1) = (v,\ 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma, \sigma, \text{acc}, x[e]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given $(\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{rao1} (\gamma,\ \sigma_1,\ \text{acc},\ v)$, by Lemma 3.2.2 we have $(l, \mu) \notin x[e]$. Therefore, by Lemma 3.2.3 we have $x[e] \cong \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l,\ \text{private const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{i})$ and $d_1 \cong \widehat{d}_1$. Given $i \neq \text{skip}$, by Lemma 3.2.1 we have

$\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $\sigma_1(l) = (\omega,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{private const } bty*,\ \text{private},\ 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable, const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1,\ 0),\ [1],\ 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ 0)],\ [1],\ 1]$ where $[1,\ (l_1,\ 0),\ [1],\ 1] \cong_\psi [1,\ [(\widehat{l}_1,\ 0)],\ [1],\ 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable},\ bty,\ \text{public},\ \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $(i < 0) \vee (i \geq n)$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$.

Given $\text{ReadOOB}(i,\ n,\ l_1,\ \text{private } bty,\ \sigma_1) = (v,\ 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $i = \widehat{i}$, $n = \widehat{n}$, $l = \widehat{l}$, and $\text{private } bty \cong \widehat{bty}$, by Lemma 3.2.62 we have $\text{ReadOOB}(\widehat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty},\ \widehat{\sigma}_1) = (\widehat{v},\ 1)$ such that $v \cong_\psi \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable, const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)],\ [1],\ 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable},\ \widehat{bty},\ \text{public},\ \widehat{n}))$, $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$, and $\text{ReadOOB}(\widehat{i}, \widehat{n}, \widehat{l}_1, \widehat{bty}, \widehat{\sigma}_1) = (\widehat{v}, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \widehat{x}) \Downarrow'_{\widehat{rao}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$ by Vanilla C rule 1D Array Read Out of Bounds.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.20 we have $(\gamma,\ \sigma_1,\ \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{v})$. Therefore, we have $(\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{rao1} (\gamma,\ \sigma_1,\ \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x}) \Downarrow'_{\widehat{rao}} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v})$, $\Pi \cong_\psi \Sigma$, and $rao1 \cong \widehat{rao}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$ by SMC^2 rule Public 1D Array Write Out of Bounds Public Index Public Value, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma,\ \sigma,\ \text{acc},\ e_1) \Downarrow_{d_1} (\gamma,\ \sigma_1,$

acc, i), (γ, σ_1, acc, e_2) $\Downarrow_{d_2}$ (γ, σ_2, acc, v), $v \neq$ skip, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_2(l) = (\omega$, public const $bty*$, 1, PermL(Freeable, public $s$const $bty*$, public, 1)), DecodePtr(public const $bty*$, 1, ω) = [1, [(l_1, 0)], [1], 1], $\sigma_2(l_1) = (\omega_1$, public bty, n, PermL(Freeable, public bty, public, n)), $(i < 0) \vee (i \geq n)$, and WriteOOB(v, i, n, l_1, public bty, σ_2) = (σ_3, 1).

Given ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{x}[\widehat{e}_1] = \widehat{e}_2$) and ψ such that (γ, σ, acc, $x[e_1] = e_2$) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{x}[\widehat{e}_1] = \widehat{e}_2$), by Definition 3.2.20 we have (γ, σ) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$) and $x[e_1] = e_2 \cong_\psi \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. Given ($\gamma$, σ, acc, $x[e_1] = e_2$) $\Downarrow_{wao}$ (γ, σ_3, acc, skip), by Lemma 3.2.2 we have $(l, \mu) \notin x[e_1] = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1] = e_2 \cong \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. By Definition 3.2.10 we have Erase($x[e_1] = e_2$ = Erase($x[e_1]$) = Erase(e_2), Erase($x[e_1]$) = $\widehat{x}$[Erase(e_1)] where $x = \widehat{x}$, Erase(e_1) = $\widehat{e}_1$, and Erase(e_2) = $\widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given (γ, σ) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$) and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}_1$) such that ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}_1$) $\cong$ (γ, σ, acc, e_1). Given (γ, σ, acc, e_1) $\Downarrow_{d_1}$ (γ, σ_1, acc, i), by the inductive hypothesis we have ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}_1$) $\Downarrow'_{\widehat{d}_1}$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{i}$) and ψ_1 such that (γ, σ_1, acc, i) $\cong_{\psi_1}$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{i}$) and $d_1 \cong \widehat{d}_1$. Given $i \neq$ skip, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have (γ, σ_1) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$) and $i \cong_\psi \widehat{i}$. Given Label(e_1, γ) = public, we have Label(i, γ) = public and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given (γ, σ_1) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$) and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{e}_2$) such that ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{e}_2$) $\cong_\psi$ (γ, σ_1, acc, e_2). Given (γ, σ_1, acc, e_2) $\Downarrow_{d_2}$ (γ, σ_2, acc, v), by the inductive hypothesis we have ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{e}_2$) $\Downarrow'_{\widehat{d}_2}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, $\widehat{v}$) and ψ_2 such that (γ, σ_2, acc, v) $\cong_{\psi_2}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, $\widehat{v}$) and $d_2 \cong \widehat{d}_2$. Given $v \neq$ skip, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have (γ, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$) and $v \cong_\psi \widehat{v}$.

Given $v \neq$ skip and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq$ skip.

Given $\gamma(x) = (l, \text{public const } bty*)$, ($\gamma$, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$), and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by public const $bty* \cong$ const $\widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega$, public const $bty*$, 1, PermL(Freeable, public const $bty*$, public, 1)), (γ, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$), and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}$, const $\widehat{bty}*$, 1, PermL(Freeable, const $\widehat{bty}*$, public, 1)) such that $\omega \cong_\psi \widehat{\omega}$.

Given DecodePtr(public const $bty*$, 1, ω) = [1, (l_1, 0), [1], 1], public const $bty* \cong$ const $\widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have DecodePtr(const $\widehat{bty}*$, 1, $\widehat{\omega}$) = [1, [($\widehat{l}_1$, 0)], [1], 1] where [1, (l_1, 0), [1], 1] $\cong_\psi$ [1, [($\widehat{l}_1$, 0)],

[1], 1] such that $(l_1, 0) \cong_\psi (\widehat{l_1}, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable, public } bty, \text{public}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l_1}$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l_1}) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable, } bty, \text{ public, } \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{public } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $(i < 0) \vee (i \geq n)$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $(\hat{i} < 0) \vee (\hat{i} \geq \widehat{n})$.

Given $\text{WriteOOB}(v, i, n, l_1, \text{public } bty, \sigma_2) = (\sigma_3,\ 1)$, $v \cong_\psi \widehat{v}$, $i = \widehat{i}$, $n = \widehat{n}$, $l_1 = \widehat{l_1}$, $\text{public } bty \cong \widehat{bty}$, and $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, by Lemma 3.2.61 we have $\text{WriteOOB}(\widehat{v},\ \hat{i},\ \widehat{n},\ \widehat{l_1},\ \widehat{bty}, \widehat{\sigma}_2) = (\widehat{\sigma}_3,\ 1)$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \hat{i})$, $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(x) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable, const } \widehat{bty}*, \text{ public, } 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l_1},\ 0)],\ [1],\ 1]$, $\widehat{\sigma}_2(\widehat{l_1}) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable, } \widehat{bty}, \text{ public, } \widehat{n}))$, $(\hat{i} < 0) \vee (\hat{i} \geq \widehat{n})$, and $\text{WriteOOB}(\widehat{v},\ \hat{i}, \widehat{n}, \widehat{l_1}, \widehat{bty}, \widehat{\sigma}_2) = (\widehat{\sigma}_3,\ 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \Downarrow'_{\widehat{wao}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Write Out of Bounds.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, x[e_1] = e_2) \Downarrow_{wao} (\gamma,\ \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \Downarrow'_{\widehat{wao}} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wao \cong \widehat{wao}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao2} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao2} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$ by SMC^2 rule Private 1D Array Write Out of Bounds Public Index Public Value, we have $\text{Label}(e_1, \gamma) = \text{public}$, $\text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, $(\gamma,\ \sigma_1,\ \text{acc},\ e_2) \Downarrow_{d_2} (\gamma,\ \sigma_2,\ \text{acc},\ v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_2(l) = (\omega, \text{ private const } bty*, 1, \text{PermL}(\text{Freeable, private const } bty*, \text{ private, } 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{ private } bty, n, \text{PermL}(\text{Freeable, private } bty, \text{ private, } n))$, $(i < 0) \vee (i \geq n)$, and $\text{WriteOOB}(v, i, n, l_1, \text{private } bty, \sigma_2) = (\sigma_3, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$ and ψ such that $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e_1] = e_2 \cong_\psi \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. Given $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao2} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$,

by Lemma 3.2.2 we have $(l, \mu) \notin x[e_1] = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1] = e_2 \cong \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. By Definition 3.2.10 we have $\text{Erase}(x[e_1] = e_2 = \text{Erase}(x[e_1]) = \text{Erase}(e_2)$, $\text{Erase}(x[e_1]) = \widehat{x}[\text{Erase}(e_1)]$ where $x = \widehat{x}$, $\text{Erase}(e_1) = \widehat{e}_1$, and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \text{acc}, e_1)$. Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{i})$ and $d_1 \cong \widehat{d}_1$. Given $i \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \text{acc}, e_2)$. Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{d_2} (\gamma, \sigma_2, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{v})$ and ψ_2 such that $(\gamma, \sigma_2, \text{acc}, v) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{v})$ and $d_2 \cong \widehat{d}_2$. Given $v \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1, (l_1, 0), [1], 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $(i < 0) \vee (i \geq n)$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$.

Given WriteOOB(v, i, n, l_1, private bty, σ_2) $= (\sigma_3, 1)$ and $v \cong_\psi \widehat{v}$, $i = \widehat{i}$, $n = \widehat{n}$, $l_1 = \widehat{l_1}$, private $bty \cong \widehat{bty}$, and $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, by Lemma 3.2.61 we have WriteOOB($\widehat{v}$, $\hat{i}$, $\widehat{n}$, $\widehat{l_1}$, $\widehat{bty}$, $\widehat{\sigma}_2$) $= (\widehat{\sigma}_3, 1)$ such that $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \hat{i})$, $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$, $\widehat{v} \neq$ skip, $\widehat{\gamma}(x) = (\widehat{l}$, const $\widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}$, const $\widehat{bty}*, 1$, PermL(Freeable, const $\widehat{bty}*$, public, 1)), DecodePtr(const $\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l_1}, 0)], [1], 1]$, $\widehat{\sigma}_2(\widehat{l_1}) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}$, PermL(Freeable, $\widehat{bty}$, public, $\widehat{n}$)), $(\hat{i} < 0) \vee (\hat{i} \geq \widehat{n})$, and WriteOOB($\widehat{v}$, $\hat{i}$, $\widehat{n}$, $\widehat{l_1}$, $\widehat{bty}$, $\widehat{\sigma}_2$) $= (\widehat{\sigma}_3, 1)$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \Downarrow'_{\widehat{wao}} (\widehat{\gamma}, \widehat{\sigma}_3, \square$, skip) by Vanilla C rule 1D Array Write Out of Bounds.

Given $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma, \sigma_3$, acc, skip$) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3, \square$, skip). Therefore, we have $(\gamma, \sigma$, acc, $x[e_1] = e_2) \Downarrow_{wao2} (\gamma, \sigma_3$, acc, skip$) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \Downarrow'_{\widehat{wao}} (\widehat{\gamma}, \widehat{\sigma}_3, \square$, skip), $\Pi \cong_\psi \Sigma$, and $wao2 \cong \widehat{wao}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma$, acc, $x[e_1] = e_2) \Downarrow_{wao1} (\gamma, \sigma_3$, acc, skip$)$

Given $\Pi \triangleright (\gamma, \sigma$, acc, $x[e_1] = e_2) \Downarrow_{wao1} (\gamma, \sigma_3$, acc, skip$)$ by SMC^2 rule Private 1D Array Write Public Value Out of Bounds Public Index, we have Label(e_1, γ) = Label(e_2, γ) = public, $(\gamma, \sigma$, acc, $e_1) \Downarrow_{d_1} (\gamma, \sigma_1$, acc, $i)$, $(\gamma, \sigma_1$, acc, $e_2) \Downarrow_{d_2} (\gamma, \sigma_2$, acc, $v)$, $v \neq$ skip, $\gamma(x) = (l$, private const $bty*)$, $\sigma_2(l) = (\omega$, private const $bty*, 1$, PermL(Freeable, private const $bty*$, private, 1)), DecodePtr(private const $bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1$, private bty, n, PermL(Freeable, private bty, private, n)), $(i < 0) \vee (i \geq n)$, and WriteOOB(encrypt(v), i, n, l_1, private $bty, \sigma_2) = (\sigma_3, 1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$ and ψ such that $(\gamma, \sigma$, acc, $x[e_1] = e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x[e_1] = e_2 \cong_\psi \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. Given $(\gamma, \sigma$, acc, $x[e_1] = e_2) \Downarrow_{wao1} (\gamma, \sigma_3$, acc, skip$)$, by Lemma 3.2.2 we have $(l, \mu) \notin x[e_1] = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1] = e_2 \cong \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. By Definition 3.2.10 we have Erase$(x[e_1] = e_2$ = Erase$(x[e_1])$ = Erase(e_2), Erase$(x[e_1]) = \widehat{x}[$Erase$(e_1)]$ where $x = \widehat{x}$, Erase$(e_1) = \widehat{e}_1$, and Erase$(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma$, acc, $e_1)$. Given $(\gamma, \sigma$, acc, $e_1) \Downarrow_{d_1} (\gamma, \sigma_1$, acc, $i)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1$, acc, $i) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and $d_1 \cong \widehat{d}_1$. Given $i \neq$ skip, by Lemma 3.2.1 we have

$\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\mathrm{Label}(e_1, \gamma) = \mathrm{public}$, we have $\mathrm{Label}(i, \gamma) = \mathrm{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \mathrm{acc}, e_2)$. Given $(\gamma, \sigma_1, \mathrm{acc}, e_2)\ \Downarrow_{d_2} (\gamma, \sigma_2, \mathrm{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{v})$ and ψ_2 such that $(\gamma, \sigma_2, \mathrm{acc}, v) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{v})$ and $d_2 \cong \widehat{d}_2$. Given $v \neq \mathrm{skip}$, by Lemma 3.2.1 we have $\psi_2 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$. Given $\mathrm{Label}(e_2, \gamma) = \mathrm{public}$, we have $\mathrm{Label}(v, \gamma) = \mathrm{public}$ and therefore $v = \widehat{v}$ by Definition 3.2.17.

Given $v \neq \mathrm{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \mathrm{skip}$.

Given $\gamma(x) = (l, \mathrm{private\ const}\ bty*)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \mathrm{const}\ \widehat{bty}*)$ such that $l = \widehat{l}$ by $\mathrm{private\ const}\ bty* \cong \mathrm{const}\ \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \mathrm{private\ const}\ bty*,\ 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private\ const}\ bty*, \mathrm{private}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty}*, \mathrm{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\mathrm{DecodePtr}(\mathrm{private\ const}\ bty*,\ 1,\ \omega) = [1, (l_1, 0), [1], 1]$, $\mathrm{private\ const}\ bty* \cong \mathrm{const}\ \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\mathrm{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \mathrm{private}\ bty, n, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty, \mathrm{private}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \mathrm{PermL}(\mathrm{Freeable},\ bty,\ \mathrm{public},\ \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\mathrm{private}\ bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $(i < 0) \vee (i \geq n)$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$.

Given $\mathrm{WriteOOB}(\mathrm{encrypt}(v), i, n, l_1, \mathrm{private}\ bty, \sigma_2) = (\sigma_3, 1)$ and $v = \widehat{v}$, by Definition 3.2.10 we have $\mathrm{encrypt}(v) \cong_\psi \widehat{v}$. Given $i = \widehat{i}$, $n = \widehat{n}$, $l_1 = \widehat{l}_1$, $\mathrm{private}\ bty \cong \widehat{bty}$, and $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, by Lemma 3.2.61 we have $\mathrm{WriteOOB}(\widehat{v},\ \widehat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty}, \widehat{\sigma}_2) = (\widehat{\sigma}_3,\ 1)$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)\ \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $\widehat{\gamma}(x) = (\widehat{l}, \mathrm{const}\ \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty}*, 1,$

PermL(Freeable, const $\widehat{bty}*$, public, 1)), DecodePtr(const $\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}$, PermL(Freeable, $\widehat{bty}$, public, $\widehat{n}$)), $(\hat{i} < 0) \vee (\hat{i} \geq \widehat{n})$, $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{\widehat{d}_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$, and WriteOOB($\widehat{v}$, $\hat{i}$, $\widehat{n}$, $\widehat{l}_1$, $\widehat{bty}, \widehat{\sigma}_2) = (\widehat{\sigma}_3, 1)$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \Downarrow'_{\widehat{wao}} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Write Out of Bounds.

Given $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$, by Definition 3.2.20 we have $(\gamma, \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao1} (\gamma, \sigma_3, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \Downarrow'_{\widehat{wao}} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wao1 \cong \widehat{wao}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{ra5} (\gamma, \sigma, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{ra5} (\gamma, \sigma, \text{acc}, [v_0, ..., v_{n-1}])$ by SMC^2 rule Private 1D Array Read Entire Array, we have $\gamma(x) = (l, \text{private const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{private}, 1))$, DecodePtr(private const $bty*$, 1, ω) $= [1, [(l_1, 0)], [1], 1]$, $\sigma(l_1) = (\omega_1, \text{private } bty, n, \text{PermL(Freeable, private } bty, \text{private}, n))$, and DecodeVal(private $bty, n, \omega_1) = [v_0, ..., v_{n-1}]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$ such that $(\gamma, \sigma, \text{acc}, x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have Erase$(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by private const $bty* \cong$ const $\widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{private}, 1))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL(Freeable, const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given DecodePtr(private const $bty*$, 1, ω) $= [1, (l_1, 0), [1], 1]$, private const $bty* \cong$ const $\widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have DecodePtr$(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma(l_1) = (\omega_1, \text{private } bty, n, \text{PermL(Freeable, private } bty, \text{private}, n))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL(Freeable, } bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, private $bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given DecodeVal(private bty, n, ω_1) $= [v_0, ..., v_{n-1}]$, private $bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have DecodeVal($bty, \widehat{n}, \widehat{\omega}_1$) $= [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL(Freeable, const } \widehat{bty}*, \text{public}, 1))$, DecodePtr(const $\widehat{bty}*, 1, \widehat{\omega}$) $= [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL(Freeable, } bty, \text{public}, \widehat{n}))$, and DecodeVal($bty, \widehat{n}, \widehat{\omega}_1$) $= [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{ra4}} (\widehat{\gamma}, \widehat{\sigma}, \square, [\widehat{v}_0, ..., \widehat{v}_{n-1}])$ by Vanilla C rule 1D Array Read Entire Array.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, by Definition 3.2.20 we have $(\gamma, \sigma, \text{acc}, [v_0, ..., v_{n-1}]) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, [\widehat{v}_0, ..., \widehat{v}_{n-1}])$. Therefore, we have $(\gamma, \sigma, \text{acc}, x) \Downarrow_{ra5} (\gamma, \sigma, \text{acc}, [v_0, ..., v_{n-1}]) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{ra4}} (\widehat{\gamma}, \widehat{\sigma}, \square, [\widehat{v}_0, ..., \widehat{v}_{n-1}])$, $\Pi \cong_\psi \Sigma$, and $ra5 \cong \widehat{ra4}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{ra4} (\gamma, \sigma, \text{acc}, [v_0, ..., v_{n-1}])$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{ra4} (\gamma, \sigma, \text{acc}, [v_0, ..., v_{n-1}])$ by SMC^2 rule Public 1D Array Read Entire Array, we have $\gamma(x) = (l, \text{public const } bty*)$, $\sigma(l) = (\omega, \text{public const } bty*, 1, \text{PermL(Freeable, public const } bty*, \text{public}, 1))$, DecodePtr(public const $bty*$, 1, ω) $= [1, [(l_1, 0)], [1], 1]$, $\sigma(l_1) = (\omega_1, \text{public } bty, n, \text{PermL(Freeable, public } bty, \text{public}, n))$, and DecodeVal(public bty, n, ω_1) $= [v_0, ..., v_{n-1}]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$ and ψ such that $(\gamma, \sigma, \text{acc}, x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have Erase(x) $= \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by public const $bty* \cong$ const $\widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{public const } bty*, 1, \text{PermL(Freeable, public const } bty*, \text{public}, 1))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL(Freeable, const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given DecodePtr(public const $bty*$, 1, ω) $= [1, (l_1, 0), [1], 1]$, public const $bty* \cong$ const $\widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have DecodePtr(const $\widehat{bty}*, 1, \widehat{\omega}$) $= [1, [(\widehat{l}_1, 0)], [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{public } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{public } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$, and $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{ra4}} (\widehat{\gamma}, \widehat{\sigma}, \square, [\widehat{v}_0, ..., \widehat{v}_{n-1}])$ by Vanilla C rule 1D Array Read Entire Array.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, by Definition 3.2.20 we have $(\gamma, \sigma, \text{acc}, [v_0, ..., v_{n-1}]) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, [\widehat{v}_0, ..., \widehat{v}_{n-1}])$. Therefore, we have $(\gamma, \sigma, \text{acc}, x) \Downarrow_{ra4} (\gamma, \sigma, \text{acc}, [v_0, ..., v_{n-1}]) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{\widehat{ra4}} (\widehat{\gamma}, \widehat{\sigma}, \square, [\widehat{v}_0, ..., \widehat{v}_{n-1}])$, $\Pi \cong_\psi \Sigma$, and $ra4 \cong \widehat{ra4}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wa5} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wa5} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Public 1D Array Write Entire Array, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}])$, $\forall v_m \in [v_0, ..., v_{n_e-1}]. v_m \neq \text{skip}$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $n_e = n$, and $\text{UpdateVal}(\sigma_1, l_1, [v_0, ..., v_{n_e-1}], \text{public } bty) = \sigma_2$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$ and ψ such that $(\gamma, \sigma, \text{acc}, x = e) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wa5} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \cong (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}])$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}]) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$ and $d_1 \cong \widehat{d}_1$. Given

$[v_0, ..., v_{n_e-1}] \neq$ skip, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $[v_0, ..., v_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$.

Given $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v_m \neq$ skip and $[v_0, ..., v_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Definition 3.2.10 we have $\forall \widehat{v}_m \in [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}].\ \widehat{v}_m \neq$ skip.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by public const $bty* \cong$ const $\widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega,\ \text{public const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given DecodePtr(public const $bty*$, 1, ω) $= [1, (l_1, 0), [1], 1]$, public const $bty* \cong$ const $\widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have DecodePtr(const $\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, public $bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $n_e = n$, $n = \widehat{n}$, and $n_e = \widehat{n}_e$, we have $\widehat{n}_e = \widehat{n}$.

Given UpdateVal($\sigma_1, l_1, [v_0, ..., v_{n_e-1}]$, public bty) $= \sigma_2$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l_1 = \widehat{l}_1$, public $bty \cong \widehat{bty}$, and $[v_0, ..., v_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Lemma 3.2.52 we have UpdateVal($\widehat{\sigma}_1, \widehat{l}_1, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}], \widehat{bty}) = \widehat{\sigma}_2$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$, $\forall \widehat{v}_m \in [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}].\ \widehat{v}_m \neq$ skip, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, DecodePtr(const $\widehat{bty}*$, 1, $\widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$, $\widehat{n}_e = \widehat{n}$, and UpdateVal($\widehat{\sigma}_1, \widehat{l}_1, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}], \widehat{bty}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wa5}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule 1D Array Write Entire Array.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wa5} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wa5}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wa5 \cong \widehat{wa5}$ by Definition 3.2.21.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e_1) \Downarrow_{wa6} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wa6} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by SMC^2 rule Private 1D Array Write Entire Private Array, we have $\text{Label}(e, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}])$, $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v_m \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1,\ \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $n_e = n$, and $\text{UpdateVal}(\sigma_1, l_1, [v_0, ..., v_{n_e-1}], \text{private } bty) = \sigma_2$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$ and ψ such that $(\gamma, \sigma, \text{acc}, x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wa6} (\gamma, \sigma_2, \text{acc}, \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}])$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}]) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$ and $d_1 \cong \widehat{d}_1$. Given $[v_0, ..., v_{n_e-1}] \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $[v_0, ..., v_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$.

Given $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v_m \neq \text{skip}$ and $[v_0, ..., v_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Definition 3.2.18 and Definition 3.2.10 we have $\forall \widehat{v}_m \in [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}].\ \widehat{v}_m \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given DecodePtr(private const $bty*$, 1, ω) = [1, (l_1, 0), [1], 1], private const $bty*$ $\cong$ const $\widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have DecodePtr($\widehat{bty}*, 1, \widehat{\omega}$) = [1, [($\widehat{l}_1$, 0)], [1], 1] where [1, (l_1, 0), [1], 1] $\cong_\psi$ [1, [($\widehat{l}_1$, 0)], [1], 1] such that (l_1, 0) $\cong_\psi$ ($\widehat{l}_1$, 0).

Given $\sigma_1(l_1)$ = (ω_1, private bty, n, PermL(Freeable, private bty, private, n)), (γ, σ_1) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$), and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1)$ = ($\widehat{\omega}_1$, $\widehat{bty}$, $\widehat{n}$, PermL(Freeable, bty, public, $\widehat{n}$)) where $\omega_1 \cong_\psi \widehat{\omega}_1$, private bty $\cong$ $\widehat{bty}$, and $n = \widehat{n}$.

Given $n_e = n$, $n = \widehat{n}$, and $n_e = \widehat{n}_e$, we have $\widehat{n}_e = \widehat{n}$.

Given UpdateVal(σ_1, l_1, [v_0, ..., v_{n_e-1}], private bty) = σ_2, (γ, σ_1) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$), $l_1 = \widehat{l}_1$, private bty $\cong$ $\widehat{bty}$, and [v_0, ..., v_{n_e-1}] $\cong_\psi$ [$\widehat{v}_0$, ..., $\widehat{v}_{\widehat{n}_e-1}$], by Lemma 3.2.52 we have UpdateVal($\widehat{\sigma}_1$, $\widehat{l}_1$, [$\widehat{v}_0$, ..., $\widehat{v}_{\widehat{n}_e-1}$], $\widehat{bty}$) = $\widehat{\sigma}_2$ such that (γ, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$).

Given ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{x} = \widehat{e}$), ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}$) $\Downarrow'_{\widehat{d}_1}$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, [$\widehat{v}_0$, ..., $\widehat{v}_{\widehat{n}_e-1}$]), $\forall \widehat{v}_m \in$ [$\widehat{v}_0$, ..., $\widehat{v}_{\widehat{n}_e-1}$]. $\widehat{v}_m \neq$ skip, $\widehat{\gamma}(\widehat{x})$ = ($\widehat{l}$, const $\widehat{bty}*$), $\widehat{\sigma}_1(\widehat{l})$ = ($\widehat{\omega}$, const $\widehat{bty}*$, 1, PermL(Freeable, const $\widehat{bty}*$, public, 1)), DecodePtr(const $\widehat{bty}*, 1, \widehat{\omega}$) = [1, [($\widehat{l}_1$, 0)], [1], 1], $\widehat{\sigma}_1(\widehat{l}_1)$ = ($\widehat{\omega}_1$, $\widehat{bty}$, $\widehat{n}$, PermL(Freeable, bty, public, $\widehat{n}$)), $\widehat{n}_e = \widehat{n}$, and UpdateVal($\widehat{\sigma}_1$, $\widehat{l}_1$, [$\widehat{v}_0$, ..., $\widehat{v}_{\widehat{n}_e-1}$], $\widehat{bty}$) = $\widehat{\sigma}_2$, we have $\Sigma \triangleright$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{x} = \widehat{e}$) $\Downarrow'_{\widehat{wa5}}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, skip) by Vanilla C rule 1D Array Write Entire Array.

Given (γ, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$), by Definition 3.2.20 we have (γ, σ_2, acc, skip) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, skip). Therefore, we have (γ, σ, acc, $x = e$) $\Downarrow_{wa6}$ (γ, σ_2, acc, skip) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{x} = \widehat{e}$) $\Downarrow'_{\widehat{wa5}}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, skip), $\Pi \cong_\psi \Sigma$, and $wa6 \cong \widehat{wa5}$ by Definition 3.2.21.

Case $\Pi \triangleright$ (γ, σ, acc, $x = e$) $\Downarrow_{wa7}$ (γ, σ_2, acc, skip)

Given $\Pi \triangleright$ (γ, σ, acc, $x = e$) $\Downarrow_{wa7}$ (γ, σ_2, acc, skip) by SMC^2 rule Private 1D Array Write Entire Public Array, we have Label(e, γ) = public, (γ, σ, acc, e) $\Downarrow_{d_1}$ (γ, σ_1, acc, [v_0, ..., v_{n_e-1}]), $\forall v_m \in$ [v_0, ..., v_{n_e-1}]. $v_m \neq$ skip, $\gamma(x)$ = (l, private const $bty*$), (bty = int) $\vee$ (bty = float), $\sigma_1(l)$ = (ω, private const $bty*$, 1, PermL(Freeable, private const $bty*$, private, 1)), DecodePtr(private const $bty*, 1, \omega$) = [1, [(l_1, 0)], [1], 1], $\forall v_m \in$ [v_0, ..., v_{n_e-1}]. v'_m = encrypt(v_m), $\sigma_1(l_1)$ = (ω_1, private bty, n, PermL(Freeable, private bty, private, n)), $n_e = n$, and UpdateVal(σ_1, l_1, [v'_0, ..., v'_{n_e-1}], private bty) = σ_2.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \text{acc},\ x = e) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, by Definition 3.2.20 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_{\psi} \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \text{acc},\ x = e) \Downarrow_{wa7} (\gamma,\ \sigma_2,\ \text{acc},\ \text{skip})$, by Lemma 3.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 3.2.4 we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \cong_{\psi} (\gamma, \sigma, \text{acc}, e)$. Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{d_1} (\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}])$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$ and ψ_1 such that $(\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}]) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$ and $d_1 \cong \widehat{d}_1$. Given $[v_0, ..., v_{n_e-1}] \neq \text{skip}$, by Lemma 3.2.1 we have $\psi_1 = \psi$. By Definition 3.2.20 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $[v_0, ..., v_{n_e-1}] \cong_{\psi} [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$. By Lemma 3.2.18, we have $n_e = \widehat{n}_e$. Given $\text{Label}(e, \gamma) == \text{public}$, we have $\text{Label}([v_0, ..., v_{n_e-1}], \gamma) == \text{public}$ and therefore $[v_0, ..., v_{n_e-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$ by Definition 3.2.18 and Definition 3.2.10.

Given $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v_m \neq \text{skip}$ and $[v_0, ..., v_{n_e-1}] \cong_{\psi} [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Definition 3.2.18 and Definition 3.2.10 we have $\forall \widehat{v}_m \in [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}].\ \widehat{v}_m \neq \text{skip}$.

Given $\gamma(x) = (l,\ \text{private const } bty*)$, $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma, \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_{\psi} \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1, (l_1, 0), [1], 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_{\psi} \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_{\psi} [1, [(\widehat{l}_1, 0)], [1], 1]$ such that $(l_1, 0) \cong_{\psi} (\widehat{l}_1, 0)$.

Given $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v'_m = \text{encrypt}(v_m)$ and $[v_0, ..., v_{n_e-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Definition 3.2.10 and Definition 3.2.18 we have $[v'_0, ..., v'_{n_e-1}] \cong_{\psi} [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$.

Given $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_{\psi} \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $n_e == n$, $n = \widehat{n}$, and $n_e = \widehat{n}_e$, we have $\widehat{n}_e == \widehat{n}$.

Given $\text{UpdateVal}(\sigma_1, l_1, [v'_0, ..., v'_{n_e-1}], \text{private } bty) = \sigma_2$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l_1 = \widehat{l}_1$, $\text{private } bty \cong \widehat{bty}$, and $[v'_0, ..., v'_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Lemma 3.2.52 we have $\text{UpdateVal}(\widehat{\sigma}_1, \widehat{l}_1, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}], \widehat{bty}) = \widehat{\sigma}_2$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{\widehat{d}_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$, $\forall \widehat{v}_m \in [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}].\ \widehat{v}_m \neq \text{skip}$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$, $\widehat{n}_e == \widehat{n}$, and $\text{UpdateVal}(\widehat{\sigma}_1, \widehat{l}_1, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}], \widehat{bty}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wa5}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule 1D Array Write Entire Array.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 3.2.20 we have $(\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \text{acc}, x = e) \Downarrow_{wa7} (\gamma, \sigma_2, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{\widehat{wa5}} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wa7 \cong \widehat{wa5}$ by Definition 3.2.21.

$\square$

3.3 Noninterference

Basic SMC^2 satisfies a strong form of noninterferences guaranteeing that two execution traces are indistinguishable up to differences in private values. This stronger version entails data-obliviousness. Instead of using execution traces, we will work directly with evaluation trees in the Basic SMC^2 semantics – equivalence of evaluation trees up to private values implies equivalence of execution traces based on the Basic SMC^2 semantics. This guarantee is provided at the semantics level, we do not consider here compiler optimizations.

For noninterference, it is convenient to introduce a notion of equivalence requiring that the two memories agree on publicly observable values. Because we assume that private data in memories are encrypted, and so their encrypted value is publicly observable, it is sufficient to consider syntactic equality of memories. Notice that if $\sigma_1 = \sigma_2$ we can still have $\sigma_1\ell \neq \sigma_2\ell$, i.e., two executions starting from the same configuration can actually differ with respect to private data. What we show is that this difference can occur only in atomic operations working on private data, which we assume is not publicly observable.

We want to consider two evaluation trees as *low-equivalent* if they are identical up to private relational operations. To formalize this, we need first to identify codes up to private relational operations – these are

atomic operations that are implemented by means of some cryptographic primitive and we assume that their difference is not publicly observable. We define *low-equivalence* over Basic SMC^2 evaluation codes in Definition 3.3.1 and evaluation trees in Definition 3.3.3. Based on the notion of low-equivalence between evaluation trees, we can now state our main noninterference result.

Theorem 3.3.1 (Noninterference over evaluation trees)**.** *For every environment γ, γ', γ''; memory σ, σ', $\sigma'' \in$ Mem; accumulator acc, acc', $\mathrm{acc}'' \in \mathbb{N}$; statement s, values v', v''; and step evaluation codes $[c'_1, ..., c'_n], [c''_1, ..., c''_n]$; if $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, s) \Downarrow^{[\bar{l}'_1,...,\bar{l}'_n]}_{[c'_1,...,c'_n]} (\gamma', \sigma', \mathrm{acc}', v')$ and $\Sigma \triangleright (\gamma, \sigma, \mathrm{acc}, s) \Downarrow_{[c''_1,...,c''_n]} (\gamma'', \sigma'', \mathrm{acc}'', v'')$, then $\gamma' = \gamma''$, $\sigma' = \sigma''$, $\mathrm{acc}' = \mathrm{acc}''$, $v' = v''$, $[c'_1, ..., c'_n] \simeq_L [c''_1, ..., c''_n]$, and $\Pi \simeq_L \Sigma$.*

Proof. Proof Sketch: By induction over all Basic SMC^2 semantic rules. Notice that low-equivalence of evaluation trees already implies the equivalence of the resulting configurations. We repeated them to make the meaning of the theorem clearer. Moreover, notice that two evaluation trees can differ only in atomic operations implemented through cryptographic primitives. Thus the two corresponding traces are equivalent and data-obliviousness follows.

We make the assumption that both evaluation traces are over the same program (this is given by having the same s in the starting states) and all public data will remain the same, including data read as input during the evaluation of the program. A portion of the complexity of this proof is within ensuring that memory accesses within our semantics remain data oblivious. Several rules follow fairly simply and leverage similar ideas, which we will discuss first, and then we will provide further intuition behind the more complex cases. The full proof is available in Section 3.3.2, with this theorem identical to Theorem 3.3.2.

For all rules leveraging helper algorithms, we must reason about the helper algorithms, and that they behave deterministically by definition and have data-oblivious memory accesses. Given this and that these helper algorithms do no modify the private data, we maintain the properties of noninterference of this theorem. First we reason that our helper algorithms to translate values into their byte representation will do so deterministically, and therefore maintain indistinguishability between the value and byte representation. We can then reason that our helper algorithms that take these byte values and store them into memory will also do so deterministically, so that when we later access the data in memory we will obtain the same indistinguishable values we had stored.

It is also important to take note here our functions to help us retrieve data from memory, particularly in cases such as when reading out of bounds of an array. When proving these cases to maintain noninterference, we leverage our definition of how memory blocks are assigned in a monotonically increasing fashion, and how the algorithms for choosing which memory block to read into after the current one are deterministic. This, as well as our original assumptions of having identical public input, allows us to reason that if we access out of bounds (including accessing data at a non-aligned position, such as a chunk of bytes in the middle of a memory block), we will be pulling from the same set of bytes each time, and therefore we will end up with the same interpretation of the data as we continue to evaluate the remainder of the program. It is important to note again here that by definition, our semantics will always

interpret bytes of data as the type it is expected to be, not the type it actually is (i.e., reading bytes of data that marked private in memory by overshooting a public array will not decrypt the bytes of data, but instead give you back a garbage public value). To reiterate this point, even when reading out of bounds, we will not reveal anything about private data, as the results of these helper algorithms will be indistinguishable.

For private pointers, it is important to note that the obtaining multiple locations is deterministic based upon the program that is being evaluated. A pointer can initially gain multiple locations through the evaluation of a private if else. Once there exists a pointer that has obtained multiple locations in such a way, it can be assigned to another pointer to give that pointer multiple locations. The other case for a pointer to gain multiple location is through the use of `pfree` on a pointer with multiple locations (i.e., the case where a pointer has locations l_1, l_2, l_3 and we free l_1) - when this occurs, if another pointer had referred to only l_1, it will now gain locations in order to mask whether we had to move the true location or not. When reasoning about pointers with multiple locations, we maintain that given the tags for which location is the true location are indistinguishable, then it is not possible to distinguish between them by their usage as defined in the rules or helper algorithms using them. Additionally, to reason about `pfree`, we leverage that the definitions of the helper algorithms are deterministic, and that (wlog), we will be freeing the same location. We will then leverage our Axiom about the multiparty protocol MPC_{free}. After the evaluation of MPC_{free}, it will deterministically update memory and all other pointers as we mentioned in the brief example above.

For the Private If Else rule, the most important element we must leverage is how values are resolved, showing that given our resolution style, we are not able to distinguish between the ending values. In order to do this, we also must reason about the entirety of the rule, including all of if else helper algorithms. First, we note that the evaluation of the **`then`** branches follows by induction, as does the evaluation of the **`else`** branch once we have reasoned through the restoration phase. It is clear from the definitions of $\mathrm{ExtractVariables}$, $\mathrm{InitializeVariables}$, and $\mathrm{RestoreVariables}$ that the behavior of these algorithms is deterministic and given the same program, we will be extracting, initializing, and restoring the same set variables every time we evaluate the program. Now, we are able to move on to reasoning about resolution, and show that given all of this and the definitions of the resolution helper algorithms and rule, we are not able to distinguish between the ending values.

Within the array rules, the main concern is in reading from and writing at a private index. We currently handle this complexity within our rules by accessing all locations within the array in rules Array Read Private Index and Array Write Private Index. In Array Read Private Index, we clearly read data from every index of the array, privately computing the true value from all values in the array. Similarly, in Array Write Private Index, we read data from every index of the array, then proceed to privately update every value of in array. All other array rules use public indices, and in turn only access that publicly known location. Within the pointer rules, our main concern is that we access all locations that are referred to by a private pointer when we have multiple locations. For this, we will reason about the contents of the rules and the helper algorithms used by the pointer rules, which can be shown to deterministically do so.

□

3.3.1 Supporting Metatheory

Definition 3.3.1. We define *low-equivalence* over codes by cases as follows: [1] if $c = c'$, then $c \simeq_L c'$; if the codes are private less than ($<$) operations, we also have: $ltt1 \simeq_L ltf1$, $ltt2 \simeq_L ltf2$, $ltt3 \simeq_L ltf3$; if the codes are private equal to ($==$) operations, we also have: $eqt1 \simeq_L eqf1$, $eqt2 \simeq_L eqf2$, $eqt3 \simeq_L eqf3$; if the codes are private not equal to ($!=$) operations, we also have: $net1 \simeq_L nef1$, $net2 \simeq_L nef2$, $net3 \simeq_L nef3$.

Definition 3.3.2. We define *low-equivalence* over two sets of evaluation codes $[c_1, ..., c_n], [c'_1, ..., c'_n]$ if and only if for every code $c_m \in [c_1, ..., c_n]$ and $c'_m \in [c'_1, ..., c'_n]$, $c_m \simeq_L c'_m$.

Definition 3.3.3. Two SMC[2] evaluation trees Π and Σ are *low-equivalent*, in symbols $\Pi \simeq_L \Sigma$, if and only if Π and Σ have the same structure as trees, and for each node in Π proving $(\gamma,\ \sigma,\ \mathrm{acc},\ s) \Downarrow_{c_\Pi} (\gamma_1,\ \sigma_1,\ \mathrm{acc}_1,\ v)$, the corresponding node in Σ proves $(\gamma,\ \sigma, \mathrm{acc}, s) \Downarrow_{c_\Sigma} (\gamma_1,\ \sigma_1, \mathrm{acc}_1,\ v)$ and $c_\Pi \cong_L c_\Sigma$.

Definition 3.3.4. Given input files $input1, input2$, $input1 = input2$ if and only if

- for every public variable x, if $\{x = n\} \in input1$ then $\{x = n\} \in input2$,
- for every public array x, if $\{x = n_0, ..., n_m\} \in input1$ then $\{x = n_0, ..., n_m\} \in input2$,
- for every private variable x, if $\{x = n\} \in input1$ then $\{x = n'\} \in input2$ such that $n = n'$ by Axiom 3.3.1, and
- for every private array x, if $\{x = n_0, ..., n_m\} \in input1$ then $\{x = n'_0, ..., n'_m\} \in input2$ such that for every index i in $0...m$, $n_i = n'_i$ by Axiom 3.3.1.

Definition 3.3.5. We define the function ϕ to return a single unused memory block identifier in a monotonically increasing fashion.

Axiom 3.3.1 (Encrypt)**.** *Given the use of an encryption scheme that ensures encrypted numbers are indistinguishable, we assume that given any two numbers n_1, n_2, their respective encrypted values n'_1, n'_2 can be viewed as equivalent.*

Axiom 3.3.2 (Private Operations)**.** *Given Axiom 3.3.1, when performing a private operation on two sets of encrypted values n_1, n'_1 and n_2, n'_2, we will have resulting encrypted values n_3, n'_3 that can also be viewed as equivalent.*

Axiom 3.3.3 (InputValue)**.** *Given two input files $input1, input2$ and variable x corresponding to a program of statement s, if and only if $input1 = input2$ by Definition 3.3.4 then $\mathrm{InputValue}(x, input1) = n$ and $\mathrm{InputValue}(x, input2) = n'$ such that $n = n'$.*

[1] In these codes, the first two letters denote the operation (less than - lt, equal to - et, not equal to - ne); the third letter denotes true (t) or false (f); and the ending number denotes what mix of public and private data is being operated over (1 - both operands are private, 2 - left operand is public, right operand is private, 3 - left operand is private, right operand is public).

Axiom 3.3.4 (InputArray)**.** *Given two input files* $input1, input2$ *and array* x *of length* m *corresponding to a program of statement* s*, if and only if* $input1 = input2$ *by Definition 3.3.4 then* $\text{InputArray}(x, input1, m) = [n_0, ..., n_{m-1}]$ *and* $\text{InputArray}(x, input2, m) = [n'_0, ..., n'_{m-1}]$ *such that for every index* i *in* $0...m$*,* $n_i = n'_i$*.*

Axiom 3.3.5 (ϕ)**.** *Given a program of statement* s*, during any two executions* Π, Σ *over* s *such that* $\Pi \simeq_L \Sigma$ *by Definition 5.3.2, if* ϕ *returns memory block identifier* l *at step* c *in* Π*, then by definition 3.3.5* ϕ *will also return* l *at step* c *in* Σ*.*

Lemma 3.3.1. *Given parameter list* $\overline{p}, \overline{p}'$*,*
if $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$*,* $\text{GetFunTypeList}(\overline{p}') = \overline{ty}'$*, and* $\overline{p} = \overline{p}'$*, then* $\overline{ty} = \overline{ty}'$*.*

Proof. By definition of Algorithm GetFunTypeList, GetFunTypeList is deterministic. □

Lemma 3.3.2. *Given parameter list* $\overline{p}, \overline{p}'$*, expression list* $\overline{e}, \overline{e}'$*,*
if $\text{GetFunParamAssign}(\overline{p}, \overline{e}) = s$*,* $\text{GetFunParamAssign}(\overline{p}', \overline{e}') = s'$*,* $\overline{p} = \overline{p}'$*, and* $\overline{e} = \overline{e}'$*, then* $s = s'$*.*

Proof. By definition of Algorithm GetFunParamAssign, GetFunParamAssign is deterministic. □

Lemma 3.3.3. *Given type* $ty, ty' \in \{a\ bty\}$*, and value* v, v'*,*
if $\text{EncodeVal}(ty, v) = \omega$*,* $\text{EncodeVal}(ty', v') = \omega'$*,* $ty = ty'$*, and* $v = v'$*, then* $\omega = \omega'$*.*

Proof. By definition of Algorithm EncodeVal, EncodeVal is deterministic. □

Lemma 3.3.4. *Given type* $ty,\ ty' \in \{a\ bty\}$*, number* $n,\ n'$*, and byte representation* $\omega,\ \omega'$*,*
if $\text{DecodeVal}(ty, n, \omega) = v$*,* $\text{DecodeVal}(ty', n', \omega') = v'$*,* $ty = ty'$*,* $n = n'$*, and* $\omega = \omega'$*, then* $v = v'$*.*

Proof. By definition of Algorithm DecodeVal, DecodeVal is deterministic. □

Lemma 3.3.5. *Given pointer type* $ty, ty' \in \{a\ \text{const}\ bty*, a\ bty*\}$*, and pointer data structure* $[\alpha, \overline{l}, \overline{j}, n]$*,* $[\alpha', \overline{l}', \overline{j}', n']$*, if* $\text{EncodePtr}(ty, [\alpha, \overline{l}, \overline{j}, n]) = \omega$*,* $\text{EncodePtr}(ty', [\alpha', \overline{l}', \overline{j}', n']) = \omega'$*,* $ty = ty'$*, and* $[\alpha, \overline{l}, \overline{j}, n] = [\alpha', \overline{l}', \overline{j}', n']$*, then* $\omega = \omega'$*.*

Proof. By definition of Algorithm EncodePtr, EncodePtr is deterministic. □

Lemma 3.3.6. *Given pointer type* $ty, ty' \in \{a\ \text{const}\ bty*, a\ bty*\}$*, number* α, α'*, and byte representation* ω, ω'*, if* $\text{DecodePtr}(ty, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, n]$*,* $\text{DecodePtr}(ty', \alpha', \omega') = [\alpha', \overline{l}', \overline{j}', n']$*,* $ty = ty'$*,* $\alpha = \alpha'$ *and* $\omega = \omega'$*, then* $[\alpha, \overline{l}, \overline{j}, n] = [\alpha', \overline{l}', \overline{j}', n']$*.*

Proof. By definition of Algorithm DecodePtr, DecodePtr is deterministic. □

Lemma 3.3.7. *Given statement* s, s'*, number* n, n'*, and parameter list* $\overline{p}, \overline{p}'$*,*
if $\text{EncodeFun}(s, n, \overline{p}) = \omega$*,* $\text{EncodeFun}(s', n', \overline{p}') = \omega'$*,* $s = s'$*,* $n = n'$*, and* $\overline{p} = \overline{p}'$*, then* $\omega = \omega'$*.*

Proof. By definition of Algorithm EncodeFun, EncodeFun is deterministic. □

Lemma 3.3.8. *Given byte representation* ω, ω'*, if* $\mathrm{DecodeFun}(\omega) = (s, n, \overline{p})$*,* $\mathrm{DecodeFun}(\omega') = (s', n', \overline{p}')$*,* $ty = ty'$*,* $n = n'$*, and* $\omega = \omega'$*, then* $s = s'$*,* $n = n'$*, and* $\overline{p} = \overline{p}'$*.*

Proof. By definition of Algorithm DecodeFun, DecodeFun is deterministic. □

Lemma 3.3.9. *Given memory* σ_1, σ_1'*, memory block identifier* l, l'*, and environment* γ, γ'*, if* $\mathrm{Free}(\sigma_1, l, \gamma) = \sigma_2'$*,* $\mathrm{Free}(\sigma_1', l', \gamma') = \sigma_2'$*,* $\sigma_1 = \sigma_1'$*,* $l = l'$*, and* $\gamma = \gamma'$*, then* $\sigma_2 = \sigma_2'$*.*

Proof. By definition of Algorithm Free, Free is deterministic. □

Lemma 3.3.10. *Given environment* γ, γ'*, memory* σ_1, σ_1' *and memory block identifier* l, l'*, if* $\mathrm{PFree}(\gamma, \sigma_1, l) = (\sigma_2, \overline{j})$*,* $\mathrm{PFree}(\gamma', \sigma_1', l') = (\sigma_2', \overline{j}')$*,* $\gamma = \gamma'$*,* $\sigma_1 = \sigma_1'$*, and* $l = l'$*, then* $\sigma_2 = \sigma_2'$ *and* $\overline{j} = \overline{j}'$*.*

Proof. By definition of Algorithm PFree, PFree is deterministic. By Axiom 3.3.1, given that private tags are always encrypted, $\overline{j} = \overline{j}'$. □

Lemma 3.3.11. *Given pointer indirection level indicator* $*, *'$*, if* $\mathrm{GetIndirection}(*) = i$*,* $\mathrm{GetIndirection}(*') = i'$*, and* $* = *'$*, then* $i = i'$*.*

Proof. By definition of Algorithm GetIndirection, GetIndirection is deterministic. □

Lemma 3.3.12. *Given memory* $\sigma_1,\ \sigma_1'$*, memory block identifier* $l,\ l'$*, value* $v,\ v'$*, and* $ty,\ ty'$*, if* $\mathrm{UpdateVal}(\sigma_1, l, v, ty) = \sigma_2$*,* $\mathrm{UpdateVal}(\sigma_1', l', v', ty') = \sigma_2'$*,* $\sigma_1 = \sigma_1'$*,* $l = l'$*,* $v = v'$*, and* $ty = ty'$*, then* $\sigma_2 = \sigma_2'$*.*

Proof. By definition of Algorithm UpdateVal, UpdateVal is deterministic. □

Lemma 3.3.13. *Given memory* $\sigma_1,\ \sigma_1'$*, location* $(l,\ \mu),\ (l',\ \mu')$*, value* $v,\ v'$*, and type* $ty,\ ty'$*, if* $\mathrm{UpdateOffset}(\sigma_1, (l, \mu), v, ty) = (\sigma_2, j)$*,* $\mathrm{UpdateOffset}(\sigma_1', (l', \mu'), v', ty') = (\sigma_2', j')$*,* $\sigma_1 = \sigma_1'$*,* $(l, \mu) = (l', \mu')$*,* $v = v'$*, and* $ty = ty'$*, then* $\sigma_2 = \sigma_2'$ *and* $j = j'$*.*

Proof. By definition of Algorithm UpdateOffset, UpdateOffset is deterministic. □

Lemma 3.3.14. *Given memory* σ_1, σ_1'*, memory block identifier list* $\overline{l}, \overline{l}'$*, tag list* $\overline{j}, \overline{j}'$*, type* ty, ty'*, and value* v_1*,* v_2, v_1', v_2'*, if* $\mathrm{UpdatePriv}(\sigma_1,\ v_1,\ \overline{l},\ \overline{j},\ ty,\ v_2) = (\sigma_2,\ j)$*,* $\mathrm{UpdatePriv}(\sigma_1,\ v_1',\ \overline{l}',\ \overline{j}',\ ty',\ v_2') = (\sigma_2',\ j')$*,* $\sigma_1 = \sigma_1'$*,* $v_1 = v_1'$*,* $\overline{l} = \overline{l}'$*,* $\overline{j} = \overline{j}'$*,* $ty = ty'$*, and* $v_2 = v_2'$*, then* $\sigma_2 = \sigma_2'$ *and* $j = j'$*.*

Proof. By definition of Algorithm UpdatePriv, UpdatePriv is deterministic. □

Lemma 3.3.15. *Given memory* σ_1, σ_1'*, location* $(l, \mu), (l', \mu')$*, pointer data structure* $[\alpha, \overline{l}, \overline{j}, i]$*,* $[\alpha', \overline{l}', \overline{j}', i']$*, and type* ty, ty' *if* $\mathrm{UpdatePrivPtr}(\sigma_1, (l, \mu), [\alpha, \overline{l}, \overline{j}, i], ty) = (\sigma_2, j)$*,* $\mathrm{UpdatePrivPtr}(\sigma_1, (l', \mu'), [\alpha', \overline{l}', \overline{j}', i'], ty') = (\sigma_2', j')$*,* $\sigma_1 = \sigma_1'$*,* $(l, \mu) = (l', \mu')$*,* $[\alpha, \overline{l}, \overline{j}, i] = [\alpha', \overline{l}', \overline{j}', i']$*, and* $ty = ty'$*, then* $\sigma_2 = \sigma_2'$ *and* $j = j'$*.*

Proof. By definition of Algorithm UpdatePtr, UpdatePtr is deterministic. □

Lemma 3.3.16. *Given memory* σ_1, σ_1'*, location* $(l, \mu), (l', \mu')$*, pointer data structure* $[\alpha, \bar{l}, \bar{j}, i]$*,* $[\alpha', \bar{l}', \bar{j}', i']$*, and type* ty, ty'*, if* $\text{UpdatePrivPtr}(\sigma_1, [\alpha, \bar{l}, \bar{j}, i], (l, \mu), ty) = (\sigma_2, j)$*,* $\text{UpdatePrivPtr}(\sigma_1, [\alpha', \bar{l}', \bar{j}', i'], (l', \mu'), ty') = (\sigma_2', j')$*,* $\sigma_1 = \sigma_1'$*,* $(l, \mu) = (l', \mu')$*,* $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$*, and* $ty = ty'$*, then* $\sigma_2 = \sigma_2'$ *and* $j = j'$*.*

Proof. By definition of Algorithm UpdatePrivPtr, UpdatePrivPtr is deterministic. □

Lemma 3.3.17. *Given number* α, α'*, location list* $\bar{l}, \bar{l}'$*, tag list* $\bar{j}, \bar{j}'$*, type* ty, ty' *and memory* σ, σ'*, if* $\text{Retrieve_vals}(\alpha, \bar{l}, \bar{j}, ty, \sigma) = (v, j)$*,* $\text{Retrieve_vals}(\alpha', \bar{l}', \bar{j}', ty', \sigma') = (v', j')$*,* $\alpha = \alpha'$*,* $\bar{l} = \bar{l}'$*,* $\bar{j} = \bar{j}'$*,* $ty = ty'$ *and* $\sigma = \sigma'$*, then* $v = v'$ *and* $j = j'$*.*

Proof. By definition of Algorithm Retrieve_vals, Retrieve_vals is deterministic. □

Lemma 3.3.18. *Given number* α_1, α_1'*, location list* $\bar{l}_1, \bar{l}_1'$*, tag list* $\bar{j}_1, \bar{j}_1'$*, type* ty, ty' *and memory* σ, σ'*, if* $\text{DerefPrivPtr}(\alpha_1, \bar{l}_1, \bar{j}_1, ty, \sigma) = ((\alpha_2, \bar{l}_2, \bar{j}_2), j)$*,* $\text{DerefPrivPtr}(\alpha_1', \bar{l}_1', \bar{j}_1', ty', \sigma') = ((\alpha_2', \bar{l}_2', \bar{j}_2'), j')$*,* $\alpha_1 = \alpha_1'$*,* $\bar{l}_1 = \bar{l}_1'$*,* $\bar{j}_1 = \bar{j}_1'$*,* $ty = ty'$ *and* $\sigma = \sigma'$*, then* $(\alpha_2, \bar{l}_2, \bar{j}_2) = (\alpha_2', \bar{l}_2', \bar{j}_2')$ *and* $j = j'$*.*

Proof. By definition of Algorithm DerefPrivPtr, DerefPrivPtr is deterministic. □

Lemma 3.3.19. *Given memory* σ, σ'*, type* public bty, public bty'*, and location* $(l_1, \mu_1), (l_1', \mu_1')$*, if* $\text{DerefPtr}(\sigma,\ \text{public } bty, (l_1,\ \mu_1)) = (v,\ j)$*,* $\sigma = \sigma'$*,* $bty = bty'$*, and* $(l_1,\ \mu_1) = (l_1', \mu_1')$*, then* $\text{DerefPtr}(\sigma', \text{public } bty', (l_1', \mu_1')) = (v', j')$ *such that* $v = v'$ *and* $j = j'$*.*

Proof. By definition of Algorithm DerefPtr, DerefPtr is deterministic. □

Lemma 3.3.20. *Given memory* σ, σ'*, type* public $bty*$, public $bty'*$*, and location* $(l_1, \mu_1), (l_1', \mu_1')$*, if* $\text{DerefPtrHLI}(\sigma, \text{public } bty*, (l_1, \mu_1)) = ([1, [(l_2, \mu_2)], [1], i], j)$*, then* $\text{DerefPtrHLI}(\sigma', \text{public } bty'*, (l_1', \mu_1')) = ([1, [(l_2', \mu_2')], [1], i'], j')$ *such that* $[1, [(l_2, \mu_2)], [1], i] = [1, [(l_2', \mu_2')], [1], i']$ *and* $j = j'$*.*

Proof. By definition of Algorithm DerefPtrHLI, DerefPtrHLI is deterministic. □

Lemma 3.3.21. *Given location* $(l, \mu), (l', \mu')$*, number* n, n'*, and memory* σ, σ'*, if* $\text{GetLocation}((l, \mu),\ n,\ \sigma) = ((l_1, \mu_1),\ j)$*,* $\text{GetLocation}((l', \mu'),\ n',\ \sigma') = ((l_1', \mu_1'),\ j')$*,* $(l, \mu) = (l', \mu')$*,* $n = n'$*, and* $\sigma = \sigma'$*, then* $(l_1, \mu_1) = (l_1', \mu_1')$ *and* $j = j'$*.*

Proof. By definition of Algorithm GetLocation, GetLocation is deterministic. □

Lemma 3.3.22. *Given type* ty, ty'*, if* $\tau(ty) = n$*,* $\tau(ty') = n'$*, and* $ty = ty'$*, then* $n = n'$*.*

Proof. By definition of Algorithm τ, τ is deterministic. □

Lemma 3.3.23. *Given location list* $\bar{l}, \bar{l}'$*, number* n, n'*, and memory* σ, σ'*, if* $\text{IncrementList}(\bar{l},\ n,\ \sigma) = (\bar{l}_1,\ j)$*,* $\text{IncrementList}(\bar{l}, n', \sigma) = (\bar{l}'_1, j')$*,* $\bar{l} = \bar{l}'$*,* $n = n'$*, and* $\sigma = \sigma'$*, then* $\bar{l}_1 = \bar{l}'_1$ *and* $j = j'$*.*

Proof. By definition of Algorithm IncrementList, IncrementList is deterministic. □

Lemma 3.3.24. *Given number* n_1, n'_1, n_2, n'_2*, memory block identifier* l, l'*, type* ty, ty'*, and memory* σ, σ'*, if* $\text{ReadOOB}(n_1, n_2, l, ty, \sigma) = (v, j)$*,* $\text{ReadOOB}(n'_1, n'_2, l', ty', \sigma') = (v', j')$*,* $n_1 = n'_1$*,* $n_2 = n'_2$*,* $l = l'$*,* $ty = ty'$*, and* $\sigma = \sigma'$*, then* $v = v'$ *and* $j = j'$*.*

Proof. By definition of Algorithm ReadOOB, ReadOOB is deterministic. □

Lemma 3.3.25. *Given value* v, v'*, number* n_1, n'_1, n_2, n'_2*, memory block identifier* l, l'*, type* ty, ty'*, and memory* σ, σ'*, if* $\text{WriteOOB}(v, n_1, n_2, l, ty, \sigma) = (\sigma_1, j)$*,* $\text{WriteOOB}(v', n'_1, n'_2, l', ty', \sigma') = (\sigma'_1, j')$*,* $v = v'$*,* $n_1 = n'_1$*,* $n_2 = n'_2$*,* $l = l'$*,* $ty = ty'$*, and* $\sigma = \sigma'$*, then* $\sigma_1 = \sigma'_1$ *and* $j = j'$*.*

Proof. By definition of Algorithm WriteOOB, WriteOOB is deterministic. □

Lemma 3.3.26. *Given statements* s_1, s_2*,* s'_1, s'_2*, if* $\text{ExtractVariables}(s_1,\ s_2) = x_{mod}$*,* $\text{ExtractVariables}(s'_1, s'_2) = x'_{mod}$*,* $s_1 = s'_1$ *and* $s_2 = s'_2$*, then* $x_{mod} = x'_{mod}$*.*

Proof. By definition of Algorithm ExtractVariables, ExtractVariables is deterministic. □

Lemma 3.3.27. *Given variable list* x_{list}, x''_{list}*, environment* γ, γ''*, memory* σ, σ''*, accumulator* acc, acc''*, if* $\text{InitializeVariables}(x_{list}, \gamma, \sigma, \text{acc}) = (\gamma', \sigma')$*,* $\text{InitializeVariables}(x''_{list}, \gamma'', \sigma'', \text{acc}'') = (\gamma''', \sigma''')$*,* $x_{list} = x''_{list}$*,* $\gamma = \gamma''$*,* $\sigma = \sigma''$*, and* $\text{acc} = \text{acc}''$*, then* $\gamma' = \gamma'''$ *and* $\sigma' = \sigma'''$*.*

Proof. By definition of Algorithm InitializeVariables, InitializeVariables is deterministic. □

Lemma 3.3.28. *Given variable list* x_{list}, x'_{list}*, environment* γ, γ'*, memory* σ, σ'*, and accumulator* acc, acc'*, if* $\text{RestoreVariables}(x_{list}, \gamma, \sigma, \text{acc}) = \sigma_2$*,* $\text{RestoreVariables}(x'_{list}, \gamma', \sigma', \text{acc}') = \sigma'_2$*,* $x_{list} = x'_{list}$*,* $\gamma = \gamma'$*,* $\sigma = \sigma'$*, and* $\text{acc} = \text{acc}'$*, then* $\sigma_2 = \sigma'_2$*.*

Proof. By definition of Algorithm RestoreVariables, RestoreVariables is deterministic. □

Lemma 3.3.29. *Given variable list* x_{list}, x'_{list}*, environment* γ, γ'*, memory* σ, σ'*, and accumulator* acc, acc'*, if* $\text{ResolveVariables}(x_{list}, \gamma, \sigma, \text{acc}, res_{\text{acc}}) = \sigma_1$*,* $\text{ResolveVariables}(x'_{list}, \gamma', \sigma', \text{acc}', res'_{\text{acc}}) = \sigma_1$*,* $x_{list} = x'_{list}$*,* $\gamma = \gamma'$*,* $\sigma = \sigma'$*,* $\text{acc} = \text{acc}'$*, and* $res_{\text{acc}} = res'_{\text{acc}}$*, then* $\sigma_1 = \sigma'_1$*.*

Proof. By definition of Algorithm ResolveVariables, ResolveVariables is deterministic. □

3.3.2 Proof of Noninterference

Theorem 3.3.2 (Noninterference over evaluation trees)**.** *For every environment* γ, γ', γ''*; memory* σ, σ', $\sigma'' \in$ Mem*; accumulator* acc, acc$'$, acc$'' \in \mathbb{N}$*; statement* s*, values* v', v''*; and step evaluation codes* $[c'_1, ..., c'_n], [c''_1, ..., c''_n]$*; if* $\Pi \triangleright (\gamma, \sigma, \text{acc}, s) \Downarrow^{[\bar{l}'_1,...,\bar{l}'_n]}_{[c'_1,...,c'_n]} (\gamma', \sigma', \text{acc}', v')$ *and* $\Sigma \triangleright (\gamma, \sigma, \text{acc}, s) \Downarrow_{[c''_1,...,c''_n]} (\gamma'', \sigma'', \text{acc}'', v'')$*, then* $\gamma' = \gamma''$*,* $\sigma' = \sigma''$*,* $\text{acc}' = \text{acc}''$*,* $v' = v''$*,* $[c'_1, ..., c'_n] \simeq_L [c''_1, ..., c''_n]$*, and* $\Pi \simeq_L \Sigma$*.*

Proof.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma_2, \text{acc}, n_3)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma_2, \text{acc}, n_3)$ by rule Private Less Than True, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n_1)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n_2)$, $n_1 <_{\text{private}} n_2$, and $\text{encrypt}(1) = n_3$.

By definition 3.3.1, given $c = ltt1$, we have $c \simeq_L c'$ if $c' = ltt1 \vee ltf1$. Therefore, we have the following two subcases:

Subcase $\Sigma \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma'_2, \text{acc}, n'_3)$

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma, \sigma'_2, \text{acc}, n'_3)$ by rule Private Less Than True, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n'_1)$, $(\gamma, \sigma'_1, \text{acc}, e'_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n'_2)$, $n'_1 <_{\text{private}} n'_2$, and $\text{encrypt}(1) = n'_3$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n_1)$ and $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n'_1)$, by the inductive hypothesis we have that $\sigma_1 = \sigma'_1$, $n_1 = n'_1$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n_2)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n'_2)$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have that $\sigma_2 = \sigma'_2$, $n_2 = n'_2$, and $c_2 \simeq_L c'_2$.

Given $\text{encrypt}(1) = n_3$ and $\text{encrypt}(1) = n'_3$, we have $n_3 = n'_3 = \text{encrypt}(1)$.

Therefore we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\text{acc} = \text{acc}$, and $n_3 = n'_3$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Subcase $\Sigma \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltf1} (\gamma, \sigma_2, \text{acc}, n_3)$

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltf1} (\gamma, \sigma_2, \text{acc}, n_3)$ by rule Private Less Than False, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_e (\gamma, \sigma'_1, \text{acc}, n'_1)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_e (\gamma, \sigma'_2, \text{acc}, n'_2)$, $n'_1 >=_{\text{private}} n'_2$, and $\text{encrypt}(0) = n'_3$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma,\ \sigma_1, \text{acc}, n_1)$ and $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma,\ \sigma'_1, \text{acc}, n'_1)$, by the inductive hypothesis we have that $\sigma_1 = \sigma'_1$, $n_1 = n'_1$, and $c_1 \simeq_L c'_1$.

Given $(\gamma,\ \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma,\ \sigma_2, \text{acc}, n_2)$, $(\gamma,\ \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma,\ \sigma'_2, \text{acc}, n'_2)$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have that $\sigma_2 = \sigma'_2$, $n_2 = n'_2$, and $c_2 \simeq_L c'_2$.

Given $\text{encrypt}(1) = n_3$ and $\text{encrypt}(0) = n'_3$, by Axiom 3.3.1 we have $n_3 = n'_3$.

Therefore we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\text{acc} = \text{acc}$, and $n_3 = n'_3$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{\textit{ltt2}} (\gamma,\ \sigma_2, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{\textit{ltt1}} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{\textit{ltt3}} (\gamma,\ \sigma_2, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{\textit{ltt1}} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{\textit{ltf1}} (\gamma,\ \sigma_2, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{\textit{ltt1}} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{\textit{ltf2}} (\gamma,\ \sigma_2, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{\textit{ltt1}} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{\textit{ltf3}} (\gamma,\ \sigma_2, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{\textit{ltt1}} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{\textit{eqt1}} (\gamma,\ \sigma_2, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{\textit{ltt1}} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{\textit{eqt2}} (\gamma,\ \sigma_2, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{\textit{ltt1}} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{\textit{eqt3}} (\gamma,\ \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqf1} (\gamma,\ \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqf2} (\gamma,\ \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqf3} (\gamma,\ \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{net1} (\gamma,\ \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{net2} (\gamma,\ \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{net3} (\gamma,\ \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{nef1} (\gamma,\ \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{nef2} (\gamma,\ \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{nef3} (\gamma,\ \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt1} (\gamma,\ \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$ by rule Public Less Than True, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n_1)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n_2)$, and $n_1 <_{\text{public}} n_2$.

By definition 3.3.1, given $c = ltt$, we have $c \simeq_L c'$ if and only if $c' = ltt$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma'_2, \text{acc}, 1)$ by rule Public Less Than True, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n'_1)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n'_2)$, and $n'_1 <_{\text{public}} n'_2$

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n_1)$ and $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n'_1)$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n_1 = n'_1$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n_2)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n'_2)$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $n_2 = n'_2$, and $c_2 \simeq_L c'_2$.

Therefore we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\text{acc} = \text{acc}$, and $1 = 1$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltf} (\gamma, \sigma_2, \text{acc}, 0)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqt} (\gamma, \sigma_2, \text{acc}, 1)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 == e_2) \Downarrow_{eqf} (\gamma, \sigma_2, \text{acc}, 0)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{net} (\gamma, \sigma_2, \text{acc}, 1)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 != e_2) \Downarrow_{nef} (\gamma, \sigma_2, \text{acc}, 0)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 < e_2) \Downarrow_{ltt} (\gamma, \sigma_2, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp} (\gamma, \sigma_2, \text{acc}, n_3)$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp} (\gamma, \sigma_2, \text{acc}, n_3)$ by rule Public Addition, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma)$

$= \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n_1)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n_2)$, and $n_1 +_{\text{public}} n_2 = n_3$.

By definition 3.3.1, given $c = bp$, we have $c \simeq_L c'$ if and only if $c' = bp$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp} (\gamma, \sigma'_2, \text{acc}, n'_3)$ by rule Public Addition, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma)$ $= \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n_1)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n'_2)$, and $n'_1 +_{\text{public}} n'_2 = n'_3$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n_1)$ and $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n'_1)$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n_1 = n'_1$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n_2)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n'_2)$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $n_2 = n'_2$, and $c_2 \simeq_L c'_2$.

Given $n_1 = n'_1$, $n_2 = n'_2$, $n_1 +_{\text{public}} n_2 = n_3$, and $n'_1 +_{\text{public}} n'_2 = n'_3$, we have $n_3 = n'_3$.

Therefore we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\text{acc} = \text{acc}$, and $n_3 = n'_3$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 - e_2) \Downarrow_{bs} (\gamma, \sigma_2, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 \cdot e_2) \Downarrow_{bm} (\gamma, \sigma_2, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp1} (\gamma, \sigma_2, \text{acc}, n_3)$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp1} (\gamma, \sigma_2, \text{acc}, n_3)$ by rule Private Addition, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma)$ $= \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n_1)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n_2)$, and $n_1 +_{\text{private}} n_2 = n_3$.

By definition 3.3.1, given $c = bp1$, we have $c \simeq_L c'$ if and only if $c' = bp1$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp1} (\gamma, \sigma'_2, \text{acc}, n'_3)$ by rule Private Addition, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma)$ $= \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n'_1)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n'_2)$, and $n'_1 +_{\text{private}} n'_2 = n'_3$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n_1)$ and $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n'_1)$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n_1 = n'_1$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n_2)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n'_2)$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $n_2 = n'_2$, and $c_2 \simeq_L c'_2$.

Given $n_1 = n'_1$, $n_2 = n'_2$, $n_1 +_{\text{private}} n_2 = n_3$, and $n'_1 +_{\text{private}} n'_2 = n'_3$, by Axiom 3.3.2 we have $n_3 = n'_3$.

Therefore we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\text{acc} = \text{acc}$, and $n_3 = n'_3$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 - e_2) \Downarrow_{bs1} (\gamma, \sigma_2, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp1} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 \cdot e_2) \Downarrow_{bm1} (\gamma, \sigma_2, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp1} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$ by rule Public-Private Addition, we have $\text{Label}(e_1, \gamma) = \text{public}$, $\text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n_1)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n_2)$, and $\text{encrypt}(n_1) +_{\text{private}} n_2 = n_3$.

By definition 3.3.1, given $c = bp2$, we have $c \simeq_L c'$ if and only if $c' = bp2$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma'_2, \text{acc}, n'_3)$ by rule Public-Private Addition, we have $\text{Label}(e_1, \gamma) = \text{public}$, $\text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n'_1)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n'_2)$, and $\text{encrypt}(n'_1) +_{\text{private}} n'_2 = n'_3$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n_1)$ and $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n'_1)$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n_1 = n'_1$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n_2)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n'_2)$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $n_2 = n'_2$, and $c_2 \simeq_L c'_2$.

Given $n_1 = n_1'$, $\text{encrypt}(n_1)$, and $\text{encrypt}(n_1')$, by Axiom 3.3.1 we have that $\text{encrypt}(n_1) = \text{encrypt}(n_1')$.

Given $\text{encrypt}(n_1) = \text{encrypt}(n_1')$, $n_2 = n_2'$, $\text{encrypt}(n_1) +_{\text{private}} n_2 = n_3$, and $\text{encrypt}(n_1') +_{\text{private}} n_2' = n_3'$, by Axiom 3.3.2 we have $n_3 = n_3'$.

Therefore we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, and $n_3 = n_3'$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp3} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 - e_2) \Downarrow_{bs2} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 - e_2) \Downarrow_{bs3} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 \cdot e_2) \Downarrow_{bm2} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 \cdot e_2) \Downarrow_{bm3} (\gamma, \sigma_2, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, e_1 + e_2) \Downarrow_{bp2} (\gamma, \sigma_2, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iep} (\gamma,\ \sigma_7, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iep} (\gamma,\ \sigma_7,\ \text{acc},\ \text{skip})$ by rule Private If Else, we have $\text{Label}(e,\ \gamma) = \text{private}$, $(\gamma,\ \sigma,\ \text{acc},\ e) \Downarrow_{c_1} (\gamma,\ \sigma_1,\ \text{acc},\ n)$, $(\gamma,\ \sigma_1,\ \text{acc},\ \text{private int } res_{\text{acc}+1} = n) \Downarrow_{c_2} (\gamma_1,\ \sigma_2,\ \text{acc},\ \text{skip})$, $\text{Extract_variables}(s_1,\ s_2) = x_{list_\text{acc}+1}$, $\text{InitializeVariables}(x_{list_\text{acc}+1},\ \gamma_1,\ \sigma_2,\ \text{acc}+1) = (\gamma_2,\ \sigma_3)$, $(\gamma_2,\ \sigma_3,\ \text{acc}+1,\ s_1) \Downarrow_{c_3} (\gamma_3,\ \sigma_4,\ \text{acc}+1,\ \text{skip})$, $\text{RestoreVariables}(x_{list_\text{acc}+1},\ \gamma_3,\ \sigma_4,\ \text{acc}+1) = \sigma_5$, $(\gamma_3,\ \sigma_5,\ \text{acc}+1,\ s_2) \Downarrow_{c_4} (\gamma_4,\ \sigma_6,\ \text{acc}+1,\ \text{skip})$, and $\text{ResolveVariables}(x_{list_\text{acc}+1}, \gamma_4, \sigma_6,\ \text{acc}+1,\ res_{\text{acc}+1}) = \sigma_7$.

By definition 3.3.1, given $c = iep$, we have $c \simeq_L c'$ if and only if $c' = iep$.

Given $\Sigma \triangleright (\gamma,\ \sigma,\ \text{acc},\ \text{if}\ (e)\ s_1\ \text{else}\ s_2) \Downarrow_{iep} (\gamma,\ \sigma'_7,\ \text{acc},\ \text{skip})$ by rule Private If Else, we have $\text{Label}(e,\ \gamma) = \text{private}$, $(\gamma,\ \sigma,\ \text{acc},\ e) \Downarrow_{c'_1} (\gamma,\ \sigma'_1,\ \text{acc},\ n')$, $(\gamma,\ \sigma'_1,\ \text{acc},\ \text{private int}\ res_{\text{acc}+1} =\ n') \Downarrow_{c'_2} (\gamma'_1,\ \sigma'_2,\ \text{acc},\ \text{skip})$, $\text{Extract_variables}(s_1,\ s_2) = x'_{list_\text{acc}+1}$, $\text{InitializeVariables}(x'_{list_\text{acc}+1},\ \gamma'_1,\ \sigma'_2,\ \text{acc}+1) = (\gamma_2,\ \sigma_3)$, $(\gamma'_2,\ \sigma'_3,\ \text{acc}+1,\ s_1) \Downarrow_{c'_3} (\gamma'_3,\ \sigma'_4, \text{acc}+1,\ \text{skip})$, $\text{RestoreVariables}(x'_{list_\text{acc}+1}, \gamma'_3, \sigma'_4,\ \text{acc}+1) = \sigma'_5$, $(\gamma'_3,\ \sigma'_5,\ \text{acc}+1,\ s_2) \Downarrow_{c'_4} (\gamma'_4,\ \sigma'_6,\ \text{acc}+1,\ \text{skip})$, and $\text{ResolveVariables}(x'_{list_\text{acc}+1}, \gamma'_4,\ \sigma'_6,\ \text{acc}+1,\ res_{\text{acc}+1}) = \sigma'_7$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma,\ \sigma_1, \text{acc}, n)$, and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma,\ \sigma'_1, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma,\ \sigma_1, \text{acc}, \text{private int}\ res_{\text{acc}+1} = n) \Downarrow_{c_2} (\gamma_1,\ \sigma_2, \text{acc}, \text{skip})$, $(\gamma,\ \sigma'_1, \text{acc}, \text{private int}\ res_{\text{acc}+1} = n') \Downarrow_{c'_2} (\gamma'_1,\ \sigma'_2, \text{acc}, \text{skip})$, $\sigma_1 = \sigma'_1$, and $n = n'$, we have $\{\text{private int}\ res_{\text{acc}+1} =\ n\} = \{\text{private int}\ res_{\text{acc}+1} =\ n'\}$. By the inductive hypothesis, we have $\gamma_1 = \gamma'_1$, $\sigma_2 = \sigma'_2$, and $c_2 \simeq_L c'_2$.

Given $\text{Extract_variables}(s_1,\ s_2) = x_{list_\text{acc}+1}$ and $\text{Extract_variables}(s_1,\ s_2) = x'_{list_\text{acc}+1}$, by Lemma 3.3.26 we have $x_{list_\text{acc}+1} = x'_{list_\text{acc}+1}$.

Given $\text{InitializeVariables}(x_{list_\text{acc}+1},\ \gamma_1, \sigma_2, \text{acc}+1) = (\gamma_2,\ \sigma_3)$, $\text{InitializeVariables}(x'_{list_\text{acc}+1}, \gamma'_1, \sigma'_2, \text{acc}+1) = (\gamma'_2,\ \sigma'_3)$, $x_{list_\text{acc}+1} = x'_{list_\text{acc}+1}$, $\gamma_1 = \gamma'_1$, and $\sigma_2 = \sigma'_2$, by Lemma 3.3.27 we have $\gamma_2 = \gamma'_2$ and $\sigma_3 = \sigma'_3$.

Given $(\gamma_2,\ \sigma_3,\ \text{acc}+1,\ s_1) \Downarrow_{c_3} (\gamma_3,\ \sigma_4,\ \text{acc}+1,\ \text{skip})$, $(\gamma'_2,\ \sigma'_3,\ \text{acc}+1,\ s_1) \Downarrow_{c'_3} (\gamma'_3,\ \sigma'_4,\ \text{acc}+1,\ \text{skip})$, $\gamma_2 = \gamma'_2$, and $\sigma_3 = \sigma'_3$, by the inductive hypothesis we have $\gamma_3 = \gamma'_3$, $\sigma_4 = \sigma'_4$, and $c_3 \simeq_L c'_3$.

Given $\text{RestoreVariables}(x_{list_\text{acc}+1},\ \gamma_3,\ \sigma_4,\ \text{acc}+1) = \sigma_5$, $\text{RestoreVariables}(x'_{list_\text{acc}+1},\ \gamma'_3, \sigma'_4, \text{acc}+1) = \sigma'_5$, $x_{list_\text{acc}+1} = x'_{list_\text{acc}+1}$, $\gamma_3 = \gamma'_3$, and $\sigma_4 = \sigma'_4$, by Lemma 3.3.28 we have $\sigma_5 = \sigma'_5$.

Given $(\gamma_3,\ \sigma_5,\ \text{acc}+1,\ s_2) \Downarrow_{c_4} (\gamma_4,\ \sigma_6,\ \text{acc}+1,\ \text{skip})$, $(\gamma'_3,\ \sigma'_5,\ \text{acc}+1,\ s_2) \Downarrow_{c'_4} (\gamma'_4,\ \sigma'_6,\ \text{acc}+1,\ \text{skip})$, $\gamma_3 = \gamma'_3$, and $\sigma_5 = \sigma'_5$, by the inductive hypothesis we have $\gamma_4 = \gamma'_4$, $\sigma_6 = \sigma'_6$, and $c_4 \simeq_L c'_4$.

Given $\text{ResolveVariables}(x_{list_\text{acc}+1},\ \gamma_4,\ \sigma_6,\ \text{acc}+1,\ res_{\text{acc}+1}) = \sigma_7$, $\text{ResolveVariables}(x'_{list_\text{acc}+1}, \gamma'_4, \sigma'_6, \text{acc}+1,\ res_{\text{acc}+1}) = \sigma'_7$, $x_{list_\text{acc}+1} = x'_{list_\text{acc}+1}$, $\gamma_4 = \gamma'_4$, and $\sigma_6 = \sigma'_6$, by Lemma 3.3.29 we have $\sigma_7 = \sigma'_7$.

Therefore, we have $\gamma = \gamma$, $\sigma_7 = \sigma'_7$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iet} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iet} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule If Else True, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$, $n \neq 0$, and $(\gamma, \sigma_1, \text{acc}, s_1) \Downarrow_{c_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$.

By definition 3.3.1, given $c = iet$, we have $c \simeq_L c'$ if and only if $c' = iet$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iet} (\gamma, \sigma_2', \text{acc}, \text{skip})$ by rule If Else True, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, n')$, $n' \neq 0$, and $(\gamma, \sigma_1', \text{acc}, s_1) \Downarrow_{c_2'} (\gamma_1', \sigma_2', \text{acc}, \text{skip})$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $n = n'$, and $c_1 \simeq_L c_1'$.

Given $(\gamma, \sigma_1, \text{acc}, s_1) \Downarrow_{c_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$, $(\gamma, \sigma_1', \text{acc}, s_1) \Downarrow_{c_2'} (\gamma_1', \sigma_2', \text{acc}, \text{skip})$, and $\sigma_1 = \sigma_1'$, by the inductive hypothesis we have $\gamma_1 = \gamma_1'$, $\sigma_2 = \sigma_2'$, and $c_2 \simeq_L c_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\text{acc} = \text{acc}$, and $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{ief} (\gamma, \sigma_2, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iet} (\gamma, \sigma_2, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \&x) \Downarrow_{loc} (\gamma, \sigma, \text{acc}, (l, 0))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \&x) \Downarrow_{loc} (\gamma, \sigma, \text{acc}, (l, 0))$ by rule Address Of, we have $\gamma(x) = (l,\ ty)$.

By definition 3.3.1, given $c = loc$, we have $c \simeq_L c'$ if and only if $c' = loc$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \&x) \Downarrow_{loc} (\gamma, \sigma, \text{acc}, (l', 0))$ by rule Address Of, we have $\gamma(x) = (l',\ ty')$.

Given $\gamma = \gamma$, we have that $l = l'$ and $ty = ty'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\text{acc} = \text{acc}$, and $(l, 0) = (l', 0)$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{sizeof}(ty)) \Downarrow_{ty} (\gamma, \sigma, \text{acc}, n)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{sizeof}(ty)) \Downarrow_{ty} (\gamma, \sigma, \text{acc}, n)$ by rule Size of Type, we have $n = \tau(ty)$.

By definition 3.3.1, given $c = ty$, we have $c \simeq_L c'$ if and only if $c' = ty$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \text{sizeof}(ty)) \Downarrow_{ty} (\gamma, \sigma, \text{acc}, n')$ by rule Size of Type, we have $n' = \tau(ty)$.

Given $n = \tau(ty)$ and $n' = \tau(ty)$, by definition of τ we have $n = n'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\text{acc} = \text{acc}$, and $n = n'$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow_{wle} (\gamma, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow_{wle} (\gamma, \sigma_1, \text{acc}, \text{skip})$ by rule While End, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$, $\text{Label}(e, \gamma) = \text{public}$, and $n = 0$.

By definition 3.3.1, given $c = wle$, we have $c \simeq_L c'$ if and only if $c' = wle$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow_{wle} (\gamma, \sigma'_1, \text{acc}, \text{skip})$ by rule While End, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow'_{c_1} (\gamma, \sigma'_1, \text{acc}, n')$, $\text{Label}(e, \gamma) = \text{public}$, and $n' = 0$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow'_{c_1} (\gamma, \sigma'_1, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow_{wlc} (\gamma, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow_{wlc} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by rule While Continue, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$, $n \neq 0$, $(\gamma, \sigma_1, \text{acc}, s) \Downarrow_{c_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$, and $(\gamma_1, \sigma_2, \text{acc}, \text{while } (e)\ s) \Downarrow_{c_3} (\gamma_2, \sigma_3,$

acc, skip).

By definition 3.3.1, given $c = wlc$, we have $c \simeq_L c'$ if and only if $c' = wlc$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow_{wlc} (\gamma, \sigma'_3, \text{acc}, \text{skip})$ by rule While Continue, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n')$, $n' \neq 0$, $(\gamma, \sigma'_1, \text{acc}, s) \Downarrow_{c'_2} (\gamma'_1, \sigma'_2, \text{acc}, \text{skip})$, and $(\gamma'_1, \sigma'_2, \text{acc}, \text{while } (e)\ s) \Downarrow_{c'_3} (\gamma'_2, \sigma'_3, \text{acc}, \text{skip})$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, s) \Downarrow_{c_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$, $(\gamma, \sigma'_1, \text{acc}, s) \Downarrow_{c'_2} (\gamma'_1, \sigma'_2, \text{acc}, \text{skip})$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\gamma_1 = \gamma'_1$, $\sigma_2 = \sigma'_2$, and $c_2 \simeq_L c'_2$.

Given $(\gamma_1, \sigma_2, \text{acc}, \text{while } (e)\ s) \Downarrow_{c_3} (\gamma_2, \sigma_3, \text{acc}, \text{skip})$, $(\gamma'_1, \sigma'_2, \text{acc}, \text{while } (e)\ s) \Downarrow_{c'_3} (\gamma'_2, \sigma'_3, \text{acc}, \text{skip})$, $\gamma_1 = \gamma'_1$, and $\sigma_2 = \sigma'_2$, by the inductive hypothesis we have $\gamma_2 = \gamma'_2$, $\sigma_3 = \sigma'_3$, and $c_3 \simeq_L c'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, s_1; s_2) \Downarrow_{ss} (\gamma_2, \sigma_2, \text{acc}, v)$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, s_1; s_2) \Downarrow_{ss} (\gamma_2, \sigma_2, \text{acc}, v)$ by rule Statement Sequencing, we have $(\gamma, \sigma, \text{acc}, s_1) \Downarrow_{c_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ and $(\gamma_1, \sigma_1, \text{acc}, s_2) \Downarrow_{c_2} (\gamma_2, \sigma_2, \text{acc}, v)$.

By definition 3.3.1, given $c = ss$, we have $c \simeq_L c'$ if and only if $c' = ss$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, s_1; s_2) \Downarrow_{ss} (\gamma'_2, \sigma'_2, \text{acc}, v')$ by rule Statement Sequencing, we have $(\gamma, \sigma, \text{acc}, s_1) \Downarrow_{c'_1} (\gamma'_1, \sigma'_1, \text{acc}, \text{skip})$ and $(\gamma'_1, \sigma'_1, \text{acc}, s_2) \Downarrow_{c'_2} (\gamma'_2, \sigma'_2, \text{acc}, v)$.

Given $(\gamma, \sigma, \text{acc}, s_1) \Downarrow_{c_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ and $(\gamma, \sigma, \text{acc}, s_1) \Downarrow_{c'_1} (\gamma'_1, \sigma'_1, \text{acc}, \text{skip})$, by the inductive hypothesis we have $\gamma_1 = \gamma'_1$, $\sigma_1 = \sigma'_1$, and $c_1 \simeq_L c'_1$.

Given $(\gamma_1, \sigma_1, \text{acc}, s_2) \Downarrow_{c_2} (\gamma_2, \sigma_2, \text{acc}, v)$, $(\gamma'_1, \sigma'_1, \text{acc}, s_2) \Downarrow_{c'_2} (\gamma'_2, \sigma'_2, \text{acc}, v)$, $\gamma_1 = \gamma'_1$, and $\sigma_1 = \sigma'_1$, by the

inductive hypothesis we have $\gamma_2 = \gamma_2'$, $\sigma_2 = \sigma_2'$, $v = v'$, and $c_2 \simeq_L c_2'$.

Therefore, we have $\gamma_2 = \gamma_2'$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, $v = v'$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, (e)) \Downarrow_{ep} (\gamma, \sigma_1, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, (e)) \Downarrow_{ep} (\gamma, \sigma_1, \text{acc}, v)$ by rule Parentheses, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, v)$.

By definition 3.3.1, given $c = ep$, we have $c \simeq_L c'$ if and only if $c' = ep$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, (e)) \Downarrow_{ep} (\gamma, \sigma_1, \text{acc}, v')$ by rule Parentheses, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, v')$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, v)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, v')$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $v = v'$, and $c_1 \simeq_L c_1'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma_1'$, $\text{acc} = \text{acc}$, $v = v'$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \{s\}) \Downarrow_{sb} (\gamma, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \{s\}) \Downarrow_{sb} (\gamma, \sigma_1, \text{acc}, \text{skip})$ by rule Statement Block, we have $(\gamma, \sigma, \text{acc}, s) \Downarrow_{c_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$.

By definition 3.3.1, given $c = sb$, we have $c \simeq_L c'$ if and only if $c' = sb$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \{s\}) \Downarrow_{sb} (\gamma, \sigma_1', \text{acc}, \text{skip})$ by rule Statement Block, we have $(\gamma, \sigma, \text{acc}, s) \Downarrow_{c_1'} (\gamma_1', \sigma_1', \text{acc}, \text{skip})$.

Given $(\gamma, \sigma, \text{acc}, s) \Downarrow_{c_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ and $(\gamma, \sigma, \text{acc}, s) \Downarrow_{c_1'} (\gamma_1', \sigma_1', \text{acc}, \text{skip})$, by the inductive hypothesis we have $\gamma_1 = \gamma_1'$, $\sigma_1 = \sigma_1'$, and $c_1 \simeq_L c_1'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma_1'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl} (\gamma, \sigma_3, \text{acc}, (l, 0))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl} (\gamma, \sigma_3, \text{acc}, (l, 0))$ by rule Cast Public Location, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, (l, 0))$, $(ty = \text{public } bty*) \vee (ty = \text{char}*)$, $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega, \text{void}, n, \text{PermL}(\text{Freeable, void, public}, n)\big)\big]$, and $\sigma_3 = \sigma_2\Big[l \rightarrow \Big(\omega, ty, \frac{n}{\tau(ty)}, \text{PermL}(\text{Freeable}, ty, \text{public}, \frac{n}{\tau(ty)}\Big)\Big)\Big]$.

By definition 3.3.1, given $c = cl$, we have $c \simeq_L c'$ if and only if $c' = cl$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl} (\gamma, \sigma'_3, \text{acc}, (l', 0))$ by rule Cast Public Location, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, (l', 0))$, $(ty = \text{public } bty'*) \vee (ty = \text{char}*)$, $\sigma'_1 = \sigma'_2\big[l' \rightarrow \big(\omega', \text{void}, n', \text{PermL}(\text{Freeable, void, public}, n')\big)\big]$, and $\sigma'_3 = \sigma'_2\Big[l' \rightarrow \Big(\omega', ty, \frac{n'}{\tau(ty)}, \text{PermL}(\text{Freeable}, ty, \text{public}, \frac{n'}{\tau(ty)}\Big)\Big)\Big]$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, (l, 0))$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, (l', 0))$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $l = l'$, and $c_1 \simeq_L c'_1$.

Given $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega, \text{void}, n, \text{PermL}(\text{Freeable, void, public}, n)\big)\big]$, $\sigma'_1 = \sigma'_2\big[l' \rightarrow \big(\omega', \text{void}, n', \text{PermL}(\text{Freeable, void, public}, n')\big)\big]$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\sigma_2 = \sigma'_2$, $\omega = \omega'$, and $n = n'$.

Given $\sigma_3 = \sigma_2\Big[l \rightarrow \big(\omega, ty, \frac{n}{\tau(ty)}, \text{PermL}(\text{Freeable}, ty, \text{public}, \frac{n}{\tau(ty)}\Big)\Big)\Big]$, $\sigma'_3 = \sigma'_2\big[l' \rightarrow \big(\omega', ty, \frac{n'}{\tau(ty)}, \text{PermL}(\text{Freeable}, ty, \text{public}, \frac{n'}{\tau(ty)})\big)\big]$, $\sigma_2 = \sigma'_2$, $l = l'$, $\omega = \omega'$, and $n = n'$, we have $\sigma_3 = \sigma'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\text{acc} = \text{acc}$, $(l, 0) = (l', 0)$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl1} (\gamma, \sigma_3, \text{acc}, (l, 0))$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cl} (\gamma, \sigma_3, \text{acc}, (l, 0))$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv} (\gamma, \sigma_1, \text{acc}, n_1)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv} (\gamma, \sigma_1, \text{acc}, n_1)$ by rule Cast Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$, $(ty = \text{public int}) \vee (ty = \text{public float})$, and $n_1 = \text{Cast}(\text{public}, ty, n)$.

By definition 3.3.1, given $c = cv$, we have $c \simeq_L c'$ if and only if $c' = cv$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv} (\gamma, \sigma_1', \text{acc}, n_1')$ by rule Cast Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, n')$, $(ty = \text{public int}) \vee (ty = \text{public float})$, and $n_1' = \text{Cast}(\text{public}, ty, n')$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $n = n'$, and $c_1 \simeq_L c_1'$.

Given $n_1 = \text{Cast}(\text{public}, ty, n)$, $n_1' = \text{Cast}(\text{public}, ty, n')$, and $n = n'$, by definition of Cast, we have $n_1 = n_1'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma_1'$, $\text{acc} = \text{acc}$, $n_1 = n_1'$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv1} (\gamma, \sigma_1, \text{acc}, n_1)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, (ty)\ e) \Downarrow_{cv} (\gamma, \sigma_1, \text{acc}, n_1)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow_{inp} (\gamma, \sigma_3, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow_{inp} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by rule SMC Input Public Value, we have $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, x)$, $\text{acc} = 0$, $(\gamma\ \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n)$, $\gamma(x) = (l, \text{public}\ bty)$, $\text{InputValue}(x, n) = n_1$, and $(\gamma, \sigma_2, \text{acc}, x = n_1) \Downarrow_{c_3} (\gamma, \sigma_3, \text{acc}, \text{skip})$.

By definition 3.3.1, given $c = inp$, we have $c \simeq_L c'$ if and only if $c' = inp$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow_{inp} (\gamma, \sigma_3', \text{acc}, \text{skip})$ by rule SMC Input Public Value, we have $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, x')$, $\text{acc} = 0$, $(\gamma\ \sigma_1', \text{acc}, e_2) \Downarrow_{c_2'} (\gamma, \sigma_2', \text{acc}, n')$, $\gamma(x') = (l', \text{public}\ bty')$, $\text{InputValue}(x', n') = n_1'$, and $(\gamma, \sigma_2', \text{acc}, x' = n_1') \Downarrow_{c_3'} (\gamma, \sigma_3', \text{acc}, \text{skip})$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, x)$ and $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, x')$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $x = x'$, and $c_1 \simeq_L c_1'$.

Given $(\gamma\ \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n)$, $(\gamma\ \sigma_1', \text{acc}, e_2) \Downarrow_{c_2'} (\gamma, \sigma_2', \text{acc}, n')$, and $\sigma_1 = \sigma_1'$, by the inductive hypothesis we have $\sigma_2 = \sigma_2'$, $n = n'$, and $c_2 \simeq_L c_2'$.

Given $\gamma(x) = (l, \text{public}\ bty)$, $\gamma(x') = (l', \text{public}\ bty')$, and $x = x'$, we have $l = l'$ and $bty = bty'$.

Given $\text{InputValue}(x, n) = n_1$, $\text{InputValue}(x', n') = n'_1$, $x = x'$, and $n = n'$, by Axiom 3.3.3 we have $n_1 = n'_1$.

Given $(\gamma, \sigma_2, \text{acc}, x = n_1) \Downarrow_{c_3} (\gamma, \sigma_3, \text{acc}, \text{skip})$, $(\gamma, \sigma'_2, \text{acc}, x' = n'_1) \Downarrow_{c'_3} (\gamma, \sigma'_3, \text{acc}, \text{skip})$, $\sigma_2 = \sigma'_2$, $x = x'$, and $n = n'$, by the inductive hypothesis we have $\sigma_3 = \sigma'_3$ and $c_3 \simeq_L c'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow_{inp3} (\gamma, \sigma_3, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow_{inp} (\gamma, \sigma_3, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp1} (\gamma, \sigma_4, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp1} (\gamma, \sigma_4, \text{acc}, \text{skip})$ by rule SMC Input Public 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, x')$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n)$, $(\gamma, \sigma_2, \text{acc}, e_3) \Downarrow_{c_3} (\gamma, \sigma_3, \text{acc}, n_1)$, $\gamma(x) = (l, \text{public const } bty*)$, $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, and $(\gamma, \sigma_3, \text{acc}, x = [m_0, ..., m_{n_1}]) \Downarrow_{c_4} (\gamma, \sigma_4, \text{acc}, \text{skip})$.

By definition 3.3.1, given $c = \mathit{inp1}$, we have $c \simeq_L c'$ if and only if $c' = \mathit{inp1}$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp1} (\gamma, \sigma'_4, \text{acc}, \text{skip})$ by rule SMC Input Public 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, x)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n')$, $(\gamma, \sigma'_2, \text{acc}, e_3) \Downarrow_{c'_3} (\gamma, \sigma'_3, \text{acc}, n'_1)$, $\gamma(x') = (l', \text{public const } bty'*)$, $\text{InputArray}(x', n', n'_1) = [m'_0, ..., m'_{n'_1}]$, and $(\gamma, \sigma'_3, \text{acc}, x' = [m'_0, ..., m'_{n'_1}]) \Downarrow_{c'_4} (\gamma, \sigma'_4, \text{acc}, \text{skip})$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, x)$ and $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, x')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $x = x'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n')$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $n = n'$, and $c_2 \simeq_L c'_2$.

Given $(\gamma, \sigma_2, \text{acc}, e_3) \Downarrow_{c_3} (\gamma, \sigma_3, \text{acc}, n_1)$, $(\gamma, \sigma'_2, \text{acc}, e_3) \Downarrow_{c'_3} (\gamma, \sigma'_3, \text{acc}, n'_1)$, and $\sigma_2 = \sigma'_2$, by the inductive

hypothesis we have $\sigma_3 = \sigma'_3$, $n_1 = n'_1$, and $c_3 \simeq_L c'_3$.

Given $\gamma(x) = (l, \text{public const } bty*)$, $\gamma(x') = (l', \text{public const } bty'*)$, and $x = x'$, we have $l = l'$ and $bty = bty'$.

Given $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, $\text{InputArray}(x', n', n'_1) = [m'_0, ..., m'_{n'_1}]$, $x = x'$, $n = n'$, and $n_1 = n'_1$, by Axiom 3.3.4 we have $[m_0, ..., m_{n_1}] = [m'_0, ..., m'_{n'_1}]$. Therefore, we have $\{x = [m_0, ..., m_{n_1}]\} = \{x' = [m'_0, ..., m'_{n'_1}]\}$.

Given $(\gamma, \sigma_3, \text{acc}, x = [m_0, ..., m_{n_1}]) \Downarrow_{c_4} (\gamma, \sigma_4, \text{acc}, \text{skip})$, $(\gamma, \sigma'_3, \text{acc}, x' = [m'_0, ..., m'_{n'_1}]) \Downarrow_{c'_4} (\gamma, \sigma'_4, \text{acc}, \text{skip})$, $\{x = [m_0, ..., m_{n_1}]\} = \{x' = [m'_0, ..., m'_{n'_1}]\}$, and $\sigma_3 = \sigma'_3$, by the inductive hypothesis we have $\sigma_4 = \sigma'_4$ and $c_4 \simeq_L c'_4$.

Therefore, we have $\gamma = \gamma$, $\sigma_4 = \sigma'_4$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp4} (\gamma, \sigma_4, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow_{inp1} (\gamma, \sigma_4, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2)) \Downarrow_{out} (\gamma, \sigma_2, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2)) \Downarrow_{out} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule SMC Output Public Value, we have $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, x)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n)$, $\gamma(x) = (l, \text{public } bty)$, $\sigma_2(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1))$, $\text{DecodeVal}(\text{public } bty, 1, \omega) = n_1$, and $\text{OutputValue}(x, n, n_1)$.

By definition 3.3.1, given $c = out$, we have $c \simeq_L c'$ if and only if $c' = out$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2)) \Downarrow_{out} (\gamma, \sigma'_2, \text{acc}, \text{skip})$ by rule SMC Output Public Value, we have $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, x')$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n')$, $\gamma(x') = (l', \text{public } bty')$, $\sigma'_2(l') = (\omega', \text{public } bty', 1, \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, 1))$, $\text{DecodeVal}(\text{public } bty', 1, \omega') = n'_1$, and $\text{OutputValue}(x', n', n'_1)$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, x)$ and $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, x')$, by the inductive hypothesis we have

$\sigma_1 = \sigma_1'$, $x = x'$, and $c_1 \simeq_L c_1'$.

Given $(\gamma,\, \sigma_1,\, \text{acc},\, e_2) \Downarrow_{c_2} (\gamma,\, \sigma_2,\, \text{acc},\, n)$, $(\gamma,\, \sigma_1',\, \text{acc},\, e_2) \Downarrow_{c_2'} (\gamma,\, \sigma_2',\, \text{acc},\, n')$, and $\sigma_1 = \sigma_1'$, by the inductive hypothesis we have $\sigma_2 = \sigma_2'$, $n = n'$, and $c_2 \simeq_L c_2'$.

Given $\gamma(x) = (l, \text{public } bty)$, $\gamma(x') = (l', \text{public } bty')$, and $x = x'$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega,\, \text{public } bty,\, 1,\, \text{PermL(Freeable, public } bty\text{, public, 1)})$, $\sigma_2'(l') = (\omega',\, \text{public } bty',\, 1,\, \text{PermL(Freeable, public } bty'\text{, public, 1)})$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodeVal(public } bty,\, 1,\, \omega) = n_1$, $\text{DecodeVal(public } bty',\, 1,\, \omega') = n_1'$, $\omega = \omega'$, and $bty = bty'$, by definition of DecodeVal we have $n_1 = n_1'$.

Given $\text{OutputValue}(x, n, n_1)$, $\text{OutputValue}(x', n', n_1')$, $x = x'$, $n = n'$, and $n_1 = n_1'$, by definition of OutputValue we will have identical output.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, acc = acc, skip = skip, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma,\, \sigma,\, \text{acc},\, \text{smcoutput}(e_1, e_2)) \Downarrow_{out3} (\gamma,\, \sigma_2,\, \text{acc},\, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma,\, \sigma,\, \text{acc},\, \text{smcoutput}(e_1, e_2)) \Downarrow_{out} (\gamma,\, \sigma_2,\, \text{acc},\, \text{skip})$.

Case $\Pi \triangleright (\gamma,\, \sigma,\, \text{acc},\, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow_{out1} (\gamma,\, \sigma_3,\, \text{acc},\, \text{skip})$
Given $\Pi \triangleright (\gamma,\, \sigma,\, \text{acc},\, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow_{out1} (\gamma,\, \sigma_3,\, \text{acc},\, \text{skip})$ by rule SMC Output Public 1D Array, we have $\text{Label}(e_2,\, \gamma) = \text{Label}(e_3,\, \gamma) = \text{public}$, $(\gamma,\, \sigma,\, \text{acc},\, e_1) \Downarrow_{c_1} (\gamma,\, \sigma_1,\, \text{acc},\, x)$, $(\gamma,\, \sigma_1,\, \text{acc},\, e_2) \Downarrow_{c_2} (\gamma,\, \sigma_2,\, \text{acc},\, n)$, $(\gamma,\, \sigma_2,\, \text{acc},\, e_3) \Downarrow_{c_3} (\gamma,\, \sigma_3,\, \text{acc},\, n_1)$, $\gamma(x) = (l,\, \text{public const } bty*)$, $\sigma_3(l) = (\omega,\, \text{public const } bty*,\, 1,\, \text{PermL(Freeable, public const } bty*\text{, public, 1)}$, $\text{DecodePtr(public const } bty*, 1, \omega) = [1,\, [(l_1,\, 0)],\, [1],\, 1]$, $\sigma_3(l_1) = (\omega_1, \text{public } bty,\, n_1,\, \text{PermL(Freeable, public } bty\text{, public, } n_1))$, $\text{DecodeVal(public } bty,\, n_1,\, \omega_1) = [m_0,\, ...,\, m_{n_1}]$, and $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$.

By definition 3.3.1, given $c = out1$, we have $c \simeq_L c'$ if and only if $c' = out1$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow_{out1} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by rule SMC Output Public 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, x')$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n')$, $(\gamma, \sigma'_2, \text{acc}, e_3) \Downarrow_{c'_3} (\gamma, \sigma'_3, \text{acc}, n'_1)$, $\gamma(x') = (l', \text{public const } bty'*)$, $\sigma'_3(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty'*, \text{public}, 1), \text{DecodePtr}(\text{public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1], \sigma'_3(l'_1) = (\omega'_1, \text{public } bty, n'_1, \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, n'_1))$, $\text{DecodeVal}(\text{public } bty', n'_1, \omega'_1) = [m'_0, ..., m'_{n'_1}]$, and $\text{OutputArray}(x', n', [m'_0, ..., m'_{n'_1}])$.

Given $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, x)$ and $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, x')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $x = x'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, n)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, n')$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $n = n'$, and $c_2 \simeq_L c'_2$.

Given $(\gamma, \sigma_2, \text{acc}, e_3) \Downarrow_{c_3} (\gamma, \sigma_3, \text{acc}, n_1)$, $(\gamma, \sigma'_2, \text{acc}, e_3) \Downarrow_{c'_3} (\gamma, \sigma'_3, \text{acc}, n'_1)$, and $\sigma_2 = \sigma'_2$, by the inductive hypothesis we have $\sigma_3 = \sigma'_3$, $n_1 = n'_1$, and $c_3 \simeq_L c'_3$.

Given $\gamma(x) = (l, \text{public const } bty*)$, $\gamma(x') = (l', \text{public const } bty'*)$, and $x = x'$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_3(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)$, $\sigma'_3(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty'*, \text{public}, 1)$, $\sigma_3 = \sigma'_3$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr}(\text{public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\omega = \omega'$, and $bty = bty'$, by definition of DecodePtr we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$ and therefore $l_1 = l'_1$.

Given $\sigma_3(l_1) = (\omega_1, \text{public } bty, n_1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n_1))$, $\sigma'_3(l'_1) = (\omega'_1, \text{public } bty, n'_1, \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, n'_1))$, $\sigma_3 = \sigma'_3$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n_1 = n'_1$.

Given $\text{DecodeVal}(\text{public } bty, n_1, \omega_1) = [m_0, ..., m_{n_1}]$, $\text{DecodeVal}(\text{public } bty', n'_1, \omega'_1) = [m'_0, ..., m'_{n'_1}]$, $bty = bty'$, $n_1 = n'_1$, and $\omega_1 = \omega'_1$, by definition of DecodeVal we have $[m_0, ..., m_{n_1}] = [m'_0, ..., m'_{n'_1}]$.

Given $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$, $\text{OutputArray}(x', n', [m'_0, ..., m'_{n'_1}])$, $x = x'$, $n = n'$, and $[m_0, ..., m_{n_1}] = [m'_0, ..., m'_{n'_1}]$, by definition of OutputArray we will have identical output.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow_{out4} (\gamma, \sigma_3, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow_{out1} (\gamma, \sigma_3, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x(\overline{p})) \Downarrow_{df} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \text{acc}, ty\ x(\overline{p})) \Downarrow_{df} (\gamma_1,\ \sigma_1, \text{acc},\ \text{skip})$ by rule Function Declaration, we have $\text{acc} = 0$, $\text{GetFunTypeList}(\overline{p})$ $= \overline{ty}$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, and $\sigma_1 = \sigma[l \rightarrow (\text{NULL}, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})]$.

By definition 3.3.1, given $c = df$, we have $c \simeq_L c'$ if and only if $c' = df$.

Given $\Sigma \triangleright (\gamma,\ \sigma,\ \text{acc},\ ty\ x(\overline{p})) \Downarrow_{df} (\gamma'_1,\ \sigma'_1,\ \text{acc},\ \text{skip})$ by rule Function Declaration, we have $\text{acc} = 0$, $\text{GetFunTypeList}(\overline{p}) = \overline{ty}'$, $l' = \phi()$, $\gamma'_1 = \gamma[x \rightarrow (l', \overline{ty}' \rightarrow ty)]$, and $\sigma'_1 = \sigma[l' \rightarrow (\text{NULL}, \overline{ty}' \rightarrow ty, 1, \text{PermL_Fun(public)})]$.

Given $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$ and $\text{GetFunTypeList}(\overline{p}) = \overline{ty}'$, by Lemma 3.3.1 we have $\overline{ty} = \overline{ty}'$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, $\gamma'_1 = \gamma[x \rightarrow (l', \overline{ty}' \rightarrow ty)]$, $l = l'$, and $\overline{ty} = \overline{ty}'$, we have $\gamma_1 = \gamma'_1$.

Given $\sigma_1 = \sigma[l \rightarrow (\text{NULL},\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)})]$, $\sigma'_1 = \sigma[l' \rightarrow (\text{NULL},\ \overline{ty}' \rightarrow ty,\ 1, \text{PermL_Fun(public)})]$, $l = l'$, and $\overline{ty} = \overline{ty}'$, we have $\sigma_1 = \sigma'_1$.

Therefore, we have $\gamma_1 = \gamma'_1$, $\sigma_1 = \sigma'_1$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x(\overline{p})\{s\}) \Downarrow_{fpd} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x(\overline{p})\{s\}) \Downarrow_{fpd} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Pre-Declared Function Definition, we have $\text{acc} = 0$, $x \in \gamma$, $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $\text{CheckPublicEffects}(s, x,\ \gamma,\ \sigma) = n$, $\text{EncodeFun}(s, n, \overline{p}) = \omega$, $\sigma = \sigma_1[l \rightarrow (\text{NULL},\ \overline{ty} \rightarrow ty,\ 1, \text{PermL_Fun(public)})]$, and $\sigma_2 = \sigma_1[l \rightarrow (\omega,\ \overline{ty} \rightarrow ty,\ 1, \text{PermL_Fun(public)})]$.

By definition 3.3.1, given $c = fpd$, we have $c \simeq_L c'$ if and only if $c' = fpd$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, ty\ x(\overline{p})\{s\}) \Downarrow_{fpd} (\gamma, \sigma'_2, \text{acc}, \text{skip})$ by rule Pre-Declared Function Definition, we have $\text{acc} = 0$, $x \in \gamma$, $\gamma(x) = (l', \overline{ty}' \rightarrow ty)$, $\text{CheckPublicEffects}(s, x, \gamma, \sigma) = n'$, $\text{EncodeFun}(s, n', \overline{p}) = \omega'$, $\sigma = \sigma'_1[l' \rightarrow (\text{NULL}, \overline{ty}' \rightarrow ty, 1, \text{PermL_Fun(public)})]$, and $\sigma'_2 = \sigma'_1[l' \rightarrow (\omega', \overline{ty}' \rightarrow ty, 1, \text{PermL_Fun(public)})]$.

Given $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $\gamma(x) = (l', \overline{ty}' \rightarrow ty)$, we have $l = l'$ and $\overline{ty} = \overline{ty}'$.

Given $\text{CheckPublicEffects}(s, x, \gamma, \sigma) = n$, $\text{CheckPublicEffects}(s, x, \gamma, \sigma) = n'$, by definition of $\text{CheckPublicEffects}$ we have $n = n'$.

Given $\text{EncodeFun}(s, n, \overline{p}) = \omega$, $\text{EncodeFun}(s, n', \overline{p}) = \omega'$, and $n = n'$, by definition of EncodeFun we have $\omega = \omega'$.

Given $\sigma = \sigma_1[l \rightarrow (\text{NULL}, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})]$, $\sigma = \sigma'_1[l' \rightarrow (\text{NULL}, \overline{ty}' \rightarrow ty, 1, \text{PermL_Fun(public)})]$, $l = l'$, and $\overline{ty} = \overline{ty}'$, we have $\sigma_1 = \sigma'_1$.

Given $\sigma_2 = \sigma_1[l \rightarrow (\omega, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})]$, $\sigma'_2 = \sigma'_1[l' \rightarrow (\omega', \overline{ty}' \rightarrow ty, 1, \text{PermL_Fun(public)})]$, $\sigma_1 = \sigma'_1$, $l = l'$, $\overline{ty} = \overline{ty}'$, and $\omega = \omega'$, we have $\sigma_2 = \sigma'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x(\overline{p})\{s\}) \Downarrow_{fd} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x(\overline{p})\ \{s\}) \Downarrow_{fd} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ by rule function definition, we have $l = \phi()$, $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$, $x \notin \gamma$, $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, $\text{acc} = 0$, $\text{CheckPublicEffects}(s, x, \gamma, \sigma) = n$, $\text{EncodeFun}(s, n, \overline{p}) = \omega$, and $\sigma_1 = \sigma[l \rightarrow (\omega, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})]$.

By definition 3.3.1, given $c = fd$, we have $c \simeq_L c'$ if and only if $c' = fd$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, ty\ x(\overline{p})\{s\}) \Downarrow_{fd} (\gamma'_1, \sigma'_1, \text{acc}, \text{skip})$ by rule function definition, we have $l' = \phi()$, $\text{GetFunTypeList}(\overline{p}) = \overline{ty}'$, $x \notin \gamma$, $\gamma'_1 = \gamma[x \rightarrow (l', \overline{ty}' \rightarrow ty)]$, $\text{acc} = 0$, $\text{CheckPublicEffects}(s, x, \gamma, \sigma) = n'$,

$\text{EncodeFun}(s, n', \overline{p}) = \omega'$, and $\sigma'_1 = \sigma[l' \rightarrow (\omega', \overline{ty}' \rightarrow ty, 1, \text{PermL_Fun(public)})]$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$ and $\text{GetFunTypeList}(\overline{p}) = \overline{ty}'$, by Lemma 3.3.1 we have $\overline{ty} = \overline{ty}'$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, $\gamma'_1 = \gamma[x \rightarrow (l', \overline{ty}' \rightarrow ty)]$, $l = l'$, and $\overline{ty} = \overline{ty}'$, we have $\gamma_1 = \gamma'_1$.

Given $\text{CheckPublicEffects}(s,\ x,\ \gamma,\ \sigma) = n$, $\text{CheckPublicEffects}(s,\ x,\ \gamma,\ \sigma) = n'$, by definition of $\text{CheckPublicEffects}$ we have $n = n'$.

Given $\text{EncodeFun}(s, n, \overline{p}) = \omega$, $\text{EncodeFun}(s, n', \overline{p}) = \omega'$, and $n = n'$, by definition of EncodeFun we have $\omega = \omega'$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})]$, $\sigma'_1 = \sigma[l' \rightarrow (\omega', \overline{ty}' \rightarrow ty, 1, \text{PermL_Fun(public)})]$, $l = l'$, $\overline{ty} = \overline{ty}'$, and $\omega = \omega'$, we have $\sigma_1 = \sigma'_1$.

Therefore, we have $\gamma_1 = \gamma'_1$, $\sigma_1 = \sigma'_1$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x(\overline{e})) \Downarrow_{fc} (\gamma, \sigma_2, \text{acc}, \text{NULL})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x(\overline{e})) \Downarrow_{fc} (\gamma, \sigma_2, \text{acc}, \text{NULL})$ by rule Function Call No Return With Public Side Effects, we have $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $\sigma(l) = (\omega, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})$, $\text{DecodeFun}(\omega) = (s, 1, \overline{p})$, $\text{acc} = 0$, $\text{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$, $(\gamma, \sigma, \text{acc}, s_1) \Downarrow_{c_1} (\gamma_1,\ \sigma_1, \text{acc}, \text{skip})$, and $(\gamma_1, \sigma_1, \text{acc}, s) \Downarrow_{c_2} (\gamma_2, \sigma_2, \text{acc}, \text{skip})$.

By definition 3.3.1, given $c = fc$, we have $c \simeq_L c'$ if and only if $c' = fc$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x(\overline{e})) \Downarrow_{fc} (\gamma, \sigma_2, \text{acc}, \text{NULL})$ by rule Function Call No Return With Public Side Effects, we have $\gamma(x) = (l', \overline{ty}' \rightarrow ty')$, $\sigma(l') = (\omega', \overline{ty}' \rightarrow ty', 1, \text{PermL_Fun(public)})$, $\text{DecodeFun}(\omega') = (s', 1, \overline{p}')$, $\text{acc} = 0$, $\text{GetFunParamAssign}(\overline{p}', \overline{e}) = s'_1$, $(\gamma,\ \sigma,\ \text{acc},\ s'_1) \Downarrow_{c'_1} (\gamma'_1,\ \sigma'_1, \text{acc}, \text{skip})$, and $(\gamma'_1, \sigma'_1, \text{acc}, s) \Downarrow_{c'_2} (\gamma'_2, \sigma'_2, \text{acc}, \text{skip})$.

Given $\gamma(x) = (l, \overline{ty} \rightarrow ty)$ and $\gamma(x) = (l', \overline{ty}' \rightarrow ty')$, we have $l = l'$, $\overline{ty} = \overline{ty}'$, and $ty = ty'$.

Given $\sigma(l) = (\omega, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})$, $\sigma(l') = (\omega', \overline{ty}' \rightarrow ty', 1,$ $\text{PermL_Fun(public)})$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodeFun}(\omega) = (s, 1, \overline{p})$, $\text{DecodeFun}(\omega') = (s', 1, \overline{p}')$, and $\omega = \omega'$, by definition of DecodeFun we have $s = s'$ and $\overline{p} = \overline{p}'$.

Given $\text{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$, $\text{GetFunParamAssign}(\overline{p}', \overline{e}) = s_1'$, and $\overline{p} = \overline{p}'$, by Lemma 3.3.2 we have $s_1 = s_1'$.

Given $(\gamma, \sigma, \text{acc}, s_1) \Downarrow_{c_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$, $(\gamma, \sigma, \text{acc}, s_1') \Downarrow_{c_1'} (\gamma_1', \sigma_1', \text{acc}, \text{skip})$, and $s_1 = s_1'$, by the inductive hypothesis we have $\gamma_1 = \gamma_1'$, $\sigma_1 = \sigma_1'$, and $c_1 \simeq_L c_1'$.

Given $(\gamma_1, \sigma_1, \text{acc}, s) \Downarrow_{c_2} (\gamma_2, \sigma_2, \text{acc}, \text{skip})$, $(\gamma_1', \sigma_1', \text{acc}, s) \Downarrow_{c_2'} (\gamma_2', \sigma_2', \text{acc}, \text{skip})$, $\gamma_1 = \gamma_1'$, and $\sigma_1 = \sigma_1'$, by the inductive hypothesis we have $\gamma_2 = \gamma_2'$, $\sigma_2 = \sigma_2'$, and $c_2 \simeq_L c_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, $\text{NULL} = \text{NULL}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x(\overline{e})) \Downarrow_{fc1} (\gamma, \sigma_2, \text{acc}, \text{NULL})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x(\overline{e})) \Downarrow_{fc} (\gamma, \sigma_2, \text{acc}, \text{NULL})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{malloc}(e)) \Downarrow_{mal} (\gamma, \sigma_2, \text{acc}, (l, 0))$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{malloc}(e)) \Downarrow_{mal} (\gamma, \sigma_2, \text{acc}, (l, 0))$ by rule Public Malloc, we have $\text{Label}(e, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$, $l = \phi()$, and $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n, \big[(0, \text{public}, \text{Freeable}), ..., (n-1, \text{public}, \text{Freeable})\big]\big)\big]$.

By definition 3.3.1, given $c = mal$, we have $c \simeq_L c'$ if and only if $c' = mal$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \text{malloc}(e)) \Downarrow_{mal} (\gamma, \sigma_2', \text{acc}, (l', 0))$ by rule Public Malloc, we have $\text{Label}(e, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, n')$, $l' = \phi()$, and $\sigma_2' = \sigma_1'\big[l' \rightarrow \big(\text{NULL}, \text{void}*, n', \big[(0, \text{public}, \text{Freeable}), ..., (n'-1, \text{public}, \text{Freeable})\big]\big)\big]$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n, \big[(0, \text{public, Freeable}), ..., (n-1, \text{public, Freeable})\big]\big)\big]$, $\sigma'_2 = \sigma'_1\big[l' \rightarrow \big(\text{NULL}, \text{void}*, n', \big[(0, \text{public, Freeable}), ..., (n'-1, \text{public, Freeable})\big]\big)\big]$, $\sigma_1 = \sigma'_1$, $l = l'$, and $n = n'$, we have $\sigma_2 = \sigma'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\text{acc} = \text{acc}$, $(l, 0) = (l', 0)$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{pmalloc}(e,\ ty)) \Downarrow_{malp} (\gamma,\ \sigma_2, \text{acc}, (l, 0))$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{pmalloc}(e,\ ty)) \Downarrow_{malp} (\gamma,\ \sigma_2, \text{acc}, (l, 0))$ by rule Private Malloc, we have $\text{Label}(e, \gamma) = \text{public}$, $(ty = \text{private int}) \vee (ty = \text{private float})$ $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma,\ \sigma_1, \text{acc}, n)$, $\text{acc} = 0$, $l = \phi()$, and $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL},\ ty,\ n, \text{PermL}(\text{Freeable}, ty, \text{private}, n)\big]$.

By definition 3.3.1, given $c = malp$, we have $c \simeq_L c'$ if and only if $c' = malp$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \text{pmalloc}(e,\ ty)) \Downarrow_{malp} (\gamma,\ \sigma'_2, \text{acc}, (l', 0))$ by rule Private Malloc, we have $\text{Label}(e, \gamma) = \text{public}$, $(ty = \text{private int}) \vee (ty = \text{private float})$ $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma,\ \sigma'_1, \text{acc}, n')$, $\text{acc} = 0$, $l' = \phi()$, and $\sigma'_2 = \sigma'_1\big[l' \rightarrow \big(\text{NULL},\ ty,\ n', \text{PermL}(\text{Freeable}, ty, \text{private}, n')\big]$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\sigma_2 = \sigma_1[l \rightarrow \big(\text{NULL},\ ty,\ n, \text{PermL}(\text{Freeable},\ ty,\ \text{private},\ n)]$, $\sigma'_2 = \sigma'_1[l' \rightarrow \big(\text{NULL},\ ty,\ n',\ \text{PermL}(\text{Freeable},\ ty,\ \text{private},\ n')]$, $\sigma_1 = \sigma'_1$, $l = l'$, and $n = n'$, we have $\sigma_2 = \sigma'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\text{acc} = \text{acc}$, $(l, 0) = (l', 0)$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{free}(e)) \Downarrow_{fre} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{free}(e)) \Downarrow_{fre} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Public Free, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, x)$, $\gamma(x) = (l, \text{public } bty*)$, $(bty = \text{int}) \vee (bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{void})$, $\text{acc} = 0$, and $\text{Free}(\sigma_1, l, \gamma) = \sigma_2$.

By definition 3.3.1, given $c = fre$, we have $c \simeq_L c'$ if and only if $c' = fre$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \text{free}(e)) \Downarrow_{fre} (\gamma, \sigma_2', \text{acc}, \text{skip})$ by rule Public Free, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, x')$, $\gamma(x') = (l', \text{public } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float}) \vee (bty' = \text{char}) \vee (bty' = \text{void})$, $\text{acc} = 0$, and $\text{Free}(\sigma_1', l', \gamma) = \sigma_2'$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, x)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, x')$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $x = x'$, and $c_1 \simeq_L c_1'$.

Given $\gamma(x) = (l, \text{public } bty*)$, $\gamma(x') = (l', \text{public } bty'*)$, and $x = x'$, we have $l = l'$ and $bty = bty'$.

Given $\text{Free}(\sigma_1, l, \gamma) = \sigma_2$, $\text{Free}(\sigma_1', l', \gamma) = \sigma_2'$, $\sigma_1 = \sigma_1'$, and $l = l'$, by Lemma 3.3.9 we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{pfree}(e)) \Downarrow_{frep} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{pfree}(e)) \Downarrow_{frep} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Private Free, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, x)$, $\gamma(x) = (l, \text{private } bty*)$, $\text{acc} = 0$, $(bty = \text{int}) \vee (bty = \text{float})$, and $\text{PFree}(\gamma, \sigma_1, l) = (\sigma_2, \overline{j})$.

By definition 3.3.1, given $c = frep$, we have $c \simeq_L c'$ if and only if $c' = frep$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \text{pfree}(e)) \Downarrow_{frep} (\gamma, \sigma_2', \text{acc}, \text{skip})$ by rule Private Free, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, x')$, $\gamma(x') = (l', \text{private } bty'*)$, $\text{acc} = 0$, $(bty' = \text{int}) \vee (bty' = \text{float})$, and $\text{PFree}(\gamma, \sigma_1', l') = (\sigma_2', \overline{j}')$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, x)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, x')$, by the inductive hypothesis we have

$\sigma_1 = \sigma_1'$, $x = x'$, and $c_1 \simeq_L c_1'$.

Given $\gamma(x) = (l,\ \text{private}\ bty*)$, $\gamma(x') = (l',\ \text{private}\ bty'*)$, and $x = x'$, we have $l = l'$ and $bty = bty'$.

Given $\text{PFree}(\gamma, \sigma_1,\ l) = (\sigma_2, \overline{j})$, $\text{PFree}(\gamma, \sigma_1',\ l') = (\sigma_2', \overline{j}')$, $\sigma_1 = \sigma_1'$, and $l = l'$, by Lemma 3.3.10 we have $\sigma_2 = \sigma_2'$ and $\overline{j} = \overline{j}'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x = e) \Downarrow_{ds} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x = e) \Downarrow_{ds} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$ by rule Declaration Assignment, we have $(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{c_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$, and $(\gamma_1, \sigma_1, \text{acc}, x = e) \Downarrow_{c_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$.

By definition 3.3.1, given $c = ds$, we have $c \simeq_L c'$ if and only if $c' = ds$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, ty\ x = e) \Downarrow_{ds} (\gamma_1', \sigma_2', \text{acc}, \text{skip})$ by rule Declaration Assignment, we have $(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{c_1'} (\gamma_1', \sigma_1', \text{acc}, \text{skip})$, and $(\gamma_1', \sigma_1', \text{acc}, x = e) \Downarrow_{c_2'} (\gamma_1', \sigma_2', \text{acc}, \text{skip})$.

Given $(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{c_1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ and $(\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{c_1'} (\gamma_1', \sigma_1', \text{acc}, \text{skip})$, by the inductive hypothesis we have $\gamma_1 = \gamma_1'$, $\sigma_1 = \sigma_1'$, and $c_1 \simeq_L c_1'$.

Given $(\gamma_1, \sigma_1, \text{acc}, x = e) \Downarrow_{c_2} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$, $(\gamma_1', \sigma_1', \text{acc}, x = e) \Downarrow_{c_2'} (\gamma_1', \sigma_2', \text{acc}, \text{skip})$, $\gamma_1 = \gamma_1'$, and $\sigma_1 = \sigma_1'$, by the inductive hypothesis we have $\sigma_2 = \sigma_2'$, and $c_2 \simeq_L c_2'$.

Therefore, we have $\gamma_1 = \gamma_1'$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x[e_1] = e_2) \Downarrow_{das} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$
This case is similar to case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x = e) \Downarrow_{ds} (\gamma_1, \sigma_2, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{d} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_d (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ by rule Public Declaration, we have $(ty = \text{public}\ bty) \vee (ty = \text{char})$, $\text{acc} = 0$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\omega = \text{EncodeVal}(ty, \text{NULL})$, and $\sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$.

By definition 3.3.1, given $c = d$, we have $c \simeq_L c'$ if and only if $c' = d$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_d (\gamma_1', \sigma_1', \text{acc}, \text{skip})$ by rule Public Declaration, we have $(ty = \text{public}\ bty) \vee (ty = \text{char})$, $\text{acc} = 0$, $l' = \phi()$, $\gamma_1' = \gamma[x \rightarrow (l', ty)]$, $\omega' = \text{EncodeVal}(ty, \text{NULL})$, and $\sigma_1' = \sigma[l' \rightarrow (\omega', ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\gamma_1' = \gamma[x \rightarrow (l', ty)]$, and $l = l'$, we have $\gamma_1 = \gamma_1'$.

Given $\omega = \text{EncodeVal}(ty, \text{NULL})$ and $\omega' = \text{EncodeVal}(ty, \text{NULL})$, by Lemma 3.3.3 we have $\omega = \omega'$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$, $\sigma_1' = \sigma[l' \rightarrow (\omega', ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$, $l = l'$, and $\omega = \omega'$, we have $\sigma_1 = \sigma_1'$.

Therefore, we have $\gamma_1 = \gamma_1'$, $\sigma_1 = \sigma_1'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{d1} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_d (\gamma_1, \sigma_1, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{dp} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x) \Downarrow_{dp} (\gamma_1, \sigma_1, \text{acc}, \text{skip})$ by rule Public Pointer Declaration, we have $(ty = \text{public}\ bty*) \vee ((ty = bty*) \wedge ((bty = \text{char}) \vee (bty = \text{void})))$, $\text{GetIndirection}(*) = i$, $\text{acc} = 0$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\omega = \text{EncodePtr}(ty, [1, [l_{default}], [1], i])$, and $\sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1))]$.

By definition 3.3.1, given $c = dp$, we have $c \simeq_L c'$ if and only if $c' = dp$.

Given $\Sigma \triangleright (\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_{dp} (\gamma_1', \sigma_1',\ \text{acc},\ \text{skip})$ by rule Public Pointer Declaration, we have $(ty = \text{public}\ bty'*) \vee ((ty = bty'*) \wedge ((bty' = \text{char}) \vee (bty' = \text{void})))$, $\text{GetIndirection}(*) = i'$, $\text{acc} = 0$, $l' = \phi()$, $\gamma_1' = \gamma[x \rightarrow (l', ty)]$, $\omega' = \text{EncodePtr}(ty,\ [1,\ [l_{default}],\ [1],\ i'])$, and $\sigma_1' = \sigma[l' \rightarrow (\omega',\ ty,\ 1,\ \text{PermL}(\text{Freeable, public}\ bty*, \text{public}, 1))]$.

Given $(ty = \text{public}\ bty*) \vee ((ty = bty*) \wedge ((bty = \text{char}) \vee (bty = \text{void})))$ and $(ty = \text{public}\ bty'*) \vee ((ty = bty'*) \wedge ((bty' = \text{char}) \vee (bty' = \text{void})))$, we have $bty = bty'$.

Given $\text{GetIndirection}(*) = i$ and $\text{GetIndirection}(*) = i'$, by Lemma 3.3.11 we have $i = i'$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\gamma_1' = \gamma[x \rightarrow (l', ty)]$, and $l = l'$, we have $\gamma_1 = \gamma_1'$.

Given $\omega = \text{EncodePtr}(ty, [1, [l_{default}], [1], i])$, $\omega' = \text{EncodePtr}(ty, [1, [l_{default}], [1], i'])$, and $i = i'$, by Lemma 3.3.5 we have $\omega = \omega'$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, \text{public}\ bty*, 1,\ \text{PermL}(\text{Freeable, public}\ bty*, \text{public}, 1))]$, $\sigma_1' = \sigma[l' \rightarrow (\omega', \text{public}\ bty'*, 1, \text{PermL}(\text{Freeable, public}\ bty'*, \text{public}, 1))]$, $l = l'$, $\omega = \omega'$, and $bty = bty'$, we have $\sigma_1 = \sigma_1'$.

Therefore, we have $\gamma_1 = \gamma_1'$, $\sigma_1 = \sigma_1'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_{dp1} (\gamma_1, \sigma_1,\ \text{acc},\ \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ ty\ x) \Downarrow_{dp} (\gamma_1, \sigma_1,\ \text{acc},\ \text{skip})$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ x) \Downarrow_{r} (\gamma,\ \sigma,\ \text{acc},\ v)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ x) \Downarrow_{r} (\gamma,\ \sigma,\ \text{acc},\ v)$ by rule Read Public Variable, we have $\gamma(x) = (l, \text{public}\ bty)$, $\sigma(l) = (\omega, \text{public}\ bty, 1, \text{PermL}(\text{Freeable, public}\ bty, \text{public}, 1))$, and $\text{DecodeVal}(\text{public}\ bty,\ 1,\ \omega) = v$.

By definition 3.3.1, given $c = r$, we have $c \simeq_L c'$ if and only if $c' = r$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_r (\gamma, \sigma, \text{acc}, v')$ by rule Read Public Variable, we have $\gamma(x) = (l', \text{public } bty')$, $\sigma(l') = (\omega', \text{public } bty', 1, \text{PermL}(\text{Freeable, public } bty', \text{public}, 1))$, and $\text{DecodeVal}(\text{public } bty', 1, \omega') = v'$.

Given $\gamma(x) = (l, \text{public } bty)$ and $\gamma(x) = (l', \text{public } bty')$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable, public } bty, \text{public}, 1))$, $\sigma(l') = (\omega', \text{public } bty', 1, \text{PermL}(\text{Freeable, public } bty', \text{public}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodeVal}(\text{public } bty, 1, \omega) = v$, $\text{DecodeVal}(\text{public } bty', 1, \omega') = v'$, $bty = bty'$, and $\omega = \omega'$, we have $v = v'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\text{acc} = \text{acc}$, $v = v$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{r1} (\gamma, \sigma, \text{acc}, v)$
This case is similar to case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_r (\gamma, \sigma, \text{acc}, v)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_w (\gamma, \sigma_2, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_w (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Public Write Variable, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, v)$, $v \neq \text{skip}$, $\text{acc} = 0$, $\gamma(x) = (l, \text{public } bty)$, and $\text{UpdateVal}(\sigma_1, l, v, \text{public } bty) = \sigma_2$.

By definition 3.3.1, given $c = w$, we have $c \simeq_L c'$ if and only if $c' = w$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_w (\gamma, \sigma_2', \text{acc}, \text{skip})$ by rule Write Public Variable, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, v')$, $v' \neq \text{skip}$,$\text{acc} = 0$, $\gamma(x) = (l', \text{public } bty')$, and $\text{UpdateVal}(\sigma_1', l', v', \text{public } bty') = \sigma_2'$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, v)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, v')$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $v = v'$, and $c_1 \simeq_L c_1'$.

Given $\gamma(x) = (l, \text{public } bty)$ and $\gamma(x) = (l', \text{public } bty')$, we have $l = l'$ and $bty = bty'$.

Given $\text{UpdateVal}(\sigma_1, l, v, \text{public } bty) = \sigma_2$, $\text{UpdateVal}(\sigma_1', l', v', \text{public } bty') = \sigma_2'$, $\sigma_1 = \sigma_1'$, $l = l'$, $bty = bty'$, and $v = v'$, by Lemma 3.3.12 we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w2} (\gamma, \sigma_2, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w} (\gamma, \sigma_2, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w1} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w1} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Write Private Variable Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$, $\gamma(x) = (l, \text{private } bty)$, and $\text{UpdateVal}(\sigma_1, l, \text{encrypt}(n), \text{private } bty) = \sigma_2$.

By definition 3.3.1, given $c = w1$, we have $c \simeq_L c'$ if and only if $c' = w1$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{w1} (\gamma, \sigma_2', \text{acc}, \text{skip})$ by rule Write Private Variable Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1', \text{acc}, n')$, $\gamma(x) = (l', \text{private } bty')$, and $\text{UpdateVal}(\sigma_1', l', \text{encrypt}(n'), \text{private } bty') = \sigma_2'$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1', \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $n = n'$, and $c_1 \simeq_L c_1'$.

Given $\gamma(x) = (l, \text{private } bty)$ and $\gamma(x) = (l', \text{private } bty')$, we have $l = l'$ and $bty = bty'$.

Given $\text{UpdateVal}(\sigma_1, l, \text{encrypt}(n), \text{private } bty) = \sigma_2$, $\text{UpdateVal}(\sigma_1', l', \text{encrypt}(n'), \text{private } bty') = \sigma_2'$, $\sigma_1 = \sigma_1'$, $l = l'$, $bty = bty'$, and $n = n'$, by Axiom 3.3.1 we have $\text{encrypt}(n) = \text{encrypt}(n')$ and therefore by Lemma 3.3.12 we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{rp} (\gamma, \sigma, \text{acc}, (l_1, \mu_1))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{rp} (\gamma, \sigma, \text{acc}, (l_1, \mu_1))$ by rule Public Pointer Read Single Location, we have $\gamma(x) = (l,\ \text{public}\ bty*)$, $\sigma(l) = (\omega, \text{public}\ bty*, 1, \text{PermL(Freeable, public}\ bty*, \text{public}, 1))$, and DecodePtr(public $bty*,\ 1,\ \omega) = [1, [(l_1, \mu_1)], [1], i]$.

By definition 3.3.1, given $c = rp$, we have $c \simeq_L c'$ if and only if $c' = rp$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{rp} (\gamma, \sigma, \text{acc}, (l'_1, \mu'_1))$ by rule Public Pointer Read Single Location, we have $\gamma(x) = (l', \text{public}\ bty'*)$, $\sigma(l') = (\omega', \text{public}\ bty'*, 1, \text{PermL(Freeable, public}\ bty'*, \text{public}, 1))$, and DecodePtr(public $bty'*,\ 1,\ \omega') = [1, [(l'_1, \mu'_1)], [1], i']$.

Given $\gamma(x) = (l,\ \text{public}\ bty*)$ and $\gamma(x) = (l',\ \text{public}\ bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{public}\ bty*, 1, \text{PermL(Freeable, public}\ bty*, \text{public}, 1))$, $\sigma(l') = (\omega', \text{public}\ bty'*, 1, \text{PermL(Freeable, public}\ bty'*, \text{public}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given DecodePtr(public $bty*,\ 1,\ \omega) = [1, [(l_1, \mu_1)], [1], i]$, DecodePtr(public $bty'*,\ 1,\ \omega') = [1, [(l'_1, \mu'_1)], [1], i']$, $bty = bty'$, and $\omega = \omega'$, we have $l_1 = l'_1$, $\mu_1 = \mu'_1$, and $i = i'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, acc = acc, $(l_1, \mu_1) = (l'_1, \mu'_1)$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{rp2} (\gamma, \sigma, \text{acc}, (l_1, \mu_1))$

This case is similar to case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{rp} (\gamma, \sigma, \text{acc}, (l_1, \mu_1))$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{rp1} (\gamma, \sigma, \text{acc}, [\alpha, \bar{l}, \bar{j}, i])$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{rp1} (\gamma, \sigma, \text{acc}, [\alpha, \bar{l}, \bar{j}, i])$ by rule Private Pointer Read Multiple Locations, we have $\gamma(x) = (l,\ \text{private}\ bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega,\ \text{private}\ bty*,\ \alpha, \text{PermL(Freeable, private}\ bty*, \text{private}, \alpha))$, and DecodePtr(private $bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$.

By definition 3.3.1, given $c = rp1$, we have $c \simeq_L c'$ if and only if $c' = rp1$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{rp1} (\gamma, \sigma, \text{acc}, [\alpha', \bar{l}', \bar{j}', i'])$ by rule Private Pointer Read Multiple Locations, we have $\gamma(x) = (l', \text{ private } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma(l') = (\omega', \text{ private } bty'*, \alpha', \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, \alpha'))$, and $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', i']$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{ private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{ private } bty*, \text{ private}, \alpha))$, $\sigma(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, \alpha'))$, and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', i']$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\text{acc} = \text{acc}$, $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Public Pointer Write Single Location, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, (l_e, \mu_e))$, $\gamma(x) = (l, \text{public } bty*)$, $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\text{acc} = 0$, $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, and $\text{UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \text{public } bty*) = (\sigma_2, 1)$.

By definition 3.3.1, given $c = wp$, we have $c \simeq_L c'$ if and only if $c' = wp$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Public Pointer Write Single Location, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, (l'_e, \mu'_e))$, $\gamma(x) = (l', \text{public } bty'*)$, $\sigma'_1(l') = (\omega', \text{public } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, $\text{acc} = 0$, $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l'_1, \mu'_1)], [1], i']$, and $\text{UpdatePtr}(\sigma'_1, (l', 0), [1, [(l'_e, \mu'_e)], [1], i'], \text{public } bty'*) = (\sigma'_2, 1)$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, (l_e, \mu_e))$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, (l'_e, \mu'_e))$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $l_e = l'_e$, $\mu_e = \mu'_e$, and $c_1 \simeq_L c'_1$.

Given $\gamma(x) = (l, \text{public } bty*)$ and $\gamma(x) = (l', \text{public } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\sigma_1'(l') = (\omega', \text{public } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, $\sigma_1 = \sigma_1'$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l_1', \mu_1')], [1], i']$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $l_1 = l_1'$, $\mu_1 = \mu_1'$, and $i = i'$.

Given $\text{UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \text{public } bty*) = (\sigma_2, 1)$, $\text{UpdatePtr}(\sigma_1', (l', 0), [1, [(l_e', \mu_e')], [1], i'], \text{public } bty'*) = (\sigma_2', 1)$, $\sigma_1 = \sigma_1'$, $l = l'$, $l_e = l_e'$, $\mu_e = \mu_e'$, $i = i'$, and $bty = bty'$, by Lemma 3.3.15 we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp1} (\gamma, \sigma_2, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp} (\gamma, \sigma_2, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp2} (\gamma, \sigma_2, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp2} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Private Pointer Write Multiple Locations, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}, \bar{j}, i])$, $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, and $\text{UpdatePtr}(\sigma_1, (l, 0), [\alpha, \bar{l}, \bar{j}, i], \text{private } bty*) = (\sigma_2, 1)$.

By definition 3.3.1, given $c = wp2$, we have $c \simeq_L c'$ if and only if $c' = wp2$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wp2} (\gamma, \sigma_2', \text{acc}, \text{skip})$ by rule Private Pointer Write Multiple Locations, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, [\alpha', \bar{l}', \bar{j}', i'])$, $\gamma(x) = (l', \text{private } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, and $\text{UpdatePtr}(\sigma_1', (l', 0), [\alpha', \bar{l}', \bar{j}', i'], \text{private } bty'*) = (\sigma_2', 1)$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}, \bar{j}, i])$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, [\alpha', \bar{l}', \bar{j}', i'])$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$, and $c_1 \simeq_L c_1'$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\mathrm{UpdatePtr}(\sigma_1, (l, 0), [\alpha, \bar{l}, \bar{j}, i], \text{private } bty*) = (\sigma_2, 1)$, $\mathrm{UpdatePtr}(\sigma_1', (l, 0), [\alpha', \bar{l}', \bar{j}', i'], \text{private } bty'*) = (\sigma_2', 1)$, $\sigma_1 = \sigma_1'$, $l = l'$, $bty = bty'$, and $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$, by Lemma 3.3.15 we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma_1'$, $\mathrm{acc} = \mathrm{acc}$, $\mathrm{skip} = \mathrm{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, *x) \Downarrow_{rdp} (\gamma, \sigma, \mathrm{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, *x) \Downarrow_{rdp} (\gamma, \sigma, \mathrm{acc}, v)$ by rule Public Pointer Dereference Single Location, we have $\gamma(x) = (l, \text{public } bty*)$, $\sigma(l) = (\omega, \text{public } bty*, 1, \mathrm{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\mathrm{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, and $\mathrm{DerefPtr}(\sigma, \text{public } bty, (l_1, \mu_1)) = (v, 1)$.

By definition 3.3.1, given $c = rdp$, we have $c \simeq_L c'$ if and only if $c' = rdp$.

Given $\Sigma \triangleright (\gamma, \sigma, \mathrm{acc}, *x) \Downarrow_{rdp} (\gamma, \sigma, \mathrm{acc}, v)$ by rule Public Pointer Dereference Single Location, we have $\gamma(x) = (l', \text{public } bty'*)$, $\sigma(l') = (\omega', \text{public } bty'*, 1, \mathrm{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, $\mathrm{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l_1', \mu_1')], [1], 1]$, and $\mathrm{DerefPtr}(\sigma, \text{public } bty', (l_1', \mu_1')) = (v', 1)$.

Given $\gamma(x) = (l, \text{public } bty*)$ and $\gamma(x) = (l', \text{public } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{public } bty*, 1, \mathrm{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\sigma(l') = (\omega', \text{public } bty'*, 1, \mathrm{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\mathrm{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, $\mathrm{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l_1', \mu_1)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, \mu_1)], [1], 1] = [1, [(l_1', \mu_1)], [1], 1]$ and therefore $(l_1, \mu_1) = (l_1', \mu_1')$.

Given $\mathrm{DerefPtr}(\sigma, \text{public } bty, (l_1, \mu_1)) = (v, 1)$, $\mathrm{DerefPtr}(\sigma, \text{public } bty', (l_1', \mu_1')) = (v', 1)$, $bty = bty'$, $(l_1, \mu_1) = (l_1', \mu_1')$, by Lemma 3.3.19 we have $v = v'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\mathrm{acc} = \mathrm{acc}$, $v = v'$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, *x) \Downarrow_{rdp1} (\gamma, \sigma, \mathrm{acc}, (l_2, \mu_2))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp1} (\gamma, \sigma, \text{acc}, (l_2, \mu_2))$ by rule Public Pointer Dereference Single Location Higher Level Indirection, we have $\gamma(x) = (l,\ \text{public}\ bty*)$, $\sigma(l) = (\omega,\ \text{public}\ bty*,\ 1, \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public}\ bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, $i > 1$, and $\text{DerefPtrHLI}(\sigma, \text{public}\ bty*, (l_1, \mu_1)) = ([1, [(l_2, \mu_2)], [1], i-1], 1)$.

By definition 3.3.1, given $c = rdp1$, we have $c \simeq_L c'$ if and only if $c' = rdp$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp1} (\gamma, \sigma, \text{acc}, (l'_2, \mu'_2))$ by rule Public Pointer Dereference Single Location Higher Level Indirection, we have $\gamma(x) = (l', \text{public}\ bty'*)$, $\sigma(l') = (\omega', \text{public}\ bty'*, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty'*, \text{public}, 1))$, $\text{DecodePtr}(\text{public}\ bty'*, 1, \omega') = [1, [(l'_1, \mu'_1)], [1], i']$, $i' > 1$, and $\text{DerefPtrHLI}(\sigma, \text{public}\ bty'*, (l'_1, \mu'_1)) = ([1, [(l'_2, \mu'_2)], [1], i'-1], 1)$.

Given $\gamma(x) = (l, \text{public}\ bty*)$ and $\gamma(x) = (l', \text{public}\ bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{public}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1))$, $\sigma(l') = (\omega', \text{public}\ bty'*, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty'*, \text{public}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{public}\ bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, $\text{DecodePtr}(\text{public}\ bty'*, 1, \omega') = [1, [(l'_1, \mu'_1)], [1], i']$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, \mu_1)], [1], i] = [1, [(l'_1, \mu'_1)], [1], i']$ and therefore $(l_1, \mu_1) = (l'_1, \mu'_1)$ and $i = i'$.

Given $\text{DerefPtrHLI}(\sigma, \text{public}\ bty*, (l_1, \mu_1)) = ([1, [(l_2, \mu_2)], [1], i-1], 1)$, $\text{DerefPtrHLI}(\sigma, \text{public}\ bty'*, (l'_1, \mu'_1)) = ([1, [(l'_2, \mu'_2)], [1], i'-1], 1)$, $bty = bty'$, and $(l_1, \mu_1) = (l'_1, \mu'_1)$, by Lemma 3.3.20 we have $[1, [(l_2, \mu_2)], [1], i-1] = [1, [(l'_2, \mu'_2)], [1], i'-1]$ and therefore $(l_2, \mu_2) = (l'_2, \mu'_2)$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\text{acc} = \text{acc}$, $(l_2, \mu_2) = (l'_2, \mu'_2)$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp2} (\gamma, \sigma, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp2} (\gamma, \sigma, \text{acc}, v)$ by rule Private Pointer Dereference, we have $\gamma(x) = (l, \text{private}\ bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private}\ bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private}\ bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, and $\text{Retrieve_vals}(\alpha, \bar{l}, \bar{j}, \text{private}\ bty, \sigma) = (v, 1)$.

By definition 3.3.1, given $c = rdp2$, we have $c \simeq_L c'$ if and only if $c' = rdp2$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp2} (\gamma, \sigma, \text{acc}, v)$ by rule Private Pointer Dereference, we have $\gamma(x) = (l', \text{private } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL(Freeable, private } bty'*, \text{private, } \alpha'))$, $\text{DecodePtr(private } bty'*, \alpha', \omega) = [\alpha', \bar{l}', \bar{j}', 1]$, and $\text{Retrieve_vals}(\alpha', \bar{l}', \bar{j}', \text{private } bty', \sigma) = (v', 1)$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL(Freeable, private } bty*, \text{private, } \alpha))$, $\sigma(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL(Freeable, private } bty'*, \text{private, } \alpha'))$, and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given $\text{DecodePtr(private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, $\text{DecodePtr(private } bty'*, \alpha', \omega) = [\alpha', \bar{l}', \bar{j}', 1]$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \bar{l}, \bar{j}, 1] = [\alpha', \bar{l}, \bar{j}, 1]$. Therefore, we have $\bar{l} = \bar{l}'$ and $\bar{j} = \bar{j}'$.

Given $\text{Retrieve_vals}(\alpha, \bar{l}, \bar{j}, \text{private } bty, \sigma) = (v, 1)$, $\text{Retrieve_vals}(\alpha', \bar{l}', \bar{j}', \text{private } bty', \sigma) = (v', 1)$, $\alpha = \alpha'$, $\bar{l} = \bar{l}'$, $\bar{j} = \bar{j}'$, and $bty = bty'$, by Lemma 3.3.17 we have $v = v'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\text{acc} = \text{acc}$, $v = v'$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow^*_{rdp2} (\gamma, \sigma, \text{acc}, v)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp2} (\gamma, \sigma, \text{acc}, v)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp3} (\gamma, \sigma, \text{acc}, [\alpha', \bar{l}', \bar{j}', i-1])$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp3} (\gamma, \sigma, \text{acc}, [\alpha', \bar{l}', \bar{j}', i-1])$ by rule Private Pointer Dereference Higher Level Indirection, we have $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL(Freeable, private } bty*, \text{private, } \alpha))$, $\text{DecodePtr(private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$, $i > 1$, and $\text{DerefPrivPtr}(\alpha, \bar{l}, \bar{j}, \text{private } bty*, \sigma) = ((\alpha', \bar{l}', \bar{j}'), 1)$.

By definition 3.3.1, given $c = rdp3$, we have $c \simeq_L c'$ if and only if $c' = rdp3$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp3} (\gamma, \sigma, \text{acc}, [\alpha''', \bar{l}''', \bar{j}''', i'-1])$ by rule Private Pointer Dereference Higher Level Indirection, we have $\gamma(x) = (l', \text{private } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma(l') = (\omega', \text{private } bty'*, \alpha'',$

PermL(Freeable, private $bty'*$, private, α'')),
DecodePtr(private $bty'*$, α'', ω') $= [\alpha'', \bar{l}'', \bar{j}'', i']$, $i' > 1$, and DerefPrivPtr(α'', $\bar{l}''$, $\bar{j}''$, private $bty'*$, σ) $=$ $((\alpha''', \bar{l}''', \bar{j}'''), 1)$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega$, private $bty*$, α, PermL(Freeable, private $bty*$, private, α)), $\sigma(l') = (\omega'$, private $bty'*$, α'', PermL(Freeable, private $bty'*$, private, α'')), and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha''$.

Given DecodePtr(private $bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$, DecodePtr(private $bty'*, \alpha'', \omega') = [\alpha'', \bar{l}'', \bar{j}'', i']$, $bty = bty'$, $\alpha = \alpha''$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \bar{l}, \bar{j} = [\alpha'', \bar{l}'', \bar{j}'', i']$. Therefore, we have $\bar{l} = \bar{l}''$, $\bar{j} = \bar{j}''$, and $i = i'$.

Given DerefPrivPtr($\alpha, \bar{l}, \bar{j}$, private $bty*, \sigma) = ((\alpha', \bar{l}', \bar{j}'), 1)$, DerefPrivPtr($\alpha''$, $\bar{l}''$, $\bar{j}''$, private $bty'*$, $\sigma) = ((\alpha'''$, $\bar{l}''', \bar{j}'''), 1)$, $\alpha = \alpha''$, $\bar{l} = \bar{l}''$, $\bar{j} = \bar{j}''$, and $bty = bty'$, by Lemma 3.3.18 we have $(\alpha', \bar{l}', \bar{j}') = (\alpha''', \bar{l}''', \bar{j}''')$.

Given $(\alpha', \bar{l}', \bar{j}') = (\alpha''', \bar{l}''', \bar{j}''')$ and $i = i'$, we have $[\alpha', \bar{l}', \bar{j}', i-1] = ([\alpha''', \bar{l}''', \bar{j}''', i'-1]$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, acc = acc, $[\alpha', \bar{l}', \bar{j}', i-1] = ([\alpha''', \bar{l}''', \bar{j}''', i'-1]$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp3} (\gamma, \sigma, \text{acc}, [\alpha', \bar{l}', \bar{j}', i-1])$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x) \Downarrow_{rdp3} (\gamma, \sigma, \text{acc}, [\alpha', \bar{l}', \bar{j}', i-1])$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp} (\gamma, \sigma_2, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Public Pointer Dereference Write Public Value, we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{public } bty*)$, $\sigma_1(l) = (\omega,$ public $bty*$, 1, PermL(Freeable, public $bty*$, public, 1)), DecodePtr(public $bty*$, 1, $\omega) = [1, [(l_1, 0)], [1], 1]$, UpdateOffset($\sigma_1, (l_1, \mu_1), v$, public $bty) = (\sigma_2, 1)$, Label$(e, \gamma) =$ public, and acc $= 0$.

By definition 3.3.1, given $c = wdp$, we have $c \simeq_L c'$ if and only if $c' = wdp$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp} (\gamma, \sigma_2', \text{acc}, \text{skip})$ by rule Public Pointer Dereference Write Public Value,

we have $(\gamma,\, \sigma,\, \text{acc},\, e) \Downarrow_{c_1'} (\gamma,\, \sigma_1',\, \text{acc},\, v')$, $v' \neq \text{skip}$, $\gamma(x) = (l', \text{public } bty'*)$, $\sigma_1'(l') = (\omega',\, \text{public } bty'*, 1,$ $\text{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1)), \text{DecodePtr}(\text{public } bty'*,\, 1,\, \omega') = [1,\, [(l_1',\, 0)],\, [1],\, 1], \text{UpdateOffset}(\sigma_1',$ $(l_1', \mu_1'), v', \text{public } bty') = (\sigma_2', 1)$, $\text{Label}(e, \gamma) = \text{public}$, and $\text{acc} = 0$.

Given $(\gamma,\, \sigma,\, \text{acc},\, e) \Downarrow_{c_1} (\gamma,\, \sigma_1,\, \text{acc},\, v)$ and $(\gamma,\, \sigma,\, \text{acc},\, e) \Downarrow_{c_1'} (\gamma,\, \sigma_1',\, \text{acc},\, v')$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $v = v'$, and $c_1 \simeq_L c_1'$.

Given $\gamma(x) = (l, \text{public } bty*)$ and $\gamma(x) = (l', \text{public } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega,\, \text{public } bty*,\, 1,\, \text{PermL}(\text{Freeable},\, \text{public } bty*,\, \text{public},\, 1))$, $\sigma_1'(l') = (\omega',\, \text{public } bty'*,\, 1,$ $\text{PermL}(\text{Freeable},\, \text{public } bty'*,\, \text{public},\, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1,\, [(l_1,\, 0)],\, [1],\, 1]$, $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1,\, [(l_1',\, 0)],\, [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1,\, [(l_1,\, 0)],\, [1],\, 1] = [1,\, [(l_1',\, 0)],\, [1],\, 1]$, and therefore $l_1 = l_1'$.

Given $\text{UpdateOffset}(\sigma_1, (l_1, \mu_1), v, \text{public } bty) = (\sigma_2, 1)$, $\text{UpdateOffset}(\sigma_1', (l_1', \mu_1'), v', \text{public } bty') = (\sigma_2', 1)$, $\sigma_1 = \sigma_1'$, $l_1 = l_1'$, $bty = bty$, and $v = v'$, by Lemma 3.3.13 we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma,\, \sigma,\, \text{acc},\, *x =\ e) \Downarrow^*_{wdp} (\gamma,\, \sigma_2,\, \text{acc},\, \text{skip})$
This case is similar to case $\Pi \triangleright (\gamma,\, \sigma,\, \text{acc},\, *x =\ e) \Downarrow_{wdp} (\gamma,\, \sigma_2,\, \text{acc},\, \text{skip})$.

Case $\Pi \triangleright (\gamma,\, \sigma,\, \text{acc},\, *x = e) \Downarrow_{wdp1} (\gamma,\, \sigma_2,\, \text{acc},\, \text{skip})$
Given $\Pi \triangleright (\gamma,\, \sigma,\, \text{acc},\, *x = e) \Downarrow_{wdp1} (\gamma,\, \sigma_2,\, \text{acc},\, \text{skip})$ by rule Public Pointer Dereference Write Higher Level Indirection, we have $\text{acc} = 0$, $(\gamma,\, \sigma,\, \text{acc},\, e) \Downarrow_{c_1} (\gamma,\, \sigma_1,\, \text{acc},\, (l_e, \mu_e))$, $\gamma(x) = (l, \text{public } bty*)$, $\sigma_1(l) = (\omega,\, \text{public}$ $bty*, 1,\, \text{PermL}(\text{Freeable},\, \text{public } bty*, \text{public},\, 1))$, $\text{DecodePtr}(\text{public } bty*, 1,\, \omega) = [1,\, [(l_1,\, \mu_1)],\, [1],\, i]$, $i > 1$, $\text{Label}(e, \gamma) = \text{public}$, and $\text{UpdatePtr}(\sigma_1,\, (l_1, \mu_1),\, [1,\, [(l_e,\, \mu_e)],\, [1],\, i - 1],\, \text{public } bty*) = (\sigma_2, 1)$.

By definition 3.3.1, given $c = wdp1$, we have $c \simeq_L c'$ if and only if $c' = wdp1$.

Given $\Sigma \triangleright (\gamma,\, \sigma,\, \text{acc},\, *x = e) \Downarrow_{wdp1} (\gamma,\, \sigma_2',\, \text{acc},\, \text{skip})$ by rule Public Pointer Dereference Write Higher Level

Indirection, we have acc $= 0$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, (l_e', \mu_e'))$, $\gamma(x) = (l', \text{public } bty'*)$, $\sigma_1'(l') = (\omega', \text{public } bty'*, 1, \text{PermL(Freeable, public } bty'*, \text{public, } 1))$, $\text{DecodePtr(public } bty'*, 1, \omega') = [1, [(l_1', \mu_1')], [1], i']$, $i' > 1$, $\text{Label}(e, \gamma) = \text{public}$, and $\text{UpdatePtr}(\sigma_1', (l_1', \mu_1'), [1, [(l_e', \mu_e')], [1], i' - 1], \text{public } bty'*) = (\sigma_2', 1)$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, (l_e, \mu_e))$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, (l_e', \mu_e'))$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $(l_e, \mu_e) = (l_e', \mu_e')$, and $c_1 \simeq_L c_1'$.

Given $\gamma(x) = (l, \text{public } bty*)$ and $\gamma(x) = (l', \text{public } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL(Freeable, public } bty*, \text{public, } 1))$, $\sigma_1'(l') = (\omega', \text{public } bty'*, 1, \text{PermL(Freeable, public } bty'*, \text{public}, 1))$, $\sigma_1 = \sigma_1'$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr(public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, $\text{DecodePtr(public } bty'*, 1, \omega') = [1, [(l_1', \mu_1')], [1], i']$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, \mu_1)], [1], i] = [1, [(l_1', \mu_1')], [1], i']$. Therefore, we have $(l_1, \mu_1) = (l_1', \mu_1')$ and $i = i'$.

Given $\text{UpdatePtr}(\sigma_1, (l_1, \mu_1), [1, [(l_e, \mu_e)], [1], i - 1], \text{public } bty*) = (\sigma_2, 1)$, $\text{UpdatePtr}(\sigma_1', (l_1', \mu_1'), [1, [(l_e', \mu_e')], [1], i' - 1], \text{public } bty'*) = (\sigma_2', 1)$, $\sigma_1 = \sigma_1'$, $(l_1, \mu_1) = (l_1', \mu_1')$, $(l_e, \mu_e) = (l_e', \mu_e')$, $bty = bty'$, and $i = i'$, by Lemma 3.3.15 we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, acc $=$ acc, skip $=$ skip, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp1}^{*} (\gamma, \sigma_2, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp1} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp2} (\gamma, \sigma_2, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp2} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Private Pointer Dereference Write Higher Level Indirection, we have $\gamma(x) = (l, \text{private } bty*)$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, (l_e, \mu_e))$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL(Freeable, private } bty*, \text{private}, \alpha))$, $\text{DecodePtr(private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$, and $\text{UpdatePrivPtr}(\sigma_1, [\alpha, \bar{l}, \bar{j}, i], [1, [(l_e, \mu_e)], [1], i - 1], \text{private } bty*) = (\sigma_2, 1)$.

By definition 3.3.1, given $c = wdp2$, we have $c \simeq_L c'$ if and only if $c' = wdp2$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp2} (\gamma, \sigma_2', \text{acc}, \text{skip})$ by rule Private Pointer Dereference Write Higher Level Indirection, we have $\gamma(x) = (l', \text{private } bty'*)$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, (l_e', \mu_e'))$, $\sigma_1'(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL(Freeable, private } bty'*, \text{private, } \alpha'))$, $\text{DecodePtr(private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', i']$, and $\text{UpdatePrivPtr}(\sigma_1', [\alpha', \bar{l}', \bar{j}', i'], [1, [(l_e', \mu_e')], [1], i' - 1], \text{private } bty'*) = (\sigma_2', 1)$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, (l_e, \mu_e))$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, (l_e', \mu_e'))$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $(l_e, \mu_e) = (l_e', \mu_e')$, and $c_1 \simeq_L c_1'$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL(Freeable, private } bty*, \text{private, } \alpha))$, $\sigma_1'(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL(Freeable, private } bty'*, \text{private, } \alpha'))$, $\sigma_1 = \sigma_1'$, and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given $\text{DecodePtr(private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$, and $\text{DecodePtr(private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', i']$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$. Therefore we have $i = i'$.

Given $\text{UpdatePrivPtr}(\sigma_1, [\alpha, \bar{l}, \bar{j}, i], [1, [(l_e, \mu_e)], [1], i - 1], \text{private } bty*) = (\sigma_2, 1)$, $\text{UpdatePrivPtr}(\sigma_1', [\alpha', \bar{l}', \bar{j}', i'], [1, [(l_e', \mu_e')], [1], i' - 1], \text{private } bty'*) = (\sigma_2', 1)$, $\sigma_1 = \sigma_1'$, $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$, $(l_e, \mu_e) = (l_e', \mu_e')$, $bty = bty'$, and $i = i'$, by Lemma 3.3.16 we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow^*_{wdp2} (\gamma, \sigma_2, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp2} (\gamma, \sigma_2, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp3} (\gamma, \sigma_2, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp3} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Private Pointer Dereference Write Private Value, we have $\gamma(x) = (l, \text{private } bty*)$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, v)$, $v \neq \text{skip}$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL(Freeable, private } bty*, \text{private, } \alpha))$, $\text{Label}(e, \gamma) = \text{private}$, $(bty = \text{int}) \vee (bty = \text{float})$, $\text{DecodePtr(private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, $\text{acc} = 0$, and $\text{UpdatePriv}(\sigma_1, \alpha, \bar{l}, \bar{j}, \text{private } bty, v) = (\sigma_2, 1)$.

By definition 3.3.1, given $c = wdp3$, we have $c \simeq_L c'$ if and only if $c' = wdp3$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp3} (\gamma, \sigma'_2, \text{acc}, \text{skip})$ by rule Private Pointer Dereference Write Private Value, we have $\gamma(x) = (l', \text{private } bty'*)$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, v')$, $v' \neq \text{skip}$, $\sigma'_1(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, \alpha'))$, $\text{Label}(e, \gamma) = \text{private}$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', 1]$, $\text{acc} = 0$, and $\text{UpdatePriv}(\sigma'_1, \alpha', \bar{l}', \bar{j}', \text{private } bty', v') = (\sigma'_2, 1)$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, v)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, v')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $v = v'$, and $c_1 \simeq_L c'_1$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\sigma'_1(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, \alpha'))$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', 1]$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \bar{l}, \bar{j}, 1] = [\alpha', \bar{l}', \bar{j}', 1]$. Therefore, we have $\bar{l} = \bar{l}'$ and $\bar{j} = \bar{j}'$.

Given $\text{UpdatePriv}(\sigma_1, \alpha, \bar{l}, \bar{j}, \text{private } bty, v) = (\sigma_2, 1)$, $\text{UpdatePriv}(\sigma'_1, \alpha', \bar{l}', \bar{j}', \text{private } bty', v') = (\sigma'_2, 1)$, $\sigma_1 = \sigma'_1$, $\alpha = \alpha'$, $\bar{l} = \bar{l}'$, $\bar{j} = \bar{j}'$, $bty = bty'$ and $v = v'$, by Lemma 3.3.14 we have $\sigma_2 = \sigma'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow^*_{wdp3} (\gamma, \sigma_2, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp3} (\gamma, \sigma_2, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp4} (\gamma, \sigma_2, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp4} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Private Pointer Dereference Write Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, v)$, $\text{acc} = 0$, $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, and $\text{UpdatePriv}(\sigma_1, \alpha, \bar{l}, \bar{j}, \text{private } bty, \text{encrypt}(v)) = (\sigma_2, 1)$.

By definition 3.3.1, given $c = wdp4$, we have $c \simeq_L c'$ if and only if $c' = wdp4$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp4} (\gamma, \sigma_2', \text{acc}, \text{skip})$ by rule Private Pointer Dereference Write Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, v')$, $\text{acc} = 0$, $\gamma(x) = (l', \text{private } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma_1(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable, private } bty'*, \text{private}, \alpha'))$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', 1]$, and $\text{UpdatePriv}(\sigma_1', \alpha', \bar{l}', \bar{j}', \text{private } bty', \text{encrypt}(v')) = (\sigma_2', 1)$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, v)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, v')$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $v = v'$, and $c_1 \simeq_L c_1'$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable, private } bty*, \text{private}, \alpha))$, $\sigma_1(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable, private } bty'*, \text{private}, \alpha'))$, $\sigma_1 = \sigma_1'$, and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', 1]$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \bar{l}, \bar{j}, 1] = [\alpha', \bar{l}', \bar{j}', 1]$. Therefore, we have $\bar{l} = \bar{l}'$ and $\bar{j} = \bar{j}'$.

Given $\text{UpdatePriv}(\sigma_1, \alpha, \bar{l}, \bar{j}, \text{private } bty, \text{encrypt}(v)) = (\sigma_2, 1)$, $\text{UpdatePriv}(\sigma_1', \alpha', \bar{l}', \bar{j}', \text{private } bty', \text{encrypt}(v')) = (\sigma_2', 1)$, $\sigma_1 = \sigma_1'$, $\alpha = \alpha'$, $\bar{l} = \bar{l}'$, $\bar{j} = \bar{j}'$, $bty = bty'$, and $v = v'$, by Axiom 3.3.1 we have $\text{encrypt}(v) = \text{encrypt}(v')$, and therefore by Lemma 3.3.14 we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow^*_{wdp4} (\gamma, \sigma_2, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp4} (\gamma, \sigma_2, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp5} (\gamma, \sigma_2, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp5} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Private Pointer Dereference Write Higher Level Indirection Multiple Locations, we have $\gamma(x) = (l, \text{private } bty*)$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, [\alpha_e, \bar{l}_e, \bar{j}_e, i-1])$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable, private } bty*, \text{private}, \alpha))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{acc} = 0$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$, $i > 1$, and $\text{UpdatePrivPtr}(\sigma_1, [\alpha, \bar{l}, \bar{j}, i], [\alpha_e, \bar{l}_e, \bar{j}_e, i-1], \text{private } bty*) = (\sigma_2, 1)$.

By definition 3.3.1, given $c = wdp5$, we have $c \simeq_L c'$ if and only if $c' = wdp5$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp5} (\gamma, \sigma_2', \text{acc}, \text{skip})$ by rule Private Pointer Dereference Write Higher Level Indirection Multiple Locations, we have $\gamma(x) = (l', \text{private } bty'*)$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, [\alpha_e', \bar{l}_e', \bar{j}_e', i' - 1])$, $\sigma_1'(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL(Freeable, private } bty'*, \text{private}, \alpha'))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{acc} = 0$, $\text{DecodePtr(private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', i']$, $i' > 1$, and $\text{UpdatePrivPtr}(\sigma_1', [\alpha', \bar{l}', \bar{j}', i'], [\alpha_e', \bar{l}_e', \bar{j}_e', i' - 1], \text{private } bty'*) = (\sigma_2', 1)$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, [\alpha_e, \bar{l}_e, \bar{j}_e, i - 1])$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \text{acc}, [\alpha_e', \bar{l}_e', \bar{j}_e', i' - 1])$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $[\alpha_e, \bar{l}_e, \bar{j}_e, i - 1] = [\alpha_e', \bar{l}_e', \bar{j}_e', i' - 1]$, and $c_1 \simeq_L c_1'$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL(Freeable, private } bty*, \text{private}, \alpha))$, $\sigma_1'(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL(Freeable, private } bty'*, \text{private}, \alpha'))$, $\sigma_1 = \sigma_1'$, and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given $\text{DecodePtr(private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$, $\text{DecodePtr(private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', i']$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$.

Given $\text{UpdatePrivPtr}(\sigma_1, [\alpha, \bar{l}, \bar{j}, i], [\alpha_e, \bar{l}_e, \bar{j}_e, i - 1], \text{private } bty*) = (\sigma_2, 1)$, $\text{UpdatePrivPtr}(\sigma_1', [\alpha', \bar{l}', \bar{j}', i'], [\alpha_e', \bar{l}_e', \bar{j}_e', i' - 1], \text{private } bty'*) = (\sigma_2', 1)$, $\sigma_1 = \sigma_1'$, $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$, $bty = bty'$, and $[\alpha_e, \bar{l}_e, \bar{j}_e, i - 1] = [\alpha_e', \bar{l}_e', \bar{j}_e', i' - 1]$, by Lemma 3.3.16 we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp5}^{*} (\gamma, \sigma_2, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, *x = e) \Downarrow_{wdp5} (\gamma, \sigma_2, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin} (\gamma, \sigma_1, \text{acc}, v_1)$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin} (\gamma, \sigma_1, \text{acc}, v_1)$ by rule Pre-Increment Public Variable, we have $\text{acc} = 0$, $\gamma(x) = (l, \text{public } bty)$, $\sigma(l) = (\omega, \text{public } bty, 1, \text{PermL(Freeable, public } bty, \text{public}, 1))$, $\text{DecodeVal(public } bty,$

$1, \omega) = v$, $v_1 =_{\text{public}} v +_{\text{public}} 1$, and $\text{UpdateVal}(\sigma, l, v_1, \text{public } bty) = \sigma_1$.

By definition 3.3.1, given $c = pin$, we have $c \simeq_L c'$ if and only if $c' = pin$.

Given $\Sigma \triangleright (\gamma,\ \sigma,\ \text{acc},\ ++\ x) \Downarrow_{pin} (\gamma,\ \sigma_1',\ \text{acc},\ v_1')$ by rule Pre-Increment Public Variable, we have $\text{acc} = 0$, $\gamma(x) = (l', \text{public } bty')$, $\sigma(l') = (\omega', \text{public } bty', 1, \text{PermL}(\text{Freeable, public } bty', \text{public, } 1))$, $\text{DecodeVal}(\text{public } bty', 1, \omega') = v'$, $v_1' =_{\text{public}} v' +_{\text{public}} 1$, and $\text{UpdateVal}(\sigma, l', v_1', \text{public } bty') = \sigma_1'$.

Given $\gamma(x) = (l, \text{public } bty)$ and $\gamma(x) = (l', \text{public } bty')$ we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1))$, $\sigma(l') = (\omega', \text{public } bty', 1, \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodeVal}(\text{public } bty,\ 1,\ \omega) = v$, $\text{DecodeVal}(\text{public } bty',\ 1,\ \omega') = v'$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.4 we have $v = v'$.

Given $v_1 =_{\text{public}} v +_{\text{public}} 1$, $v_1' =_{\text{public}} v' +_{\text{public}} 1$, and $v = v'$, we have $v_1 = v_1'$.

Given $\text{UpdateVal}(\sigma, l, v_1, \text{public } bty) = \sigma_1$, $\text{UpdateVal}(\sigma, l', v_1', \text{public } bty') = \sigma_1'$, $l = l'$, $bty = bty$, and $v_1 = v_1'$, by Lemma 3.3.12 we have $\sigma_1 = \sigma_1'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma_1'$, $\text{acc} = \text{acc}$, $v_1 = v_1'$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ ++\ x) \Downarrow_{pin1} (\gamma,\ \sigma_1,\ \text{acc},\ v_1)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \text{acc},\ ++\ x) \Downarrow_{pin1} (\gamma,\ \sigma_1,\ \text{acc},\ v_1)$ by rule Pre-Increment Private Variable, we have $\text{acc} = 0$, $\gamma(x) = (l, \text{private } bty)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega,\ \text{private } bty, 1,\ \text{PermL}(\text{Freeable, private } bty,\ \text{private, } 1))$, $\text{DecodeVal}(\text{private } bty, 1, \omega) = v$, $v_1 =_{\text{private}} v +_{\text{private}} \text{encrypt}(1)$, and $\text{UpdateVal}(\sigma, l, v_1, \text{private } bty) = \sigma_1$.

By definition 3.3.1, given $c = pin1$, we have $c \simeq_L c'$ if and only if $c' = pin1$.

Given $\Sigma \triangleright (\gamma,\ \sigma,\ \text{acc},\ ++\ x) \Downarrow_{pin1} (\gamma,\ \sigma_1',\ \text{acc},\ v_1')$ by rule Pre-Increment Private Variable, we have $\text{acc} = 0$,

$\gamma(x) = (l', \text{private } bty')$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma(l') = (\omega', \text{private } bty', 1, \text{PermL}(\text{Freeable, private } bty', \text{private}, 1))$, $\text{DecodeVal}(\text{private } bty', 1, \omega') = v'$, $v'_1 =_{\text{private}} v' +_{\text{private}} \text{encrypt}(1)$, and $\text{UpdateVal}(\sigma, l', v'_1, \text{private } bty') = \sigma'_1$.

Given $\gamma(x) = (l, \text{private } bty)$ and $\gamma(x) = (l', \text{private } bty')$ we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable, private } bty, \text{private}, 1))$, $\sigma(l') = (\omega', \text{private } bty', 1, \text{PermL}(\text{Freeable, private } bty', \text{private}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodeVal}(\text{private } bty, 1, \omega) = v$, $\text{DecodeVal}(\text{private } bty', 1, \omega) = v'$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.4 we have $v = v'$.

Given $v_1 =_{\text{private}} v +_{\text{private}} \text{encrypt}(1)$, $v'_1 =_{\text{private}} v' +_{\text{private}} \text{encrypt}(1)$, and $v = v'$, by Axiom 3.3.2 we have $v_1 = v'_1$.

Given $\text{UpdateVal}(\sigma, l, v_1, \text{private } bty) = \sigma_1$, $\text{UpdateVal}(\sigma, l, v_1, \text{private } bty') = \sigma_1$, $l = l'$, $bty = bty'$, and $v_1 = v'_1$, by Lemma 3.3.12 we have $\sigma_1 = \sigma'_1$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\text{acc} = \text{acc}$, $v_1 = v'_1$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$ by rule Pre-Increment Public Pointer Single Location, we have $\text{acc} = 0$, $\gamma(x) = (l, \text{public } bty*)$, $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable, public } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, $\text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty), \sigma) = ((l_2, \mu_2), 1)$, and $\text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \text{public } bty*) = (\sigma_1, 1)$.

By definition 3.3.1, given $c = pin2$, we have $c \simeq_L c'$ if and only if $c' = pin2$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin2} (\gamma, \sigma'_1, \text{acc}, (l'_2, \mu'_2))$ by rule Pre-Increment Public Pointer Single Location, we have $\text{acc} = 0$, $\gamma(x) = (l', \text{public } bty'*)$, $\sigma(l') = (\omega', \text{public } bty'*, 1, \text{PermL}(\text{Freeable, public } bty'*, \text{public}, 1))$, $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l'_1, \mu'_1)], [1], 1]$, $\text{GetLocation}((l'_1, \mu'_1), \tau(\text{public } bty'), \sigma) = ((l'_2, \mu'_2), 1)$, and $\text{UpdatePtr}(\sigma, (l', 0), [1, [(l'_2, \mu'_2)], [1], 1], \text{public } bty'*) = (\sigma'_1, 1)$.

Given $\gamma(x) = (l, \text{public } bty*)$ and $\gamma(x) = (l', \text{public } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\sigma(l') = (\omega', \text{public } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l'_1, \mu'_1)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, \mu_1)], [1], 1] = [1, [(l'_1, \mu'_1)], [1], 1]$. Therefore, we have $(l_1, \mu_1) = (l'_1, \mu'_1)$.

Given $((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty), \sigma)$, $((l'_2, \mu'_2), 1) = \text{GetLocation}((l'_1, \mu'_1), \tau(\text{public } bty'), \sigma)$, $(l_1, \mu_1) = (l'_1, \mu'_1)$, and $bty = bty'$, by Lemma 3.3.21 we have $(l_2, \mu_2) = (l'_2, \mu'_2)$.

Given $\text{UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \text{public } bty*) = (\sigma_1, 1)$, $\text{UpdatePtr}(\sigma, (l', 0), [1, [(l'_2, \mu'_2)], [1], 1], \text{public } bty'*) = (\sigma'_1, 1)$, $l = l'$, $bty = bty$, and $(l_2, \mu_2) = (l'_2, \mu'_2)$, by Lemma 3.3.15 we have $\sigma_1 = \sigma'_1$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\text{acc} = \text{acc}$, $(l_2, \mu_2) = (l'_2, \mu'_2)$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\, x) \Downarrow^*_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\, x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\, x) \Downarrow_{pin6} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\, x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\, x) \Downarrow^*_{pin6} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\, x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\, x) \Downarrow_{pin3} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\, x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\, x) \Downarrow^*_{pin3} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\, x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin7} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow^*_{pin7} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin2} (\gamma, \sigma_1, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin4} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1])$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin4} (\gamma, \sigma_1, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1])$ by rule Pre-Increment Private Pointer Multiple Locations, we have $\text{acc} = 0$, $\gamma(x) = (l, \text{private } bty*)$, $\sigma(l) = (\omega, \text{ private } bty*, \alpha, \text{PermL(Freeable, private } bty*, \text{ private, } \alpha))$, $\text{DecodePtr(private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, $\text{IncrementList}(\bar{l}, \tau(\text{private } bty), \sigma) = (\bar{l}', 1)$, and $\text{UpdatePtr}(\sigma, (l, 0), [\alpha, \bar{l}', \bar{j}, 1], \text{private } bty*) = (\sigma_1, 1)$.

By definition 3.3.1, given $c = pin4$, we have $c \simeq_L c'$ if and only if $c' = pin4$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, ++\ x) \Downarrow_{pin4} (\gamma, \sigma_1', \text{acc}, [\alpha', \bar{l}''', \bar{j}', 1])$ by rule Pre-Increment Private Pointer Multiple Locations, we have $\text{acc} = 0$, $\gamma(x) = (l', \text{ private } bty'*)$, $\sigma(l') = (\omega', \text{ private } bty'*, \alpha, \text{PermL(Freeable, private } bty*, \text{ private, } \alpha'))$, $\text{DecodePtr(private } bty'*, \alpha', \omega') = [\alpha', \bar{l}'', \bar{j}', 1]$, $\text{IncrementList}(\bar{l}'', \tau(\text{private } bty'), \sigma) = (\bar{l}''', 1)$, $\text{CheckOvershooting}_M(\text{private } bty', \bar{l}''', \bar{j}', \sigma) = 1$, and $\text{UpdatePtr}(\sigma, (l', 0), [\alpha', \bar{l}''', \bar{j}', 1], \text{private } bty'*) = (\sigma_1', 1)$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{ private } bty*, \alpha, \text{PermL(Freeable, private } bty*, \text{ private, } \alpha))$, $\sigma(l') = (\omega', \text{ private } bty'*, \alpha, \text{PermL(Freeable, private } bty*, \text{ private, } \alpha'))$, $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given $\text{DecodePtr(private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, $\text{DecodePtr(private } bty'*, \alpha', \omega') = [\alpha', \bar{l}'', \bar{j}', 1]$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \bar{l}, \bar{j}, 1] = [\alpha', \bar{l}'', \bar{j}', 1]$. Therefore, we have $\bar{l} = \bar{l}''$ and $\bar{j} = \bar{j}'$.

Given $\text{IncrementList}(\bar{l}, \tau(\text{private } bty), \sigma) = (\bar{l}', 1)$, $\text{IncrementList}(\bar{l}'', \tau(\text{private } bty'), \sigma) = (\bar{l}''', 1)$, $\bar{l} = \bar{l}''$, and $bty = bty'$, by Lemma 3.3.22 we have $\tau(\text{private } bty) = \tau(\text{private } bty')$, and by Lemma 3.3.23 we have $\bar{l}' = \bar{l}'''$.

Given $\mathrm{UpdatePtr}(\sigma, (l, 0), [\alpha, \bar{l}', \bar{j}, 1], \text{private } bty*) = (\sigma_1, 1)$, $\mathrm{UpdatePtr}(\sigma, (l', 0), [\alpha', \bar{l}''', \bar{j}', 1], \text{private } bty'*) = (\sigma'_1, 1)$, $l = l'$, $bty = bty'$, and $[\alpha, \bar{l}, \bar{j}, 1] = [\alpha', \bar{l}'', \bar{j}', 1]$, by Lemma 3.3.15 we have $\sigma_1 = \sigma'_1$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\mathrm{acc} = \mathrm{acc}$, $[\alpha, \bar{l}', \bar{j}, 1] = [\alpha', \bar{l}''', \bar{j}', 1]$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, ++\ x) \Downarrow^*_{pin4} (\gamma, \sigma_1, \mathrm{acc}, [\alpha, \bar{l}', \bar{j}, 1])$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, ++\ x) \Downarrow_{pin4} (\gamma, \sigma_1, \mathrm{acc}, [\alpha, \bar{l}', \bar{j}, 1])$.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, ++\ x) \Downarrow_{pin5} (\gamma, \sigma_1, \mathrm{acc}, [\alpha, \bar{l}', \bar{j}, i])$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, ++\ x) \Downarrow_{pin4} (\gamma, \sigma_1, \mathrm{acc}, [\alpha, \bar{l}', \bar{j}, 1])$.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, ++\ x) \Downarrow_{pin5} (\gamma, \sigma_1, \mathrm{acc}, [\alpha, \bar{l}', \bar{j}, i])$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, ++\ x) \Downarrow_{pin4} (\gamma, \sigma_1, \mathrm{acc}, [\alpha, \bar{l}', \bar{j}, 1])$.

Case $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, ty\ x[e]) \Downarrow_{da} (\gamma_1, \sigma_3, \mathrm{acc}, \mathrm{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, ty\ x[e]) \Downarrow_{da} (\gamma_1, \sigma_3, \mathrm{acc}, \mathrm{skip})$ by rule Public 1 Dimension Array Declaration, we have $((ty = \text{public } bty) \wedge ((bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{int}))) \vee (ty = \text{char})$, $l = \phi()$, $\mathrm{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \mathrm{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \mathrm{acc}, n)$, $\gamma_1 = \gamma[x \rightarrow (l, \text{public const } bty*)]$, $l_1 = \phi()$, $\omega = \mathrm{EncodePtr}(\text{public const } bty*, [1, [(l_1, 0)], [1], 1])$, $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{public const } bty*, 1, \mathrm{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))]$, $\mathrm{acc} = 0$, $n > 0$, $\omega_1 = \mathrm{EncodeVal}(\text{private } bty, \text{NULL})$, and $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{public } bty, n, \mathrm{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))]$.

By definition 3.3.1, given $c = da$, we have $c \simeq_L c'$ if and only if $c' = da$.

Given $\Sigma \triangleright (\gamma, \sigma, \mathrm{acc}, ty\ x[e]) \Downarrow_{da} (\gamma'_1, \sigma'_3, \mathrm{acc}, \mathrm{skip})$ by rule Public 1 Dimension Array Declaration, we have $((ty = \text{public } bty') \wedge ((bty' = \text{float}) \vee (bty' = \text{char}) \vee (bty' = \text{int}))) \vee (ty = \text{char})$, $l' = \phi()$, $\mathrm{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \mathrm{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \mathrm{acc}, n')$, $\gamma'_1 = \gamma[x \rightarrow (l', \text{public const } bty'*)]$, $l'_1 = \phi()$, $\omega' = \mathrm{EncodePtr}(\text{public const } bty'*, [1, [(l'_1, 0)], [1], 1])$, $\sigma'_2 = \sigma'_1[l \rightarrow (\omega', \text{public const } bty'*, 1, \mathrm{PermL}(\text{Freeable}, \text{public const } bty'*, \text{public}, 1))]$, $\mathrm{acc} = 0$, $n' > 0$, $\omega'_1 = \mathrm{EncodeVal}(\text{private } bty', \text{NULL})$, and $\sigma'_3 = \sigma'_2[l'_1 \rightarrow (\omega'_1, \text{public } bty', n', \mathrm{PermL}(\text{Freeable}, \text{public } bty', \text{public}, n'))]$.

Given $(ty = \text{public } bty)$ and $(ty = \text{public } bty')$, we have $bty = bty'$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, n)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{public const } bty*)]$, $\gamma'_1 = \gamma[x \rightarrow (l', \text{public const } bty'*)]$, $l = l'$, $bty = bty'$, we have $\gamma_1 = \gamma'_1$.

Given $l_1 = \phi()$ and $l'_1 = \phi()$, by Axiom 3.3.5 we have $l_1 = l'_1$.

Given $\omega = \text{EncodePtr}(\text{public const } bty*, [1, [(l_1, 0)], [1], 1])$, $\omega' = \text{EncodePtr}(\text{public const } bty'*, [1, [(l'_1, 0)], [1], 1])$, $bty = bty'$, and $l_1 = l'_1$, by Lemma 3.3.5 we have $\omega = \omega'$.

Given $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable, public const } bty*, \text{public}, 1))]$, $\sigma'_2 = \sigma'_1[l \rightarrow (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable, public const } bty'*, \text{public}, 1))]$, $\sigma_1 = \sigma'_1$, $l = l'$, $\omega = \omega'$, and $bty = bty'$, we have $\sigma_2 = \sigma'_2$.

Given $\omega_1 = \text{EncodeVal}(\text{private } bty, \text{NULL})$, $\omega'_1 = \text{EncodeVal}(\text{private } bty', \text{NULL})$, and $bty = bty'$, by Lemma 3.3.3 we have $\omega_1 = \omega'_1$.

Given $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable, public } bty, \text{public}, n))]$, $\sigma'_3 = \sigma'_2[l'_1 \rightarrow (\omega'_1, \text{public } bty', n', \text{PermL}(\text{Freeable, public } bty', \text{public}, n'))]$, $\sigma_2 = \sigma'_2$, $l_1 = l'_1$, $\omega_1 = \omega'_1$, $n = n'$, and $bty = bty'$, we have $\sigma_3 = \sigma'_3$.

Therefore, we have $\gamma_1 = \gamma'_1$, $\sigma_3 = \sigma'_3$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x[e]) \Downarrow_{da1} (\gamma_1, \sigma_3, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, ty\ x[e]) \Downarrow_{da} (\gamma_1, \sigma_3, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra} (\gamma, \sigma_1, \text{acc}, v_i)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra} (\gamma, \sigma_1, \text{acc}, v_i)$ by rule Public 1D Array Read Public Index, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable, public const } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable, public } bty, \text{public}, n))$, $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, and $0 \leq i \leq n-1$.

By definition 3.3.1, given $c = ra$, we have $c \simeq_L c'$ if and only if $c' = ra$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra} (\gamma, \sigma'_1, \text{acc}, v'_i)$ by rule Public 1D Array Read Public Index, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, $\gamma(x) = (l', \text{public const } bty'*)$, $\sigma'_1(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable, public const } bty'*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_1(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL}(\text{Freeable, public } bty', \text{public}, n'))$, $\text{DecodeVal}(\text{public } bty', n', \omega'_1) = [v'_0, ..., v'_{n'-1}]$, and $0 \leq i' \leq n'-1$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $\gamma(x) = (l, \text{public const } bty*)$ and $\gamma(x) = (l', \text{public const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable, public const } bty*, \text{public}, 1))$, $\sigma'_1(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable, public const } bty'*, \text{public}, 1))$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr}(\text{public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we have $l_1 = l'_1$.

Given $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable, public } bty, \text{public}, n))$, $\sigma'_1(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL}(\text{Freeable, public } bty', \text{public}, n'))$, $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{DecodeVal}(\text{public } bty', n', \omega'_1) = [v'_0, ..., v'_{n'-1}]$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v'_0, ..., v'_{n'-1}]$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v'_m$.

Given $0 \leq i \leq n-1$, $0 \leq i' \leq n'-1$, $i = i'$, $n = n'$, and $\forall m \in \{0, ..., n-1\}$, $v_m = v'_m$, we have $v_i = v'_i$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\text{acc} = \text{acc}$, $v_i = v'_i$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra3} (\gamma, \sigma_1, \text{acc}, v_i)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra} (\gamma, \sigma_1, \text{acc}, v_i)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra1} (\gamma, \sigma_1, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra1} (\gamma, \sigma_1, \text{acc}, v)$ by rule Private 1D Array Read Private Index we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$, $\gamma(x) = (l, \text{private const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{private}, 1))$, $\text{DecodePtr(private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL(Freeable, private } bty, \text{private}, n))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{DecodeVal(private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, and $v = \bigvee_{m=0}^{n-1}(i = \text{encrypt}(m)) \wedge v_m$.

By definition 3.3.1, given $c = ra1$, we have $c \simeq_L c'$ if and only if $c' = ra1$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra1} (\gamma, \sigma'_1, \text{acc}, v')$ by rule Private 1D Array Read Private Index we have $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, $\gamma(x) = (l', \text{private const } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma'_1(l') = (\omega', \text{private const } bty'*, 1, \text{PermL(Freeable, private const } bty'*, \text{private}, 1))$, $\text{DecodePtr(private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_1(l'_1) = (\omega'_1, \text{private } bty', n', \text{PermL(Freeable, private } bty', \text{private}, n'))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{DecodeVal(private } bty', n', \omega'_1) = [v'_0, ..., v'_{n'-1}]$, and $v' = \bigvee_{m'=0}^{n'-1}(i' = \text{encrypt}(m')) \wedge v'_m$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $\sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{private}, 1))$, $\sigma'_1(l') = (\omega', \text{private const } bty'*, 1, \text{PermL(Freeable, private const } bty'*, \text{private}, 1))$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we have $l_1 = l'_1$.

Given $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable, private } bty, \text{private}, n))$, $\sigma'_1(l'_1) = (\omega'_1, \text{private } bty', n', \text{PermL}(\text{Freeable, private } bty', \text{private}, n'))$, $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{DecodeVal}(\text{private } bty', n', \omega'_1) = [v'_0, ..., v'_{n'-1}]$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v'_0, ..., v'_{n'-1}]$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v'_m$.

Given $v = \bigvee_{m=0}^{n-1}(i = \text{encrypt}(m)) \wedge v_m$ and $v' = \bigvee_{m'=0}^{n'-1}(i' = \text{encrypt}(m')) \wedge v'_m$, we have $m \in \{0, ..., n-1\}$ and $m' \in \{0, ..., n'-1\}$. Given $n = n'$, we have $m, m' \in \{0, ..., n-1\}$ and $m = m'$. Given $m = m'$, we have $\text{encrypt}(m) = \text{encrypt}(m')$. Given $\forall m \in \{0, ..., n-1\}$, $v_m = v'_m$ and $i = i'$, we have $v = v'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\text{acc} = \text{acc}$, $v = v'$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra2} (\gamma, \sigma_1, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra2} (\gamma, \sigma_1, \text{acc}, v)$ by rule Public 1D Array Read Private Index, we have $\gamma(x) = (l, \text{public const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$, $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable, public const } bty*, \text{public}, 1))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable, public } bty, \text{public}, n))$, $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, and $v = \bigvee_{m=0}^{n-1}(i = \text{encrypt}(m)) \wedge \text{encrypt}(v_m)$.

By definition 3.3.1, given $c = ra2$, we have $c \simeq_L c'$ if and only if $c' = ra2$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{ra2} (\gamma, \sigma'_1, \text{acc}, v')$ by rule Public 1D Array Read Private Index, we have $\gamma(x) = (l', \text{public const } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, $\sigma'_1(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable, public const } bty'*, \text{public}, 1))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{DecodePtr}(\text{public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_1(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL}(\text{Freeable, public } bty', \text{public}, n))$, $\text{DecodeVal}(\text{public } bty', n', \omega'_1) = [v'_0, ..., v'_{n'-1}]$, and $v' = \bigvee_{m'=0}^{n'-1}(i' = \text{encrypt}(m')) \wedge \text{encrypt}(v'_{m'})$.

Given $\gamma(x) = (l, \text{public const } bty*)$ and $\gamma(x) = (l', \text{public const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL(Freeable, public const } bty*, \text{public}, 1))$, $\sigma'_1(l') = (\omega', \text{public const } bty'*, 1, \text{PermL(Freeable, public const } bty'*, \text{public}, 1))$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr(public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr(public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we have $l_1 = l'_1$.

Given $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL(Freeable, public } bty, \text{public}, n))$, $\sigma'_1(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL(Freeable, public } bty', \text{public}, n))$, $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodeVal(public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{DecodeVal(public } bty', n', \omega'_1) = [v'_0, ..., v'_{n'-1}]$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v'_0, ..., v'_{n'-1}]$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v'_m$.

Given $v = \bigvee_{m=0}^{n-1}(i = \text{encrypt}(m)) \wedge \text{encrypt}(v_m)$, $v' = \bigvee_{m'=0}^{n'-1}(i' = \text{encrypt}(m')) \wedge \text{encrypt}(v'_{m'})$, $i = i'$, $n = n'$, and $\forall m \in \{0, ..., n-1\}$, $v_m = v'_m$, we have $m, m' \in \{0, ..., n-1\}$. By Axiom 3.3.1 we have $\forall m \in \{0, ..., n-1\}$, $\text{encrypt}(m) = \text{encrypt}(m')$ and $\text{encrypt}(v_m) = \text{encrypt}(v'_{m'})$. Therefore, we have $v = v'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\text{acc} = \text{acc}$, $v = v'$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa} (\gamma, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by rule Public 1D Array Write Public Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_2(l) = (\omega, \text{public const } bty*, 1, \text{PermL(Freeable, public const } bty*, \text{public}, 1))$, $\text{DecodePtr(public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{public } bty, n, \text{PermL(Freeable, public } bty, \text{public}, n))$, $\text{DecodeVal(public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{v}{v_i}\right)$, $0 \leq i \leq n-1$, and $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \text{public } bty) = \sigma_3$.

By definition 3.3.1, given $c = wa$, we have $c \simeq_L c'$ if and only if $c' = wa$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa} (\gamma, \sigma'_3, \text{acc}, \text{skip})$ by rule Public 1D Array Write Public Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, v'')$, $v'' \neq \text{skip}$, $\gamma(x) = (l', \text{public const } bty'*)$, $\sigma'_2(l') = (\omega', \text{public const } bty'*, 1, \text{PermL(Freeable, public const } bty'*, \text{public}, 1))$, $\text{DecodePtr(public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_2(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL(Freeable, public } bty', \text{public}, n'))$, $\text{DecodeVal(public } bty', n', \omega'_1) = [v''_0, ..., v''_{n'-1}]$, $[v'''_0, ..., v'''_{n'-1}] = [v''_0, ..., v''_{n-1}]\left(\frac{v''}{v''_{i'}}\right)$, $0 \leq i' \leq n' - 1$, and $\text{UpdateVal}(\sigma'_2, l'_1, [v'''_0, ..., v'''_{n'-1}], \text{public } bty') = \sigma'_3$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, v'')$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $v = v''$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x) = (l, \text{public const } bty*)$ and $\gamma(x) = (l', \text{public const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega, \text{public const } bty*, 1, \text{PermL(Freeable, public const } bty*, \text{public}, 1))$, $\sigma'_2(l') = (\omega', \text{public const } bty'*, 1, \text{PermL(Freeable, public const } bty'*, \text{public}, 1))$, $\sigma_2 = \sigma'_2$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr(public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr(public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we have $l_1 = l'_1$.

Given $\sigma_2(l_1) = (\omega_1, \text{public } bty, n, \text{PermL(Freeable, public } bty, \text{public}, n))$, $\sigma'_2(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL(Freeable, public } bty', \text{public}, n'))$, $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodeVal(public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{DecodeVal(public } bty', n', \omega'_1) = [v''_0, ..., v''_{n'-1}]$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v''_0, ..., v''_{n'-1}]$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$.

Given $0 \leq i \leq n - 1$, $0 \leq i' \leq n' - 1$, $i = i'$, $n = n'$, and $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$, we have $v_i = v''_{i'}$.

Given $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}](\frac{v}{v_i})$, $[v'''_0, ..., v'''_{n'-1}] = [v''_0, ..., v''_{n'-1}](\frac{v''}{v''_{i'}})$, $[v_0, ..., v_{n-1}] = [v''_0, ..., v''_{n'-1}]$, $v = v''$, and $v_i = v''_{i'}$, we have $[v'_0, ..., v'_{n-1}] = [v'''_0, ..., v'''_{n'-1}]$.

Given $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \text{public } bty) = \sigma_3$, $\text{UpdateVal}(\sigma'_2, l'_1, [v'''_0, ..., v'''_{n'-1}], \text{public } bty') = \sigma'_3$, $\sigma_2 = \sigma'_2$, $l_1 = l'_1$, $bty = bty'$, and $[v'_0, ..., v'_{n-1}] = [v'''_0, ..., v'''_{n'-1}]$, by Lemma 3.3.12 we have $\sigma_3 = \sigma'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, acc = acc, skip = skip, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa4} (\gamma, \sigma_3, \text{acc}, \text{skip})$

This case is similar to case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa} (\gamma, \sigma_3, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa1} (\gamma, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa1} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by rule Private 1D Array Write Public Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable, private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable, private } bty, \text{private}, n))$, $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}](\frac{\text{encrypt}(v)}{v_i})$, $0 \leq i \leq n-1$, and $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \text{private } bty) = \sigma_3$.

By definition 3.3.1, given $c = wa1$, we have $c \simeq_L c'$ if and only if $c' = wa1$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa1} (\gamma, \sigma'_3, \text{acc}, \text{skip})$ by rule Private 1D Array Write Public Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, v'')$, $v'' \neq \text{skip}$, $\gamma(x) = (l', \text{private const } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable, private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_2(l'_1) = (\omega'_1, \text{private } bty, n', \text{PermL}(\text{Freeable, private } bty', \text{private}, n'))$, $\text{DecodeVal}(\text{private } bty', n', \omega'_1) = [v''_0, ..., v''_{n'-1}]$, $[v'''_0, ..., v'''_{n'-1}] = [v''_0, ..., v''_{n'-1}](\frac{\text{encrypt}(v'')}{v''_{i'}})$, $0 \leq i' \leq n'-1$, and $\text{UpdateVal}(\sigma'_2, l'_1, [v'''_0, ..., v'''_{n'-1}], \text{private } bty') = \sigma'_3$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, v'')$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $v = v''$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{private, 1}))$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL(Freeable, private const } bty*, \text{private, 1}))$, $\sigma_2 = \sigma'_2$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr(private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr(private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we have $l_1 = l'_1$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL(Freeable, private } bty, \text{private, } n))$, $\sigma'_2(l'_1) = (\omega'_1, \text{private } bty, n', \text{PermL(Freeable, private } bty', \text{private, } n'))$, $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodeVal(private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{DecodeVal(private } bty', n', \omega'_1) = [v''_0, ..., v''_{n'-1}]$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v''_0, ..., v''_{n'-1}]$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$.

Given $0 \leq i \leq n-1$, $0 \leq i' \leq n'-1$, $i = i'$, $n = n'$, and $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$, we have $v_i = v''_{i'}$.

Given $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}](\frac{\text{encrypt}(v)}{v_i})$, $[v'''_0, ..., v'''_{n'-1}] = [v''_0, ..., v''_{n'-1}](\frac{\text{encrypt}(v'')}{v''_{i'}})$, $[v_0, ..., v_{n-1}] = [v''_0, ..., v''_{n'-1}]$, $v = v''$, and $v_i = v''_{i'}$, by Axiom 3.3.1 we have $\text{encrypt}(v) = \text{encrypt}(v'')$. Therefore, we have $[v'_0, ..., v'_{n-1}] = [v'''_0, ..., v'''_{n'-1}]$.

Given $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \text{private } bty) = \sigma_3$, $\text{UpdateVal}(\sigma'_2, l'_1, [v'''_0, ..., v'''_{n'-1}], \text{private } bty') = \sigma'_3$, $\sigma_2 = \sigma'_2$, $l_1 = l'_1$, $bty = bty'$, and $[v'_0, ..., v'_{n-1}] = [v'''_0, ..., v'''_{n'-1}]$, by Lemma 3.3.12 we have $\sigma_3 = \sigma'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa2} (\gamma, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa2} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by rule Private 1D Array Write Public Value Private Index, we have $\text{Label}(e_1, \gamma) = \text{private}$, $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $(bty = \text{int}) \vee (bty = \text{float})$, $v' = \text{encrypt}(v)$, $\forall v_m \in [v_0, ..., v_{n-1}].\ v'_m = ((i = \text{encrypt}(m)) \wedge v') \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, and $\text{UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \text{private } bty) = \sigma_3$.

By definition 3.3.1, given $c = wa2$, we have $c \simeq_L c'$ if and only if $c' = wa2$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa2} (\gamma, \sigma'_3, \text{acc}, \text{skip})$ by rule Private 1D Array Write Public Value Private Index, we have $\text{Label}(e_1, \gamma) = \text{private}$, $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v'')$, $v'' \neq \text{skip}$, $\gamma(x) = (l', \text{private const } bty'*)$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty'*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_2(l'_1) = (\omega'_1, \text{private } bty', n', \text{PermL}(\text{Freeable}, \text{private } bty', \text{private}, n'))$, $\text{DecodeVal}(\text{private } bty', n', \omega'_1) = [v''_0, ..., v''_{n'-1}]$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $v''' = \text{encrypt}(v'')$, $\forall v''_{m'} \in [v''_0, ..., v''_{n'-1}].\ v'''_{m'} = ((i' = \text{encrypt}(m')) \wedge v''') \vee (\neg(i' = \text{encrypt}(m')) \wedge v''_{m'})$, and $\text{UpdateVal}(\sigma'_2, l'_1, [v'''_0, ..., v'''_{n'-1}], \text{private } bty') = \sigma'_3$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, v'')$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $v = v''$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty'*, \text{private}, 1))$, $\sigma_2 = \sigma'_2$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we

have $l_1 = l'_1$.

Given $\sigma_2(l_1) = (\omega_1,$ private bty, n, PermL(Freeable, private bty, private, n)), $\sigma'_2(l'_1) = (\omega'_1,$ private bty', n', PermL(Freeable, private bty', private, n')), $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given DecodeVal(private $bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, DecodeVal(private $bty', n', \omega'_1) = [v''_0, ..., v''_{n'-1}]$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v''_0, ..., v''_{n'-1}]$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$.

Given $v' = \text{encrypt}(v)$, $v''' = \text{encrypt}(v'')$, and $v = v''$, by Axiom 3.3.1 we have $v' = v'''$.

Given $\forall v_m \in [v_0, ..., v_{n-1}].\, v'_m = ((i = \text{encrypt}(m)) \wedge v') \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, $\forall v''_{m'} \in [v''_0, ..., v''_{n'-1}].\, v'''_{m'} = ((i' = \text{encrypt}(m')) \wedge v''') \vee (\neg(i' = \text{encrypt}(m')) \wedge v''_{m'})$, $v' = v'''$, $i = i'$, $n = n'$, and $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$, we have $m, m' \in \{0, ..., n-1\}$ and by Axiom 3.3.1 we have $\text{encrypt}(m) = \text{encrypt}(m')$. Therefore, we have $[v'_0, ..., v'_{n-1}] = [v'''_0, ..., v'''_{n'-1}]$.

Given UpdateVal(σ_2, $l_1, [v'_0, ..., v'_{n-1}]$, private bty) $= \sigma_3$, UpdateVal(σ'_2, $l'_1, [v'''_0, ..., v'''_{n'-1}]$, private bty') $= \sigma'_3$, $\sigma_2 = \sigma'_2$, $l_1 = l'_1$, $bty = bty'$, and $[v'_0, ..., v'_{n-1}] = [v'''_0, ..., v'''_{n'-1}]$, by Lemma 3.3.12 we have $\sigma_3 = \sigma'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, acc = acc, skip = skip, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa3} (\gamma, \sigma_3, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa3} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by rule Private 1D Array Write Private Value Private Index, we have Label(e_1, γ) = Label(e_2, γ) = private, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v)$, $v \neq$ skip, $\gamma(x) = (l,$ private const $bty*)$, $\sigma_2(l) = (\omega,$ private const $bty*$, 1, PermL(Freeable, private const $bty*$, private, 1)), DecodePtr(private const $bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1,$ private $bty, n,$ PermL(Freeable, private bty, private, n)), DecodeVal(private $bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $(bty = \text{int}) \vee (bty = \text{float})$, $\forall v_m \in [v_0, ..., v_{n-1}].\, v'_m = ((i = \text{encrypt}(m)) \wedge v) \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, and UpdateVal($\sigma_2$, l_1, $[v'_0, ..., v'_{n-1}]$, private bty) $= \sigma_3$.

By definition 3.3.1, given $c = wa3$, we have $c \simeq_L c'$ if and only if $c' = wa3$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wa3} (\gamma, \sigma'_3, \text{acc}, \text{skip})$ by rule Private 1D Array Write Private Value Private Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, v'')$, $v'' \neq \text{skip}$, $\gamma(x) = (l', \text{private const } bty'*)$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty'*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_2(l'_1) = (\omega'_1, \text{private } bty', n', \text{PermL}(\text{Freeable}, \text{private } bty', \text{private}, n'))$, $\text{DecodeVal}(\text{private } bty', n', \omega'_1) = [v''_0, ..., v''_{n'-1}]$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\forall v''_{m'} \in [v''_0, ..., v''_{n'-1}].\ v'''_{m'} = ((i' = \text{encrypt}(m')) \wedge v'') \vee (\neg(i' = \text{encrypt}(m')) \wedge v''_{m'})$, and $\text{UpdateVal}(\sigma'_2, l'_1, [v'''_0, ..., v'''_{n'-1}], \text{private } bty') = \sigma'_3$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, v'')$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $v = v''$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega, \text{ private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty'*, \text{private}, 1))$, $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we have $l_1 = l'_1$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $\sigma'_2(l'_1) = (\omega'_1, \text{private } bty', n', \text{PermL}(\text{Freeable}, \text{private } bty', \text{private}, n'))$, $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{DecodeVal}(\text{private } bty', n', \omega'_1) = [v''_0, ..., v''_{n'-1}]$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v''_0, ..., v''_{n'-1}]$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$.

Given $\forall v_m \in [v_0, ..., v_{n-1}].\ v'_m = ((i = \text{encrypt}(m)) \wedge v) \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, $\forall v''_{m'} \in [v''_0, ..., v''_{n'-1}].\ v'''_{m'} = ((i' = \text{encrypt}(m')) \wedge v'') \vee (\neg(i' = \text{encrypt}(m')) \wedge v''_{m'})$, $v' = v'''$, $i = i'$, $n = n'$, and $\forall m \in \{0, ..., n-1\}$,

$v_m = v''_m$, we have $m, m' \in \{0, ..., n-1\}$ and by Axiom 3.3.1 we have $\text{encrypt}(m) = \text{encrypt}(m')$. Therefore, we have $[v'_0, ..., v'_{n-1}] = [v'''_0, ..., v'''_{n'-1}]$.

Given $\text{UpdateVal}(\sigma_2,\, l_1, [v'_0, ..., v'_{n-1}], \text{private } bty) = \sigma_3$, $\text{UpdateVal}(\sigma'_2,\, l'_1, [v'''_0, ..., v'''_{n'-1}], \text{private } bty') = \sigma'_3$, $\sigma_2 = \sigma'_2$, $l_1 = l'_1$, $bty = bty'$, and $[v'_0, ..., v'_{n-1}] = [v'''_0, ..., v'''_{n'-1}]$, by Lemma 3.3.12 we have $\sigma_3 = \sigma'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma,\, \sigma,\, \text{acc},\, x[e]) \;\Downarrow_{rao}\; (\gamma,\; \sigma_1,\, \text{acc},\, v)$

Given $\Pi \triangleright (\gamma,\, \sigma,\, \text{acc},\, x[e]) \;\Downarrow_{rao}\; (\gamma,\; \sigma_1,\, \text{acc},\, v)$ by rule Public 1D Array Read Out of Bounds Public Index, we have $\text{Label}(e,\, \gamma) = \text{public}$, $\gamma(x) = (l,\, \text{public const } bty*)$, $(\gamma,\, \sigma,\, \text{acc},\, e) \Downarrow_{c_1} (\gamma,\; \sigma_1,\, \text{acc},\, i)$, $\sigma_1(l) = (\omega,\, \text{public const } bty*,\, 1,\, \text{PermL}(\text{Freeable},\, \text{public const } bty*,\, \text{public},\, 1))$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1,\, [(l_1,\, 0)],\, [1],\, 1]$, $\sigma_1(l_1) = (\omega_1,\, \text{public} bty,\, n,\, \text{PermL}(\text{Freeable},\, \text{public } bty,\, \text{public},\, n))$, $(i < 0) \vee (i \geq n)$, and $\text{ReadOOB}(i,\; n,\; l_1,\; \text{public } bty,\; \sigma_1) = (v,\; 1)$.

By definition 3.3.1, given $c = rao$, we have $c \simeq_L c'$ if and only if $c' = rao$.

Given $\Sigma \triangleright (\gamma,\, \sigma,\, \text{acc},\, x[e]) \;\Downarrow_{rao}\; (\gamma,\; \sigma'_1,\, \text{acc},\, v')$ by rule Public 1D Array Read Out of Bounds Public Index, we have $\text{Label}(e,\, \gamma) = \text{public}$, $\gamma(x) = (l',\, \text{public const } bty'*)$, $(\gamma,\, \sigma,\, \text{acc},\, e) \Downarrow_{c'_1} (\gamma,\; \sigma'_1,\, \text{acc},\, i')$, $\sigma'_1(l') = (\omega',\, \text{public const } bty'*,\, 1,\, \text{PermL}(\text{Freeable},\, \text{public const } bty'*,\, \text{public},\, 1))$, $\text{DecodePtr}(\text{public const } bty'*,\; 1,\; \omega') = [1,\; [(l'_1,\; 0)],\; [1],\; 1]$, $\sigma'_1(l'_1) = (\omega'_1,\, \text{public } bty,\, n',\, \text{PermL}(\text{Freeable},\, \text{public } bty',\, \text{public},\, n'))$, $(i' < 0) \vee (i' \geq n')$, and $\text{ReadOOB}(i',\; n',\; l'_1,\, \text{public } bty',\, \sigma'_1) = (v',\; 1)$.

Given $\gamma(x) = (l,\, \text{public const } bty*)$ and $\gamma(x) = (l',\, \text{public const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $(\gamma,\, \sigma,\, \text{acc},\, e) \;\Downarrow_{c_1}\; (\gamma,\, \sigma_1,\, \text{acc},\, i)$ and $(\gamma,\, \sigma,\, \text{acc},\, e) \;\Downarrow_{c'_1}\; (\gamma,\, \sigma'_1,\, \text{acc},\, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $\sigma_1(l) = (\omega,\, \text{public const } bty*,\, 1,\, \text{PermL}(\text{Freeable},\, \text{public const } bty*,\, \text{public},\, 1))$, $\sigma'_1(l') = (\omega',\, \text{public const } bty'*,\, 1,\, \text{PermL}(\text{Freeable},\, \text{public const } bty'*,\, \text{public},\, 1))$, $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given DecodePtr(public const $bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, DecodePtr(public const $bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we have $l_1 = l'_1$.

Given $\sigma_1(l_1) = (\omega_1,$ public $bty,\ n,$ PermL(Freeable, public bty, public, n)), $\sigma'_1(l'_1) = (\omega'_1,$ public $bty,\ n',$ PermL(Freeable, public bty', public, n')), $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given ReadOOB($i,\ n,\ l_1,$ public $bty,\ \sigma_1) = (v,\ 1)$, ReadOOB($i',\ n',\ l'_1,$ public $bty',\ \sigma'_1) = (v',\ 1)$, $i = i'$, $n = n'$, $l_1 = l'_1$, $bty = bty'$, and $\sigma_1 = \sigma'_1$, by Lemma 3.3.24 we have $v = v'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, acc = acc, $v = v'$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow^*_{rao} (\gamma,\ \sigma_1, \text{acc}, v)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{rao} (\gamma,\ \sigma_1, \text{acc}, v)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{rao1} (\gamma,\ \sigma_1, \text{acc}, v)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{rao} (\gamma,\ \sigma_1, \text{acc}, v)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow^*_{rao1} (\gamma,\ \sigma_1, \text{acc}, v)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e]) \Downarrow_{rao} (\gamma,\ \sigma_1, \text{acc}, v)$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao} (\gamma,\ \sigma_3, \text{acc}, \text{skip})$ by rule Public 1D Array Write Out of Bounds Public Index Public Value, we have Label(e_1, γ) = Label(e_2, γ) = public, acc = 0, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v)$, $v \neq$ skip, $\gamma(x) = (l,$ public const $bty*)$, $\sigma_2(l) = (\omega,$ public const $bty*, 1,$ PermL(Freeable, public const $bty*$, public, 1)), DecodePtr(public const $bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1,$ public $bty, n,$ PermL(Freeable, public bty, public, n)), $(i < 0) \vee (i \geq n)$, and WriteOOB($v, i, n, l_1,$ public $bty, \sigma_2) = (\sigma_3, 1)$.

By definition 3.3.1, given $c = wao$, we have $c \simeq_L c'$ if and only if $c' = wao$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao} (\gamma, \sigma'_3, \text{acc}, \text{skip})$ by rule Public 1D Array Write Out of Bounds Public Index Public Value, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, v')$, $v' \neq \text{skip}$, $\gamma(x) = (l', \text{public const } bty'*)$, $\sigma'_2(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty'*, \text{public}, 1))$, $\text{DecodePtr}(\text{ public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_2(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, n'))$, $(i' < 0) \vee (i' \geq n')$, and $\text{WriteOOB}(v', i', n', l'_1, \text{public } bty', \sigma'_2) = (\sigma'_3, 1)$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, v'')$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $v = v''$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x) = (l, \text{public const } bty*)$ and $\gamma(x) = (l', \text{public const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\sigma'_2(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty'*, \text{public}, 1))$, $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr}(\text{ public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we have $l_1 = l'_1$.

Given $\sigma_2(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $\sigma'_2(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, n'))$, $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{WriteOOB}(v, i, n, l_1, \text{public } bty, \sigma_2) = (\sigma_3, 1)$, $\text{WriteOOB}(v', i', n', l'_1, \text{public } bty', \sigma'_2) = (\sigma'_3, 1)$, $v = v'$, $i = i'$, $n = n'$, $l_1 = l'_1$, $bty = bty'$, and $\sigma_2 = \sigma'_2$, by Lemma 3.3.25 we have $\sigma_3 = \sigma'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow^*_{wao} (\gamma, \sigma_3, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao} (\gamma, \sigma_3, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao2} (\gamma, \sigma_3, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao} (\gamma, \sigma_3, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow^*_{wao2} (\gamma, \sigma_3, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao} (\gamma, \sigma_3, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao1} (\gamma, \sigma_3, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao1} (\gamma, \sigma_3, \text{acc}, \text{skip})$ by rule Private 1D Array Write Public Value Out of Bounds Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(i < 0) \vee (i \geq n)$, and $\text{WriteOOB}(\text{encrypt}(v), i, n, l_1, \text{private } bty, \sigma_2) = (\sigma_3, 1)$.

By definition 3.3.1, given $c = wao1$, we have $c \simeq_L c'$ if and only if $c' = wao1$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao1} (\gamma, \sigma'_3, \text{acc}, \text{skip})$ by rule Private 1D Array Write Public Value Out of Bounds Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e_1) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, v')$, $v' \neq \text{skip}$, $\gamma(x) = (l', \text{private const } bty'*)$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty'*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_2(l'_1) = (\omega'_1, \text{private } bty', n', \text{PermL}(\text{Freeable}, \text{private } bty', \text{private}, n'))$, $(i' < 0) \vee (i' \geq n')$, and $\text{WriteOOB}(\text{encrypt}(v'), i', n', l'_1, \text{private } bty', \sigma'_2) = (\sigma'_3, 1)$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, i)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \text{acc}, e_2) \Downarrow_{c_2} (\gamma, \sigma_2, \text{acc}, v)$, $(\gamma, \sigma'_1, \text{acc}, e_2) \Downarrow_{c'_2} (\gamma, \sigma'_2, \text{acc}, v'')$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $v = v''$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{private}, 1))$, $\sigma_2'(l') = (\omega', \text{private const } bty'*, 1, \text{PermL(Freeable, private const } bty'*, \text{private}, 1))$, $\sigma_2 = \sigma_2'$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr(private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr(private const } bty'*, 1, \omega') = [1, [(l_1', 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l_1', 0)], [1], 1]$. Therefore we have $l_1 = l_1'$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL(Freeable, private } bty, \text{private}, n))$, $\sigma_2'(l_1') = (\omega_1', \text{private } bty', n', \text{PermL(Freeable, private } bty', \text{private}, n'))$, $\sigma_2 = \sigma_2'$, and $l_1 = l_1'$, we have $\omega_1 = \omega_1'$ and $n = n'$.

Given $v = v'$, by Axiom 3.3.1 we have $\text{encrypt}(v) = \text{encrypt}(v')$.

Given $\text{WriteOOB}(\text{encrypt}(v), i, n, l_1, \text{private } bty, \sigma_2) = (\sigma_3, 1)$, $\text{WriteOOB}(\text{encrypt}(v'), i', n', l_1', \text{private } bty', \sigma_2') = (\sigma_3', 1)$, $\text{encrypt}(v) = \text{encrypt}(v')$, $i = i'$, $n = n'$, $l_1 = l_1'$, $bty = bty'$, and $\sigma_2 = \sigma_2'$, by Lemma 3.3.25 we have $\sigma_3 = \sigma_3'$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma_3'$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow^*_{wao1} (\gamma, \sigma_3, \text{acc}, \text{skip})$
This case is similar to case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x[e_1] = e_2) \Downarrow_{wao1} (\gamma, \sigma_3, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{ra5} (\gamma, \sigma, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{ra5} (\gamma, \sigma, \text{acc}, [v_0, ..., v_{n-1}])$ by rule Private 1D Array Read Entire Array, we have $\gamma(x) = (l, \text{private const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{private}, 1))$, $\text{DecodePtr(private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma(l_1) = (\omega_1, \text{private } bty, n, \text{PermL(Freeable, private } bty, \text{private}, n))$, and $\text{DecodeVal(private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$.

By definition 3.3.1, given $c = ra5$, we have $c \simeq_L c'$ if and only if $c' = ra5$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{ra5} (\gamma, \sigma, \text{acc}, [v_0', ..., v_{n'-1}'])$ by rule Private 1D Array Read Entire Array, we have

$\gamma(x) = (l', \text{private const } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma(l') = (\omega', \text{private const } bty'*, 1, \text{PermL(Freeable, private const } bty'*\text{, private, } 1))$, $\text{DecodePtr(private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma(l'_1) = (\omega'_1, \text{private } bty', n', \text{PermL(Freeable, private } bty'\text{, private, } n'))$, and $\text{DecodeVal(private } bty', n', \omega'_1) = [v'_0, ..., v'_{n'-1}]$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{private const } bty*, 1, \text{PermL(Freeable, private const } bty*\text{, private, } 1))$, $\sigma(l') = (\omega', \text{private const } bty'*, 1, \text{PermL(Freeable, private const } bty'*\text{, private, } 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr(private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr(private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore we have $l_1 = l'_1$.

Given $\sigma(l_1) = (\omega_1, \text{private } bty, n, \text{PermL(Freeable, private } bty\text{, private, } n))$, $\sigma(l'_1) = (\omega'_1, \text{private } bty', n', \text{PermL(Freeable, private } bty'\text{, private, } n'))$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodeVal(private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{DecodeVal(private } bty', n', \omega'_1) = [v'_0, ..., v'_{n'-1}]$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v'_0, ..., v'_{n'-1}]$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\text{acc} = \text{acc}$, $[v_0, ..., v_{n-1}] = [v'_0, ..., v'_{n'-1}]$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{ra4} (\gamma, \sigma, \text{acc}, [v_0, ..., v_{n-1}])$
This case is similar to case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x) \Downarrow_{ra5} (\gamma, \sigma, \text{acc}, [v_0, ..., v_{n-1}])$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e_1) \Downarrow_{wa5} (\gamma, \sigma_2, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wa5} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Public 1D Array Write Entire Array, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}])$, $\forall v_m \in [v_0, ..., v_{n_e-1}].v_m \neq \text{skip}$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL(Freeable, public const } bty*\text{, public, } 1))$, $\text{DecodePtr(public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL(Freeable, public } bty\text{, public, } n))$, $n_e = n$, and $\text{UpdateVal}(\sigma_1, l_1, [v_0, ..., v_{n_e-1}], \text{public } bty) = \sigma_2$.

By definition 3.3.1, given $c = wa5$, we have $c \simeq_L c'$ if and only if $c' = wa5$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x = e) \Downarrow_{wa5} (\gamma, \sigma'_2, \text{acc}, \text{skip})$ by rule Public 1D Array Write Entire Array, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, [v'_0, ..., v'_{n'_e-1}])$, $\forall v'_m \in [v'_0, ..., v'_{n'_e-1}].v'_m \neq \text{skip}$, $\gamma(x) = (l', \text{public const } bty'*)$, $\sigma'_1(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty'*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_1(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, n'))$, $n'_e = n'$, and $\text{UpdateVal}(\sigma'_1, l'_1, [v'_0, ..., v'_{n'_e-1}], \text{public } bty') = \sigma'_2$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}])$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, [v'_0, ..., v'_{n'_e-1}])$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $[v_0, ..., v_{n_e-1}] = [v'_0, ..., v'_{n'_e-1}]$, and $c_1 \simeq_L c'_1$.

Given $\gamma(x) = (l, \text{public const } bty*)$ and $\gamma(x) = (l', \text{public const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\sigma'_1(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty'*, \text{public}, 1))$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr}(\text{ public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore we have $l_1 = l'_1$.

Given $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $\sigma'_1(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, n'))$, $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{UpdateVal}(\sigma_1, l_1, [v_0, ..., v_{n_e-1}], \text{public } bty) = \sigma_2$, $\text{UpdateVal}(\sigma'_1, l'_1, [v'_0, ..., v'_{n'_e-1}], \text{public } bty') = \sigma'_2$, $\sigma_1 = \sigma'_1$, $l_1 = l'_1$, $bty = bty'$, and $[v_0, ..., v_{n_e-1}] = [v'_0, ..., v'_{n'_e-1}]$, by Lemma 3.3.12 we have $\sigma_2 = \sigma'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e_1) \Downarrow_{wa6} (\gamma, \sigma_3, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e_1) \Downarrow_{wa5} (\gamma, \sigma_2, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e_1) \Downarrow_{wa7} (\gamma, \sigma_2, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, x = e_1) \Downarrow_{wa7} (\gamma, \sigma_2, \text{acc}, \text{skip})$ by rule Private 1D Array Write Entire Public Array, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}])$, $\forall v_m \in [v_0, ..., v_{n_e-1}].v_m \neq \text{skip}$, $\gamma(x) = (l, \text{ private const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_1(l) = (\omega, \text{ private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{ private, } 1))$, $\text{DecodePtr(private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\forall v_m \in [v_0, ..., v_{n_e-1}]. v'_m = \text{encrypt}(v_m)$, $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{ PermL(Freeable, private } bty, \text{ private, } n))$, $n_e = n$, and $\text{UpdateVal}(\sigma_1, l_1, [v'_0, ..., v'_{n_e-1}], \text{private } bty) = \sigma_2$.

By definition 3.3.1, given $c = wa7$, we have $c \simeq_L c'$ if and only if $c' = wa7$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, x = e_1) \Downarrow_{wa7} (\gamma, \sigma'_2, \text{acc}, \text{skip})$ by rule Private 1D Array Write Entire Public Array, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, [v''_0, ..., v''_{n'_e-1}])$, $\forall v''_m \in [v''_0, ..., v''_{n'_e-1}].v_m \neq \text{skip}$, $\gamma(x) = (l', \text{private const } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma'_1(l') = (\omega', \text{ private const } bty'*, 1, \text{PermL(Freeable, private const } bty'*, \text{ private, } 1))$, $\text{DecodePtr(private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\forall v''_{m'} \in [v''_0, ..., v''_{n'_e-1}]. v'''_{m'} = \text{encrypt}(v''_{m'})$, $\sigma'_1(l'_1) = (\omega'_1, \text{ private } bty', n', \text{ PermL(Freeable, private } bty', \text{ private, } n'))$, $n'_e = n'$, and $\text{UpdateVal}(\sigma'_1, l'_1, [v'''_0, ..., v'''_{n'_e-1}], \text{private } bty') = \sigma'_2$.

Given $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \text{acc}, [v_0, ..., v_{n_e-1}])$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow_{c'_1} (\gamma, \sigma'_1, \text{acc}, [v''_0, ..., v''_{n'_e-1}])$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $[v_0, ..., v_{n_e-1}] = [v''_0, ..., v''_{n'_e-1}]$, and $c_1 \simeq_L c'_1$.

Given $[v_0, ..., v_{n_e-1}] = [v''_0, ..., v''_{n'_e-1}]$, we have $n_e = n'_e$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega, \text{ private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{ private, } 1))$, $\sigma'_1(l') = (\omega', \text{private const } bty'*, 1, \text{PermL(Freeable, private const } bty'*, \text{ private, } 1))$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr(private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr(private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore we have $l_1 = l'_1$.

Given $\forall v_m \in [v_0, ..., v_{n_e-1}]. v'_m = \text{encrypt}(v_m)$, $\forall v''_{m'} \in [v''_0, ..., v''_{n'_e-1}]. v'''_{m'} = \text{encrypt}(v''_{m'})$, $n_e = n'_e$, and $[v_0,$

$..., v_{n_e-1}] = [v''_0, ..., v''_{n'_e-1}]$, by Axiom 3.3.1 we have $\text{encrypt}(v_m) = \text{encrypt}(v''_{m'})$ and therefore $[v'_0, ..., v'_{n_e-1}] = [v'''_0, ..., v'''_{n'_e-1}]$.

Given $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL(Freeable, private } bty\text{, private, } n))$, $\sigma'_1(l'_1) = (\omega'_1, \text{ private } bty', n', \text{PermL(Freeable, private } bty'\text{, private, } n'))$, $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{UpdateVal}(\sigma_1, l_1, [v'_0, ..., v'_{n_e-1}], \text{private } bty) = \sigma_2$, $\text{UpdateVal}(\sigma'_1, l'_1, [v'''_0, ..., v'''_{n'_e-1}], \text{private } bty') = \sigma'_2$, $\sigma_1 = \sigma'_1$, $l_1 = l'_1$, $bty = bty'$, and $[v'_0, ..., v'_{n_e-1}] = [v'''_0, ..., v'''_{n'_e-1}]$, by Lemma 3.3.12 we have $\sigma_2 = \sigma'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 3.3.3, we have $\Pi \simeq_L \Sigma$.

$\square$

4 Location-tracking SMC^2

In this chapter, we present Location-tracking SMC^2 along with its proofs of correctness and noninterference. While Basic SMC^2 presented a solution for a formal model of Secure Multiparty Computation compiler for general purpose programs written in C, unfortunately, directly adopting this approach exhibits problematic behavior when more complex operations are considered. Basic SMC^2 could not provide full pointer support, as pointer dereference writes within private branches were disallowed, as well as needing to ensure array writes at public indices did not go out of bounds. The main challenge with these operations within private-conditioned branches was due to the tracking of modifications relying on variable names. Thus, if a pointer referred to the location where a variable was stored and both an assignment to the variable and a pointer dereference write occurred, we could no longer guarantee that the value stored as a result of the resolution would be the correct value, as in attempting to use that style of tracking we would end up resolving the same location twice. Let us next consider examples illustrating why these operations are problematic.

The first level of locations that the pointer refers to is managed, as shown in Figure 6.2b, but dereferencing the pointer and modifying the value stored at the location that is pointed to can result in incorrect program behavior and ultimately information leakage. Any SMC system that supports both pointers and branches, using PICCO style pointers and classic branch resolution cannot handle these cases. This occurs because the approach we discussed above relies on assignment statements and single-level, constant location changes to properly restore and resolve the changes made inside the private-conditioned branch. One can try to modify this approach to support tracking changes made using pointers at a higher level of indirection (i.e., tracking `*p=c` using temporary variables `dp_t` and `dp_e`, as shown in Figure 4.1a). However, this modification can lead to the incorrect resolution of data when multiple levels of indirection of a pointer are modified within the private-conditioned branches. An example of this is shown in Figure 4.1a, where we modify `*p` in the **`then`** branch, and then change the location that is pointed to by `p` in the **`else`** branch. The **`then`** branch, restoration, and the **`else`** branch will execute correctly, however, resolving the variables after the completion of the **`else`** branch will not. Given that we have modified the location pointed to by `p`, when we attempt to resolve the modification we made using `*p`, we will read from and modify the value in $l_{\mathtt{b}}$ (where `p` currently

```
1 private int a=3,
2   b=7,c=5,*p=&a;
3 if (a<b) {
4   *p=c; }
5 else {
6   p=&b; }
```

```
1 private int a=3,b=7,c=5,*p=&a,res=a<b,
2   dp_e=*p,dp_t,*p_e=p,*p_t;
3 {*p=c;}
4 dp_t=*p; p_t=p; *p=dp_e; p=p_e;
5 {p=&b;}
6 *p=(res·dp_t)+((1-res)·*p); p=resolve(res,p_t,p);
```

location	initial	**then**	restore	**else**	resolve
$l_{\mathtt{a}}$	3	5	3	3	3
$l_{\mathtt{b}}$	7	7	7	7	5
$l_{\mathtt{p}}$	$(l_{\mathtt{a}}), (1)$	$(l_{\mathtt{a}}), (1)$	$(l_{\mathtt{a}}), (1)$	$(l_{\mathtt{b}}), (1)$	$(l_{\mathtt{a}}, l_{\mathtt{b}}), (1, 0)$
$l_{\mathtt{dp_t}}$			5	5	5
$l_{\mathtt{p_t}}$			$(l_{\mathtt{a}}), (1)$	$(l_{\mathtt{a}}), (1)$	$(l_{\mathtt{a}}), (1)$

location	value
$l_{\mathtt{res}}$	1
$l_{\mathtt{dp_e}}$	3
$l_{\mathtt{p_e}}$	$(l_{\mathtt{a}}), (1)$

(a) Challenges of pointer manipulations within private-conditioned branches.

```
1 public int i=1,j=2;
2 private int a[j]={0,0},
3   b=7,c=3,d=4;
4 if (c<d) {
5   a[i]=c; }
6 else {
7   a[j]=d; }
```

```
1 public int i=1,j=2;
2 private int a[j]={0,0},b=7,c=3,d=4;
3 private int res=c<d,a_t,a_e=a;
4 a[i]=c;
5 a_t=a; a=a_e;
6 a[j]=d;
7 a=(res·a_t)+((1-res)·a);
```

location	initial	**then**	restore	**else**	resolve
$l_{\mathtt{a}}$	0, 0	0, 3	0, 0	0, 0	0, 3
$l_{\mathtt{b}}$	7	7	7	4	4
$l_{\mathtt{a_t}}$			0, 3	0, 3	0, 3

location	value
$l_{\mathtt{res}}$	1
$l_{\mathtt{a_e}}$	0, 0

(b) Challenges of writing at a public index in a private array within private-conditioned branches.

Figure 4.1: Examples of the challenges of private-conditioned branching examples, with pointer challenges shown in 4.1a and array challenges shown in 4.1b. We show values in memory that change in the table to the left, and values for temporary variables that do not change in the table to the right. We indicate correct updates in memory in green, and problematic values in memory in red.

points) instead of the value in $l_{\mathtt{a}}$ (where `p` pointed and wrote to in the **`then`** branch).

We encounter a similar issue if we write to a public index within a private array during the execution of a private-conditioned branch, and that index happens to be out of bounds. We give an example of this in Figure 4.1b. Here, we assume that `b` is assigned the location directly after the array data's location, thus giving us a *well-aligned* out-of-bounds write to illustrate why simple variable tracking is not enough here. In any given implementation, an out-of-bounds access is not guaranteed to be *well-aligned*, and therefore unpredictable behavior can occur. In this example, we have a zero-indexed array `a` of two elements. In the **`then`** branch, we modify index 1 (or the second element), then store this updated array and return the array back to it's initial state. In the **`else`** branch, we modify index 2 (out-of-bounds of the array), which updates

the value stored for b. Given that we were not tracking b, this value does not get resolved, and any further uses of b will result in incorrect results. By using location tracking, we would catch that the location $l_{\mathtt{b}}$ was modified, and in turn properly restore it to it's original value (as long as the out-of-bounds access is *well-aligned*).

Therefore, we introduce Location-tracking SMC^2, which extends the Basic SMC^2 model with additional tracking mechanisms to enable location-tracking during private-conditioned branches, resolving the potential issues for tracking with pointers and arrays discussed above. We will discuss the specifics of Location-tracking SMC^2 next.

4.1 Formal Semantics

$$
\begin{array}{rlcl}
C & \in \mathit{Configuration} & ::= & (\gamma, \sigma, \Delta, \overline{\chi}, \ \mathrm{bid}, \ \mathrm{acc}, s) \\
\\
\delta & \in \mathit{LocationMap} & ::= & f : l \rightarrow (v, v, v) \\
\Delta & \in \mathit{LocationMapList} & ::= & [\,] \mid \delta :: \Delta \\
\\
\chi & \in \mathit{LocalVariableList} & ::= & [\,] \mid l :: \chi \\
\overline{\chi} & \in \mathit{LocalVariableTracker} & ::= & [\,] \mid \chi :: \overline{\chi} \\
\mathrm{bid} & \in \mathit{BranchIdentifier} & ::= & \mathit{none} \mid \mathit{then} \mid \mathit{else}
\end{array}
$$

Figure 4.2: Location-tracking SMC^2 configuration with added location map Δ, local variable list $\overline{\chi}$, and branch identifier bid

In this section, we will present the Location-tracking SMC^2 semantics with respect to the grammar (Figure 3.1). The semantic judgements in Location-tracking SMC^2 are defined over a seven-tuple configuration $C = (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s)$, where each rule is a reduction from one configuration to a subsequent. We denote the environment as γ; memory as σ; location map Δ; local variable tracker $\overline{\chi}$; branch identifier bid; the level of nesting of private-conditioned branches as acc; and a big-step evaluation of a statement s to a value v using $\Downarrow$. We annotate each evaluation with evaluation codes (i.e., $\Downarrow_d^t$) to facilitate reasoning over evaluation trees, and we annotate evaluations that are not *well-aligned* with a star (i.e., $\Downarrow_d^{t*}$) to identify the rules that we cannot prove correctness over, as they produce unpredictable behavior. The assertions in each semantic rule can be read in sequential order, from left to right and top to bottom. When proving correctness for Location-tracking SMC^2, we will use the same Vanilla C semantics as for Basic SMC^2, shown in subsection 3.1.3. We chose not to develop an additional set of semantics as the Vanilla C semantics do not change in relation to this set of semantics, and adding extra $\square$ in to the Vanilla C configuration in

order to maintain an exact configuration mapping proved unnecessary, as we simply define the mapping as consolidating all unused portions for Vanilla C (i.e., $\Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}$) into the single $\square$.

In this semantics, we introduce three new tracking features: location map Δ, local variable tracker $\overline{\chi}$, and branch identifier bid. All three of these features are introduced to manage tracking changes by location within private-conditioned branches. We will briefly discuss each here, and more fully discuss their uses when we discuss the private-conditioned **`if else`** statement and its algorithms. The first, location map Δ, is a list of maps δ, one for each level of nesting within the current private-conditioned **`if else`** statement. At the start of a private-conditioned **`if else`** statement, we append a new sublist to Δ to constitute the current scope of changes within this specific private-conditioned **`if else`** statement. This sublist is removed from Δ once we have completed resolution and the changes are out of scope. Of course, it is not removed before changes are propagated to any higher level of tracking when we are in a nested private-conditioned **`if else`** statement – this is discussed in more detail when we discuss the private-conditioned **`if else`** statement and the algorithms used within it. Each location map δ maps memory block identifiers to a tuple of three values – the original value stored in the block, the value stored within the block by the end of the **`then`** branch, and the value stored within the block by the end of the **`else`** branch. This gives us the full set of different values within a block throughout the evaluation of the private-conditioned **`if else`** statement, allowing us to properly restore the modified memory blocks between the evaluation of the branches and resolve the true value once we have completed evaluation of both branches. It is important to note here that the three values stored in each mapping will be of the same type, and either a singular value, a list of values, or a pointer data structure.

The second, local variable tracker $\overline{\chi}$, is a list of lists (one for each level of nesting within the current private-conditioned **`if else`** statement evaluation) tracking which memory blocks refer to local variables within the current scope (i.e., level of nesting). This allows us to limit the tracking to variables that will exist beyond the scope of the private-conditioned **`if else`** statement. The last, branch identifier bid, allows us to know which private-conditioned branch we are currently in, if any. This is necessary to help us with storing values into Δ, as if we are in the **`then`** branch, we need to store the updated value as the **`then`** value within Δ, and similarly for when we are in the **`else`** branch. The $none$ identifier is used to indicate that we are not within a private-conditioned branch, and therefore we do not need to be tracking any modifications that are made; in the semantic rules, we use this in conjunction with the assertion that the accumulator is 0.

4.1.1 Location-tracking SMC^2

In this section, we give the Location-tracking SMC^2 semantics for completeness, and give a basic description of changes between this set of semantics and the previous set. For a more detailed description of the rules, please refer back to the Basic SMC^2 versions in Section 3.1.4. Figure 4.3 gives the semantics for **`if else`** statements. Figures 4.4 gives the semantics for declarations, reading from, and writing to regular (non-array, non-pointer) variables, as well as for loops. Figure 4.5 and 4.6 give the semantics for reading in input data and writing out output data, respectively. Figure 4.7 gives the semantics for the pre-increment operation.

Figure 4.8 gives the semantics for addition and subtraction, and Figure 4.9 gives the semantics for multiplication and division. Figures 4.10, 4.11, and 4.12 give the semantics for less than comparisons, equal to comparisons, and not equal to comparisons, respectively. Figure 4.13 gives the semantics for functions, sequencing, and declaration assignments. Figure 4.14 gives the semantics for memory allocation, deallocation, and casting. Figure 4.15 gives the semantics for array declarations and writing an entire array. Figures 4.16, 4.17, and 4.18 give the semantics for reading from an array, writing to an array, and reading and writing out-of-bounds of an array, respectively. Figure 4.19 gives the semantics for pointer declarations, reading, and writing. Figures 4.20 and 4.21 give the semantics for pointer dereferences and pointer dereference writes, respectively.

Location-tracking SMC^2 rules Public If Else True and Public If Else False are nearly identical to the Basic SMC^2 rules, just with updated configurations. The first three assertions of the Location-tracking SMC^2 rule Private If Else are very similar to the Basic SMC^2 version, as we first ensure the condition is private, then evaluate it to a value and store it. For Location-tracking SMC^2, we track all modifications by location as they occur, so we no longer have the extraction or initialization algorithms, and instead push an empty list into our tracking structures to prepare them for tracking modifications and local locations at this level of nesting. We then proceed to evaluate the **`then`** branch, which will keep location map Δ up to date throughout the evaluation, adding new modifications as new locations are modified and updating them if more than one modification of the location occurs, so the value stored for the **`then`** branch will be the most recent value.

Next, we restore the original values back into memory for all locations, using our updated algorithm (Algorithm 92) in order to properly handle the location map Δ. The idea behind restoration remains the same - store all final changes for the branch and restore memory back to its original state from before the evaluation of the **`then`** branch. We then evaluate the **`else`** branch, which will again keep location map Δ up to date on

```
1 private int a=3, b=7, c=0;
2 if (a < b) {c = a;}
3 else {c = b;}
```

(a) SMC^2 code.

```
1 private int a=3, b=7, c=0;
2 private int res1 = a < b,
3 c_t = c, c_e = c;
4 c = a;
5 c_t = c; c = c_e;
6 c = b;
7 c = (res1 ∧ c_t) ∨ (¬res1 ∧ c);
```

(b) Basic SMC^2 code execution.

```
1 private int a=3, b=7, c=0;
2 private int res1 = a < b;
3 c = a;  l_c = (0, 3, 0);
4 c = l_c[0];
5 c = b;  l_c = (0, 3, 7);
6 c = (res1 ∧ l_c[1]) ∨ (¬res1 ∧ l_c[2]);
```

(c) Location-tracking SMC^2 code execution.

$$\frac{\begin{array}{l}\text{Label}(e,\gamma) = \text{private} \qquad (\gamma,\sigma,\text{acc},e) \Downarrow_e (\gamma,\sigma_1,\text{acc},n)\\ (\gamma,\sigma_1,\text{acc},\text{private int } res_{\text{acc}+1} = n) \Downarrow_s (\gamma_1,\sigma_2,\text{acc},\text{skip})\\ \text{Extract_variables}(s_1, s_2) = x_{list}\\ \text{InitializeVariables}(x_{list},\ \gamma_1,\ \sigma_2,\ \text{acc}+1) = (\gamma_2,\sigma_3)\\ (\gamma_2,\sigma_3,\text{acc}+1,s_1) \Downarrow_s (\gamma_3,\sigma_4,\text{acc}+1,\text{skip})\\ \text{RestoreVariables}(x_{list},\ \gamma_3,\ \sigma_4,\ \text{acc}+1) = \sigma_5\\ (\gamma_2,\sigma_5,\text{acc}+1,s_2) \Downarrow_s (\gamma_4,\sigma_6,\text{acc}+1,\text{skip})\\ \text{ResolveVariables}(x_{list},\ \gamma_4,\ \sigma_6,\ \text{acc}+1,\ res_{\text{acc}+1}) = \sigma_7\end{array}}{(\gamma,\ \sigma,\ \text{acc},\ \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow_{iep} (\gamma,\ \sigma_7,\ \text{acc},\ \text{skip})}$$

(d) Basic SMC^2 rule Private If Else.

$$\frac{\begin{array}{l}\text{Label}(e,\gamma) = \text{private}\\ (\gamma,\sigma,\Delta,\overline{\chi},\text{bid},\text{acc},e)\\ \qquad \Downarrow^t_e (\gamma,\sigma_1,\Delta_1,\overline{\chi},\text{bid},\text{acc},n)\\ (\gamma,\ \sigma_1,\Delta_1,\overline{\chi},\text{bid},\text{acc},\text{private int } res_{\text{acc}+1} = n)\\ \qquad \Downarrow^t_s (\gamma_1,\sigma_2,\Delta_1,\overline{\chi}_1,\text{bid},\text{acc},\text{skip})\\ \Delta_2 = \Delta_1.push([\,]) \qquad \overline{\chi}_2 = \overline{\chi}_1.push([\,])\\ (\gamma_1,\sigma_2,\Delta_2,\overline{\chi}_2,then,\text{acc}+1,s_1)\\ \qquad \Downarrow^t_s (\gamma_2,\sigma_3,\Delta_3,\overline{\chi}_3,then,\text{acc}+1,\text{skip})\\ \text{T_restore}(\sigma_3,\Delta_3,\text{acc}+1) = \sigma_4\\ (\gamma_1,\sigma_4,\Delta_3,\overline{\chi}_2,else,\text{acc}+1,s_2)\\ \qquad \Downarrow^t_s (\gamma_3,\sigma_5,\Delta_4,\overline{\chi}_4,else,\text{acc}+1,\text{skip})\\ \text{T_resolve}(\sigma_5,\Delta_4,\overline{\chi},\text{bid},\text{acc}+1,res_{\text{acc}+1}) = (\sigma_6,\Delta_5)\\ \Delta_6 = \Delta_5.pop()\end{array}}{\begin{array}{c}(\gamma,\sigma,\Delta,\overline{\chi},\text{bid},\text{acc},\text{if } (e)\ s_1 \text{ else } s_2)\\ \Downarrow^t_{iep} (\gamma,\sigma_6,\Delta_6,\overline{\chi},\text{bid},\text{acc},\text{skip})\end{array}}$$

(e) Location-tracking SMC^2 rule Private If Else.

$$\frac{\begin{array}{c}\text{Label}(e,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_e (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)\\ n \neq 0 \qquad (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1) \Downarrow^t_s (\gamma_1,\ \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow^t_{iet} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

(f) Location-tracking SMC^2 rule Public If Else True

$$\frac{\begin{array}{c}\text{Label}(e,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_e (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)\\ n = 0 \qquad (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_2) \Downarrow^t_s (\gamma_1,\ \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow^t_{ief} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

(g) Location-tracking SMC^2 rule Public If Else False

Figure 4.3: **`if else`** branching on private data example (4.3a, 4.3b) matching to the Basic SMC^2 (4.3d), and Location-tracking SMC^2 (4.3e) rules. Coloring in the rules highlight the corresponding code and rule execution. The public **`if else`** rules (4.3f, 4.3g) are shown for reference.

all modifications. Finally, we use our updated resolution algorithm (Algorithm 93), to evaluate and store the true values into memory. Again, the idea behind this algorithm is the same as the Basic SMC^2 version, and the resolution mechanisms are identical for both values and pointer locations; we simply facilitate having tracked everything by location within location map Δ instead of using temporary variables. Lastly, we also remove the tracking list for this level of nesting from Δ, but keep all of the other changes made to outer levels of nesting within Δ to ensure proper tracking, no matter the level of nesting. In our ending state, we return the original local variable tracker $\overline{\chi}$, as any additions we made are out of scope beyond this rule, just like those made to environment γ.

We have added the semantic rule Location-tracking Private Declaration (Inside a Private - Conditioned If Else Branch), allowing private declarations within private-conditioned branches. It facilitates adding the new location to our tracking structure for local variables. The other two basic declaration rules remain the same, as does reading from variables. In the rules for writing to regular (non-pointer, non-array) variables, the only difference is using Algorithm 84 in place of the Basic SMC^2 update algorithm (47). This algorithm handles updates just like the previous one did, only with additional checks to see if we are within a private-conditioned branch, and if so, to ensure location map Δ if up to date with the original value and any modifications.

The Location-tracking SMC^2 rules for input and output of data are nearly identical to those in Basic SMC^2, simply with updated configurations. Likewise, the Location-tracking SMC^2 rules for binary comparisons and operations are only differing from Basic SMC^2 in their configurations. The Location-tracking SMC^2 rules for the pre-increment operation also nearly identical to those in Basic SMC^2, simply using the Location-tracking SMC^2 versions of the update algorithms in order to ensure that tracking is handled appropriately when such operations occur within private-conditioned branches. Due to the length of the semantics, we do not show the versions of these rules that are not *well-aligned*, as the rules are nearly identical to the *well-aligned* versions; the only difference is with the tag indicating alignment being 0 instead of 1. We are still able to handle these versions of the rules within the noninterference proof, and the interested reader can look back on the Basic SMC^2 versions of these rules to better visualize the differences.

Additionally, our Location-tracking SMC^2 semantic rules for sequencing remain nearly identical to the Basic SMC^2 semantics, ensuring to pass along the new tracking structures appropriately. The Location-tracking SMC^2 semantics for functions do not change, evaluating new function definitions for any public side effects and ensuring that functions with public side effects cannot be called from within private-conditioned branches. The Location-tracking SMC^2 rules for memory allocation, deallocation, and casting do not change.

Location-tracking Public Declaration

$$\frac{(ty = \text{public } bty) \vee (ty = \text{char}) \quad (\text{acc} = 0) \wedge (\text{bid} = none) \quad l = \phi() \quad \gamma_1 = \gamma[x \rightarrow (l,\ ty)] \quad \omega = \text{EncodeVal}(ty, \text{NULL}) \quad \sigma_1 = \sigma[l \rightarrow (\omega,\ ty,\ 1,\ \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x)\ \Downarrow^t_d\ (\gamma_1,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Declaration

$$\frac{((ty = bty) \vee (ty = \text{private } bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float})) \quad (\text{acc} = 0) \wedge (\text{bid} = none) \quad l = \phi() \quad \omega = \text{EncodeVal}(ty, \text{NULL}) \quad \gamma_1 = \gamma[x \rightarrow (l, ty)] \quad \sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{private}, 1))]}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x)\ \Downarrow^t_{d1}\ (\gamma_1,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Declaration (Inside a Private - Conditioned If Else Branch)

$$\frac{((ty = bty) \vee (ty = \text{private } bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float})) \quad l = \phi() \quad \gamma_1 = \gamma[x \rightarrow (l, ty)] \quad \sigma_1 = \sigma[l \rightarrow (\text{NULL}, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{private}, 1))] \quad \overline{\chi}_1 = l :: \overline{\chi}[\text{acc}] \quad (\text{acc} > 0) \wedge ((\text{bid} = then) \vee (\text{bid} = else))}{(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x)\ \Downarrow^t_{d2}\ (\gamma_1, \sigma_1, \Delta, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})}$$

Location-tracking Read Public Variable

$$\frac{\gamma(x) = (l,\ \text{public } bty) \quad \sigma(l) = (\omega,\ \text{public } bty,\ 1,\ \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1)) \quad \text{DecodeVal}(\text{public } bty,\ 1,\ \omega) = v}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x)\ \Downarrow^t_r\ (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)}$$

Location-tracking Read Private Variable

$$\frac{\gamma(x) = (l,\ \text{private } bty) \quad \sigma(l) = (\omega,\ \text{private } bty,\ 1,\ \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) \quad \text{DecodeVal}(\text{private } bty,\ 1,\ \omega) = v}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x)\ \Downarrow^t_{r1}\ (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)}$$

Location-tracking Write Public Variable

$$\frac{\text{Label}(e, \gamma) = \text{public} \quad (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_e (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v) \quad v_1 \neq \text{skip} \quad \gamma(x) = (l,\ \text{public } bty) \quad \text{T_UpdateVal}(\sigma_1,\ l,\ v,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{public } bty) = (\sigma_2,\ \Delta_2)}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e)\ \Downarrow^t_w\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Write Private Variable

$$\frac{\text{Label}(e, \gamma) = \text{private} \quad (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_e (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v) \quad v_1 \neq \text{skip} \quad \gamma(x) = (l,\ \text{private } bty) \quad \text{T_UpdateVal}(\sigma_1,\ l,\ v,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{private } bty) = (\sigma_2,\ \Delta_2)}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e)\ \Downarrow^t_{w1}\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Write Private Variable Public Value

$$\frac{\text{Label}(e, \gamma) = \text{public} \quad (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_e (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v) \quad v_1 \neq \text{skip} \quad \gamma(x) = (l, \text{private } bty) \quad \text{T_UpdateVal}(\sigma_1, l, \text{encrypt}(v), \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty) = (\sigma_2,\ \Delta_2)}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e)\ \Downarrow^t_{w2}\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking While End

$$\frac{\text{Label}(e, \gamma) = \text{public} \quad (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_e (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n) \quad n\ =\ 0}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{while}\ (e)\ s)\ \Downarrow^t_{wle}\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking While Continue

$$\frac{\text{Label}(e, \gamma) = \text{public} \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow^t_e\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \quad n\ \neq\ 0 \quad (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s)\ \Downarrow^t_s\ (\gamma_1,\ \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \quad (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{while}\ (e)\ s)\ \Downarrow^t_s\ (\gamma_2,\ \sigma_3,\ \Delta_3,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ \text{skip})}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{while}\ (e)\ s)\ \Downarrow^t_{wlc}\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{while}\ (e)\ s)}$$

Figure 4.4: Location-tracking SMC^2 semantic rules for basic variable declarations, reading, writing, and loops.

Location-tracking SMC Input Public Value

$$\frac{\begin{array}{c}\text{acc} = 0 \quad \text{Label}(e_2, \gamma) = \text{public} \quad (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow_e^t (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x) \\ (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow_e^t (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n) \quad \gamma(x) = (l,\ \text{public}\ bty) \\ \text{InputValue}(x,\ n) = n_1 \quad (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = n_1) \Downarrow_s^t (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcinput}(e_1,\ e_2)) \Downarrow_{inp}^t (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking SMC Input Private Value

$$\frac{\begin{array}{c}\text{acc} = 0 \quad \text{Label}(e_2, \gamma) = \text{public} \quad (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow_e^t (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \quad \gamma(x) = (l,\ \text{private}\ bty) \\ \text{InputValue}(x,\ n) = n_1 \quad (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = n_1) \Downarrow_s^t (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcinput}(e_1,\ e_2)) \Downarrow_{inp3}^t (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking SMC Input Public Array

$$\frac{\begin{array}{c}\text{Label}(e_2, \gamma) = \text{public} \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \\ \text{Label}(e_3, \gamma) = \text{public} \quad (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_3) \Downarrow_e^t (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \quad \gamma(x) = (l,\ \text{public const}\ bty*) \\ \text{acc} = 0 \quad \text{InputArray}(x,\ n,\ n_1) = [m_0,\ ...,\ m_{n_1}] \\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = [m_0,\ ...,\ m_{n_1}]) \Downarrow_s^t (\gamma,\ \sigma_4,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcinput}(e_1,\ e_2,\ e_3)) \Downarrow_{inp1}^t (\gamma,\ \sigma_4,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking SMC Input Private Array

$$\frac{\begin{array}{c}\text{Label}(e_2, \gamma) = \text{public} \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \\ \text{Label}(e_3, \gamma) = \text{public} \quad (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \\ \text{acc} = 0 \quad (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_3) \Downarrow_e^t (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ \gamma(x) = (l,\ \text{private const}\ bty*) \quad \text{InputArray}(x,\ n,\ n_1) = [m_0,\ ...,\ m_{n_1}] \\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = [m_0,\ ...,\ m_{n_1}]) \Downarrow_s^t (\gamma,\ \sigma_4,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcinput}(e_1,\ e_2,\ e_3)) \Downarrow_{inp4}^t (\gamma,\ \sigma_4,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Figure 4.5: Location-tracking SMC^2 semantic rules for input.

We add the assertion that $\text{bid} = none$ into the rules for memory allocation and deallocation as an additional check to asserting that the accumulator is 0, ensuring that we are not within a private-conditioned branch as these are rules causing public side effects. For Location-tracking SMC^2 arrays, the rules remain nearly identical. We add rule Location-tracking Private Array Declaration (Inside a Private - Conditioned If Else Branch), allowing local private array within branches, which will be added to the tracking structure for local locations, as such locations are not intended to persist beyond the scope of the branch and do not need to be tracked. For Location-tracking SMC^2 array writes, we use our new algorithms for updating memory to facilitate handling of tracking within private-conditioned branches; the main functionality of these algorithms does not change. For the sake of space, as with pre-increment operations, we do not show the versions of reading and writing out-of-bounds that are not *well-aligned*, as these versions do not show anything new beyond the *well-aligned* versions. We are still able to handle these versions of the rules within the noninterference proof, and the interested reader can look back on the Basic SMC^2 versions of these rules to better visualize the differences between such rules.

Location-tracking SMC Output Value

$$\frac{\begin{array}{c}\text{Label}(e_2,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \qquad \gamma(x) = (l,\ \text{public } bty) \\ \sigma_2(l) = (\omega,\ \text{public } bty,\ 1,\ \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1)) \\ \text{DecodeVal}(\text{public } bty,\ 1,\ \omega) = n_1 \qquad \text{OutputValue}(x,\ n, n_1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcoutput}(e_1,\ e_2)) \Downarrow_{out}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking SMC Output Private Value

$$\frac{\begin{array}{c}\text{Label}(e_2,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \qquad \gamma(x) = (l,\ \text{private } bty) \\ \sigma_2(l) = (\omega,\ \text{private } bty,\ 1,\ \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) \\ \text{DecodeVal}(\text{private } bty,\ 1,\ \omega) = n_1 \qquad \text{OutputValue}(x,\ n, n_1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcoutput}(e_1,\ e_2)) \Downarrow_{out3}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking SMC Output Public Array

$$\frac{\begin{array}{c}\text{Label}(e_2,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \\ \text{Label}(e_3,\gamma) = \text{public} \qquad (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \\ (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, e_3) \Downarrow_e^t (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, n_1) \qquad \gamma(x) = (l, \text{public const } bty*) \\ \sigma_3(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ \text{public } bty,\ 1] \\ \sigma_3(l_1) = (\omega_1,\ \text{public } bty,\ n_1,\ \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n_1)) \\ \text{DecodeVal}(\text{public } bty,\ n_1,\ \omega_1) = [m_0,\ ...,\ m_{n_1}] \qquad \text{OutputArray}(x,\ n,\ [m_0,\ ...,\ m_{n_1}])\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcoutput}(e_1,\ e_2,\ e_3)) \Downarrow_{out1}^t (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking SMC Output Private Array

$$\frac{\begin{array}{c}\text{Label}(e_2,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \\ \text{Label}(e_3,\gamma) = \text{public} \qquad (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \\ (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, e_3) \Downarrow_e^t (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, n_1) \qquad \gamma(x) = (l, \text{private const } bty*) \\ \sigma_3(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ \text{private } bty,\ 1] \\ \sigma_3(l_1) = (\omega_1,\ \text{private } bty,\ n_1,\ \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n_1)) \\ \text{DecodeVal}(\text{private } bty,\ n_2,\ \omega_1) = [m_0,\ ...,\ m_{n_1}] \qquad \text{OutputArray}(x,\ n,\ [m_0,\ ...,\ m_{n_1}])\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcoutput}(e_1,\ e_2,\ e_3)) \Downarrow_{out4}^t (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Figure 4.6: Location-tracking SMC^2 semantic rules for output.

For Location-tracking SMC^2 pointers, the rules remain nearly identical. We add rule Location-tracking Private Pointer Declaration (Inside a Private - Conditioned If Else Branch), allowing local private pointers within branches. These are added to the tracking structure for local locations, as such locations are not intended to persist beyond the scope of the branch and do not need to be tracked. For the sake of space, as with pre-increment operations, we do not show the versions of dereference reading and dereference writing that are not *well-aligned*, as these versions do not show anything new beyond the *well-aligned* versions. We are still able to handle these versions of the rules within the noninterference proof, and the interested reader can look back on the Basic SMC^2 versions of these rules to better visualize the differences between such rules.

Location-tracking Pre-Increment Public Variable

$$\frac{\begin{array}{c}\text{acc} = 0 \qquad \gamma(x) = (l,\ \text{public } bty) \qquad \sigma(l) = (\omega,\ \text{public } bty,\ 1,\ \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1)) \\ \text{DecodeVal}(\text{public } bty,\ 1,\ \omega) = v \qquad v_1 =_{\text{public}} v +_{\text{public}} 1 \\ \text{T_UpdateVal}(\sigma,\ l,\ v_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{public } bty) = (\sigma_1,\ \Delta)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++\ x)\ \Downarrow^{t}_{pin}\ (\gamma,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v_1)}$$

Location-tracking Pre-Increment Private Variable

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{private } bty) \qquad \sigma(l) = (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) \\ \text{DecodeVal}(\text{private } bty, 1, \omega) = v_1 \qquad (bty = \text{int}) \vee (bty = \text{float}) \qquad v_2 =_{\text{private}} v_1 +_{\text{private}} \text{encrypt}(1) \\ \text{T_UpdateVal}(\sigma, l, v_2, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty) = (\sigma_1, \Delta_1)\end{array}}{(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x)\ \Downarrow^{t}_{pin1}\ (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_2)}$$

Location-tracking Pre-Increment Public Pointer Single Location

$$\frac{\begin{array}{c}\gamma(x) = (l,\ \text{public } bty*) \qquad \sigma(l) = (\omega,\ \text{public } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1] \qquad ((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty), \sigma) \\ \text{T_UpdatePtr}(\sigma,\ (l, 0),\ [1,\ [(l_2, \mu_2)],\ [1],\ 1],\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{public } bty*) = (\sigma_1,\ \Delta_1, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++\ x)\ \Downarrow^{t}_{pin2}\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_2, \mu_2))}$$

Location-tracking Pre-Increment Private Pointer Single Location

$$\frac{\begin{array}{c}\gamma(x) = (l,\ \text{private } bty*) \qquad \sigma(l) = (\omega,\ \text{private } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1] \qquad ((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{private } bty), \sigma) \\ \text{T_UpdatePtr}(\sigma,\ (l, 0),\ [1,\ [(l_2, \mu_2)],\ [1],\ 1],\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{private } bty*) = (\sigma_1,\ \Delta_1, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++\ x)\ \Downarrow^{t}_{pin6}\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_2, \mu_2))}$$

Location-tracking Pre-Increment Public Pointer Higher Level Indirection Single Location

$$\frac{\begin{array}{c}\gamma(x) = (l,\ \text{public } bty*) \qquad \sigma(l) = (\omega,\ \text{public } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i] \qquad ((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty*), \sigma) \\ i > 1 \qquad \text{T_UpdatePtr}(\sigma,\ (l, 0),\ [1,\ [(l_2, \mu_2)],\ [1],\ i],\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{public } bty) = (\sigma_1,\ \Delta_1, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++\ x)\ \Downarrow^{t}_{pin3}\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_2, \mu_2))}$$

Location-tracking Pre-Increment Private Pointer Higher Level Indirection Single Location

$$\frac{\begin{array}{c}\gamma(x) = (l,\ \text{private } bty*) \qquad \sigma(l) = (\omega,\ \text{private } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i] \qquad ((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{private } bty*), \sigma) \\ i > 1 \qquad \text{T_UpdatePtr}(\sigma,\ (l, 0),\ [1,\ [(l_2, \mu_2)],\ [1],\ i],\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{private } bty) = (\sigma_1,\ \Delta_1, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++\ x)\ \Downarrow^{t}_{pin7}\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_2, \mu_2))}$$

Location-tracking Pre-Increment Pointer Multiple Locations

$$\frac{\begin{array}{c}\gamma(x) = (l,\ \text{private } bty*) \qquad \sigma(l) = (\omega,\ \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ 1] \qquad \text{IncrementList}(\bar{l},\ \tau(\text{private } bty), \sigma) = (\bar{l}', 1) \\ \text{T_UpdatePtr}(\sigma,\ (l, 0),\ [n,\ \bar{l},\ \bar{j},\ 1],\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{private } bty*) = (\sigma_1,\ \Delta_1, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++\ x)\ \Downarrow^{t}_{pin4}\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [n,\ \bar{l}',\ \bar{j},\ 1])}$$

Location-tracking Pre-Increment Pointer Higher Level Indirection Multiple Locations

$$\frac{\begin{array}{c}\gamma(x) = (l,\ \text{private } bty*) \qquad \sigma(l) = (\omega,\ \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i] \qquad \text{IncrementList}(\bar{l},\ \tau(\text{private } bty*), \sigma) = (\bar{l}', 1) \\ \text{T_UpdatePtr}(\sigma,\ (l, 0),\ [\alpha,\ \bar{l}',\ \bar{j},\ i],\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{private } bty*) = (\sigma_1,\ \Delta_1, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++\ x)\ \Downarrow^{t}_{pin5}\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [\alpha,\ \bar{l}',\ \bar{j},\ i])}$$

Figure 4.7: Location-tracking SMC^2 semantic rules for the pre-increment operator.

Location-tracking Public Addition

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 +_{\text{public}} n_2 = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 + e_2)\ \Downarrow_{bp}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private Addition

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 +_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 + e_2)\ \Downarrow_{bp1}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public - Private Addition

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad \text{encrypt}(n_1) +_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 + e_2)\ \Downarrow_{bp2}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private - Public Addition

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 +_{\text{private}} \text{encrypt}(n_2) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 + e_2)\ \Downarrow_{bp3}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public Subtraction

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 -_{\text{public}} n_2 = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 - e_2)\ \Downarrow_{bs}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private Subtraction

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 -_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 - e_2)\ \Downarrow_{bs1}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private - Public Subtraction

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 -_{\text{private}} \text{encrypt}(n_2) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 - e_2)\ \Downarrow_{bs2}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public - Private Subtraction

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad \text{encrypt}(n_1) -_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 - e_2)\ \Downarrow_{bs3}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Figure 4.8: Location-tracking SMC^2 semantics for addition and subtraction.

Location-tracking Public Multiplication

$$\frac{\begin{array}{ll}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} & (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) & n_1 \cdot_{\text{public}} n_2 = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 \cdot e_2)\ \Downarrow_{bm}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private Multiplication

$$\frac{\begin{array}{ll}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} & (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) & n_1 \cdot_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 \cdot e_2)\ \Downarrow_{bm1}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public - Private Multiplication

$$\frac{\begin{array}{ll}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) & (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) & \text{encrypt}(n_1) \cdot_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 \cdot e_2)\ \Downarrow_{bm2}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private - Public Multiplication

$$\frac{\begin{array}{ll}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) & (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) & n_1 \cdot_{\text{private}} \text{encrypt}(n_2) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 \cdot e_2)\ \Downarrow_{bm3}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public Division

$$\frac{\begin{array}{ll}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} & (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) & n_1 \div_{\text{public}} n_2 = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 \div e_2)\ \Downarrow_{bd}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private Division

$$\frac{\begin{array}{ll}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} & (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) & n_1 \div_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 \div e_2)\ \Downarrow_{bd1}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public - Private Division

$$\frac{\begin{array}{ll}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) & (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) & \text{encrypt}(n_1) \cdot_{\text{private}} n_2 = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 \cdot e_2)\ \Downarrow_{bd2}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private - Public Division

$$\frac{\begin{array}{ll}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) & (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) & n_1 \div_{\text{private}} \text{encrypt}(n_2) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 \div e_2)\ \Downarrow_{bd3}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Figure 4.9: Location-tracking SMC^2 semantics for multiplication and division.

Location-tracking Public Less Than True

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \quad n_1 <_{\text{public}} n_2\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 < e_2)\ \Downarrow_{ltt}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ 1)}$$

Location-tracking Private Less Than True

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \quad n_1 <_{\text{private}} n_2 \quad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 < e_2)\ \Downarrow_{ltt1}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public - Private Less Than True

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \quad \text{encrypt}(n_1) <_{\text{private}} n_2 \quad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 < e_2)\ \Downarrow_{ltt2}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private - Public Less Than True

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \quad n_1 <_{\text{private}} \text{encrypt}(n_2) \quad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 < e_2)\ \Downarrow_{ltt3}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public Less Than False

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \quad n_1 >=_{\text{public}} n_2\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 < e_2)\ \Downarrow_{ltf}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ 0)}$$

Location-tracking Private Less Than False

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \quad n_1 >=_{\text{private}} n_2 \quad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 < e_2)\ \Downarrow_{ltf1}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public - Private Less Than False

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \quad \text{encrypt}(n_1) >=_{\text{private}} n_2 \quad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 < e_2)\ \Downarrow_{ltf2}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private - Public Less Than False

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \quad n_1 >=_{\text{private}} \text{encrypt}(n_2) \quad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 < e_2)\ \Downarrow_{ltf3}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Figure 4.10: Location-tracking SMC^2 semantics for less than comparisons.

Location-tracking Public Equal To True

$$\frac{\begin{array}{c}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 =_{\text{public}} n_2\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 == e_2)\ \Downarrow_{eqt}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ 1)}$$

Location-tracking Private Equal To True

$$\frac{\begin{array}{c}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 =_{\text{private}} n_2 \qquad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 == e_2)\ \Downarrow_{eqt1}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public - Private Equal To True

$$\frac{\begin{array}{c}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \\ (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad \text{encrypt}(n_1) =_{\text{private}} n_2 \qquad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 == e_2)\ \Downarrow_{eqt2}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private - Public Equal To True

$$\frac{\begin{array}{c}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 =_{\text{private}} \text{encrypt}(n_2) \qquad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 == e_2)\ \Downarrow_{eqt3}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public Equal To False

$$\frac{\begin{array}{c}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 \neq_{\text{public}} n_2\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 == e_2)\ \Downarrow_{eqf}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ 0)}$$

Location-tracking Private Equal To False

$$\frac{\begin{array}{c}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 \neq_{\text{private}} n_2 \qquad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 == e_2)\ \Downarrow_{eqf1}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public - Private Equal To False

$$\frac{\begin{array}{c}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad \text{encrypt}(n_1) \neq_{\text{private}} n_2 \qquad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 == e_2)\ \Downarrow_{eqf2}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private - Public Equal To False

$$\frac{\begin{array}{c}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 \neq_{\text{private}} \text{encrypt}(n_2) \qquad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 == e_2)\ \Downarrow_{eqf3}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Figure 4.11: Location-tracking SMC^2 semantics for equal to comparisons.

Location-tracking Public Not Equal To True

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 \neq_{\text{public}} n_2\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 ! = e_2)\ \Downarrow_{net}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ 1)}$$

Location-tracking Private Not Equal To True

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 \neq_{\text{private}} n_2 \qquad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 ! = e_2)\ \Downarrow_{net1}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public - Private Not Equal To True

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad \text{encrypt}(n_1) \neq_{\text{private}} n_2 \qquad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 ! = e_2)\ \Downarrow_{net2}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private - Public Not Equal To True

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 \neq_{\text{private}} \text{encrypt}(n_2) \qquad \text{encrypt}(1) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 ! = e_2)\ \Downarrow_{net3}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public Not Equal To False

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 =_{\text{public}} n_2\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 ! = e_2)\ \Downarrow_{nef}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ 0)}$$

Location-tracking Private Not Equal To False

$$\frac{\begin{array}{l}\text{Label}(e_1,\gamma) = \text{Label}(e_2,\gamma) = \text{private} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 =_{\text{private}} n_2 \qquad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 ! = e_2)\ \Downarrow_{nef1}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Public - Private Not Equal To False

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{public}) \wedge (\text{Label}(e_2,\gamma) = \text{private}) \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad \text{encrypt}(n_1) =_{\text{private}} n_2 \qquad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 ! = e_2)\ \Downarrow_{nef2}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Location-tracking Private - Public Not Equal To False

$$\frac{\begin{array}{l}(\text{Label}(e_1,\gamma) = \text{private}) \wedge (\text{Label}(e_2,\gamma) = \text{public}) \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_2) \qquad n_1 =_{\text{private}} \text{encrypt}(n_2) \qquad \text{encrypt}(0) = n_3\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1 ! = e_2)\ \Downarrow_{nef3}^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_3)}$$

Figure 4.12: Location-tracking SMC^2 semantics for not equal to comparisons.

Location-tracking Statement Sequencing

$$\frac{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1)\ \Downarrow^t_s\ (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \quad (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ s_2)\ \Downarrow^t_s\ (\gamma_2,\ \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ v)}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1;\ s_2)\ \Downarrow^t_{ss}\ (\gamma_2,\ \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ v)}$$

Location-tracking Statement Block

$$\frac{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s)\ \Downarrow^t_s\ (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \{s\})\ \Downarrow^t_{sb}\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Parentheses

$$\frac{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow^t_e\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (e))\ \Downarrow^t_{ep}\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)}$$

Location-tracking Declaration Assignment

$$\frac{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x)\ \Downarrow^t_s\ (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \quad (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ x = e)\ \Downarrow^t_s\ (\gamma_1,\ \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ var = e)\ \Downarrow^t_{ds}\ (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Array Declaration Assignment

$$\frac{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x[e])\ \Downarrow^t_s\ (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \quad (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ x = e)\ \Downarrow^t_s\ (\gamma_1,\ \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x[e_1] = e_2)\ \Downarrow^t_{das}\ (\gamma_1,\ \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Function Declaration

$$\frac{(\text{acc} = 0) \wedge (\text{bid} = none) \quad l = \phi() \quad \text{GetFunTypeList}(\overline{p}) = \overline{ty} \quad \gamma_1 = \gamma[x \rightarrow (l,\ \overline{ty} \rightarrow ty)] \quad \sigma_1 = \sigma[l \rightarrow (\text{NULL},\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)})]}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p}))\ \Downarrow^t_{df}\ (\gamma_1,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Function Definition

$$\frac{(\text{acc} = 0) \wedge (\text{bid} = none) \quad x \notin \gamma \quad l = \phi() \quad \text{GetFunTypeList}(\overline{p}) = \overline{ty} \quad \gamma_1 = \gamma[x \rightarrow (l,\ \overline{ty} \rightarrow ty)] \quad \text{CheckPublicEffects}(s,\ x,\ \gamma, \sigma) = n \quad \text{EncodeFun}(s,\ n,\ \overline{p}) = \omega \quad \sigma_1 = \sigma[l \rightarrow (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)})]}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})\{s\})\ \Downarrow^t_{fpd}\ (\gamma_1,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Pre-Declared Function Definition

$$\frac{(\text{acc} = 0) \wedge (\text{bid} = none) \quad x \in \gamma \quad \gamma(x) = (l,\ \overline{ty} \rightarrow ty) \quad \text{CheckPublicEffects}(s,\ x,\ \gamma, \sigma) = n \quad \sigma = \sigma_1[l \rightarrow (\text{NULL},\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)})] \quad \text{EncodeFun}(s,\ n,\ \overline{p}) = \omega \quad \sigma_2 = \sigma_1[l \rightarrow (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)})]}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})\{s\})\ \Downarrow^t_{fd}\ (\gamma,\ \sigma_2,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Function Call Without Public Side Effects

$$\frac{\gamma(x) = (l,\ \overline{ty} \rightarrow ty) \quad \sigma(l) = (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)}) \quad \text{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1 \quad \text{DecodeFun}(\omega) = (s,\ n,\ \overline{p}) \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1)\ \Downarrow^t_s\ (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \quad n = 0 \quad (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ s)\ \Downarrow^t_s\ (\gamma_2,\ \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ \text{skip})}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x(\overline{e}))\ \Downarrow^t_{fc1}\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{NULL})}$$

Location-tracking Function Call With Public Side Effects

$$\frac{\gamma(x) = (l,\ \overline{ty} \rightarrow ty) \quad \sigma(l) = (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)}) \quad \text{DecodeFun}(\omega) = (s,\ n,\ \overline{p}) \quad n = 1 \quad \text{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1 \quad (\text{acc} = 0) \wedge (\text{bid} = none) \quad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1)\ \Downarrow^t_s\ (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \quad (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ s)\ \Downarrow^t_s\ (\gamma_2,\ \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ \text{skip})}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x(\overline{e}))\ \Downarrow^t_{fc}\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{NULL})}$$

Figure 4.13: Location-tracking SMC^2 semantic rules for sequencing and functions

Location-tracking Public Malloc

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \qquad (\text{bid} = \mathit{none}) \wedge (\text{acc} = 0) \\ \text{Label}(\gamma, e) = \text{public} \qquad l = \phi() \qquad \sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, n)\big)\big]\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{malloc}(e)) \Downarrow_{mal}^t (\gamma,\ \sigma_2,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0))}$$

Location-tracking Private Malloc

$$\frac{\begin{array}{c}\text{Label}(\gamma, e) = \text{public} \qquad (\text{bid} = \mathit{none}) \wedge (\text{acc} = 0) \qquad (\mathit{ty} = \text{private int}*) \vee (\mathit{ty} = \text{private float}*) \\ (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \qquad l = \phi() \\ \sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL},\ \text{void}*,\ n \cdot \tau(\mathit{ty}),\ \text{PermL}(\text{Freeable}, \mathit{ty}, \text{private}, n \cdot \tau(\mathit{ty}))\big)\big]\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{pmalloc}(e,\ \mathit{ty})) \Downarrow_{malp}^t (\gamma,\ \sigma_2,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0))}$$

Location-tracking Public Free

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \qquad \gamma(x) = (l,\ \text{public}\ \mathit{bty}*) \\ (\text{bid} = \mathit{none}) \wedge (\text{acc} = 0) \qquad \text{Free}(\sigma_1,\ l, \gamma) = \sigma_2\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{free}(e))\ \Downarrow_{fre}^t (\gamma,\ \sigma_2,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Free

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \qquad \gamma(x) = (l,\ \text{private}\ \mathit{bty}*) \\ (\text{bid} = \mathit{none}) \wedge (\text{acc} = 0) \qquad (\mathit{bty} = \text{int}) \vee (\mathit{bty} = \text{float}) \qquad \text{PFree}(\gamma, \sigma_1,\ l) = (\sigma_2, \overline{l}, \overline{j})\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{pfree}(e))\ \Downarrow_{frep}^t (\gamma,\ \sigma_2,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Cast Private Location

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0)) \\ (\mathit{ty} = \text{private int}*) \vee (\mathit{ty} = \text{private float}*) \vee (\mathit{ty} = \text{int}*) \vee (\mathit{ty} = \text{float}*) \\ \sigma_1 = \sigma_2\big[l \rightarrow \big(\omega,\ \text{void},\ n,\ \text{PermL}(\text{Freeable}, \mathit{ty}, \text{private}, n)\big)\big] \\ \sigma_3 = \sigma_2\big[l \rightarrow \big(\omega,\ \mathit{ty},\ \frac{n}{\tau(\mathit{ty})},\ \text{PermL}(\text{Freeable}, \mathit{ty}, \text{private}, \frac{n}{\tau(\mathit{ty})})\big)\big]\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (\mathit{ty})\ e) \Downarrow_{cl1}^t (\gamma,\ \sigma_3,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0))}$$

Location-tracking Cast Public Location

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0)) \\ \text{acc} = 0 \qquad \sigma_1 = \sigma_2\big[l \rightarrow \big(\omega,\ \text{void},\ n,\ \text{PermL}(\text{Freeable}, \text{void}, \text{public}, n)\big)\big] \\ (\mathit{ty} = \text{public}\ \mathit{bty}*) \vee (\mathit{ty} = \text{char}*) \qquad \sigma_3 = \sigma_2\big[l \rightarrow \big(\omega,\ \mathit{ty},\ \frac{n}{\tau(\mathit{ty})},\ \text{PermL}(\text{Freeable}, \mathit{ty}, \text{public}, \frac{n}{\tau(\mathit{ty})})\big)\big]\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (\mathit{ty})\ e) \Downarrow_{cl}^t (\gamma,\ \sigma_3,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0))}$$

Location-tracking Cast Public Value

$$\frac{\begin{array}{c}\text{Label}(e, \gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \\ (\mathit{ty} = \text{public int}) \vee (\mathit{ty} = \text{public float}) \qquad n_1 = \text{Cast}(\text{public},\ \mathit{ty},\ n)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (\mathit{ty})\ e)\ \Downarrow_{cv}^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1)}$$

Location-tracking Cast Private Value

$$\frac{\begin{array}{c}\text{Label}(e, \gamma) = \text{private} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \\ (\mathit{ty} = \text{private int}) \vee (\mathit{ty} = \text{private float}) \vee (\mathit{ty} = \text{int}) \vee (\mathit{ty} = \text{float}) \qquad n_1 = \text{Cast}(\text{private},\ \mathit{ty},\ n)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (\mathit{ty})\ e)\ \Downarrow_{cv1}^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1)}$$

Location-tracking Address Of

$$\frac{\gamma(x) = (l,\ \mathit{ty})}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \&x)\ \Downarrow_{loc}^t (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0))}$$

Location-tracking Size of type

$$\frac{n = \tau(\mathit{ty})}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{sizeof}(\mathit{ty}))\ \Downarrow_{ty}^t (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)}$$

Figure 4.14: Location-tracking SMC^2 semantic rules for memory allocation, deallocation, and casting.

Location-tracking Public Array Declaration

$$\frac{\begin{array}{c}((ty = \text{public } bty) \wedge ((bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{int}))) \vee (ty = \text{char}) \qquad (\text{acc} = 0) \wedge (\text{bid} = none) \\ l = \phi() \qquad l_1 = \phi() \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \qquad n > 0 \\ \gamma_1 = \gamma[x\ \rightarrow\ (l,\ \text{public const } bty*)] \qquad \omega = \text{EncodePtr}(\text{public const } bty*,\ [1,\ [(l_1, 0)],\ [1],\ 1]) \\ \text{Label}(e_1, \gamma) = \text{public} \qquad \sigma_2 = \sigma_1[l\ \rightarrow\ (\omega,\ \text{public const } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))] \\ \omega_1 = \text{EncodeVal}(\text{public } bty, \text{NULL}) \qquad \sigma_3 = \sigma_2[l_1\ \rightarrow\ (\omega_1,\ \text{public } bty,\ n,\ \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))]\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x[e]) \Downarrow_{da}^t (\gamma_1,\ \sigma_3,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Array Declaration

$$\frac{\begin{array}{c}\text{Label}(e_1, \gamma) = \text{public} \qquad ((ty = \text{private } bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float})) \qquad l = \phi() \qquad l_1 = \phi() \\ (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t (\gamma,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \qquad n > 0 \qquad \gamma_1 = \gamma[x\ \rightarrow\ (l,\ \text{private const } bty*)] \\ \omega = \text{EncodePtr}(\text{private const } bty*,\ [1,\ [(l_1, 0)],\ [1],\ 1]) \qquad \omega_1 = \text{EncodeVal}(\text{private } bty, \text{NULL}) \\ \sigma_2 = \sigma_1[l\ \rightarrow\ (\omega,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))] \\ (\text{acc} = 0) \wedge (\text{bid} = none) \qquad \sigma_3 = \sigma_2[l_1\ \rightarrow\ (\omega_1,\ \text{private } bty,\ n,\ \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))]\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x[e]) \Downarrow_{da1}^t (\gamma_1,\ \sigma_3,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Array Declaration (Inside a Private - Conditioned If Else Branch)

$$\frac{\begin{array}{c}\text{Label}(e, \gamma) = \text{public} \qquad ((ty = \text{private } bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float})) \\ (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow_e^t (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n) \qquad n > 0 \qquad l = \phi() \qquad l_1 = \phi() \\ \gamma_1 = \gamma[x \rightarrow (l, \text{private const } bty*)] \qquad \omega = \text{EncodePtr}(\text{private const } bty*, [1, [l_1], [1], 1]) \\ \sigma_2 = \sigma_1[l \rightarrow (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))] \\ \sigma_3 = \sigma_2[l_1 \rightarrow (\text{NULL}, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))] \\ (\text{acc} > 0) \wedge ((\text{bid} = then) \vee (\text{bid} = else)) \qquad \overline{\chi}_1 = l :: l_1 :: \overline{\chi}[\text{acc}]\end{array}}{(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow_{da2}^t (\gamma_1, \sigma_3, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})}$$

Location-tracking Public Array Write Entire Array

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [v_0, ..., v_{n_e-1}]) \\ \forall v_m \in [v_0, ..., v_{n_e-1}].v_m \neq \text{skip} \qquad \gamma(x) = (l,\ \text{public const } bty*) \qquad n_e = n \qquad \text{Label}(e, \gamma) = \text{public} \\ \sigma_1(l) = (\omega,\ \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n)) \\ \text{T_UpdateVal}(\sigma_1,\ l_1,\ [v_0, ..., v_{n_e-1}],\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{public } bty) = (\sigma_2,\ \Delta_2)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow_{wa5}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Array Write Entire Private Array

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [v_0, ..., v_{n_e-1}]) \\ \forall v_m \in [v_0, ..., v_{n_e-1}].v_m \neq \text{skip} \qquad \gamma(x) = (l,\ \text{private const } bty*) \qquad n_e = n \qquad \text{Label}(e, \gamma) = \text{private} \\ \sigma_1(l) = (\omega,\ \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) \\ \text{T_UpdateVal}(\sigma_1,\ l_1,\ [v_0, ..., v_{n_e-1}],\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{private } bty) = (\sigma_2,\ \Delta_2)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow_{wa6}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Array Write Entire Public Array

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [v_0, ..., v_{n_e-1}]) \\ \forall v_m \in [v_0, ..., v_{n_e-1}].\ (v'_m = \text{encrypt}(v_m)) \wedge (v_m \neq \text{skip}) \qquad \gamma(x) = (l,\ \text{private const } bty*) \qquad n_e = n \\ \sigma_1(l) = (\omega,\ \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \qquad \text{Label}(e, \gamma) = \text{public} \\ \sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n)) \\ \text{T_UpdateVal}(\sigma_1,\ l_1,\ [v'_0, ..., v_{n'_e-1}],\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{private } bty) = (\sigma_2,\ \Delta_2)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow_{wa7}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Figure 4.15: Location-tracking SMC^2 semantic rules for array declarations and writing an entire array.

Location-tracking Public Array Read Public Index

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \qquad \gamma(x) = (l,\ \text{public const}\ bty*) \\ \sigma_1(l) = (\omega,\ \text{public const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public const}\ bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public const}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \qquad \text{Label}(e, \gamma) = \text{public} \\ \sigma_1(l_1) = (\omega_1, \text{public}\ bty, n, \text{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, n)) \\ 0 \leq i \leq n-1 \qquad \text{DecodeVal}(\text{public}\ bty,\ n,\ \omega_1) = [v_0, ..., v_{n-1}]\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e])\ \Downarrow_{ra}^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v_i)}$$

Location-tracking Private Array Read Public Index

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \qquad \gamma(x) = (l,\ \text{private const}\ bty*) \\ \sigma_1(l) = (\omega,\ \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private const}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \qquad 0 \leq i \leq n-1 \\ \sigma_1(l_1) = (\omega_1, \text{private}\ bty, n, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n)) \\ \text{Label}(e, \gamma) = \text{public} \qquad \text{DecodeVal}(\text{private}\ bty,\ n,\ \omega_1) = [v_0, ..., v_{n-1}]\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e])\ \Downarrow_{ra3}^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v_i)}$$

Location-tracking Private Array Read Private Index

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \qquad \gamma(x) = (l,\ \text{private const}\ bty*) \\ \sigma_1(l) = (\omega,\ \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1)) \\ \text{Label}(e, \gamma) = \text{private} \qquad (bty = \text{int}) \vee (bty = \text{float}) \\ \text{DecodePtr}(\text{private const}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \sigma_1(l_1) = (\omega_1, \text{private}\ bty, n, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n)) \qquad v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge v_m \\ \text{DecodeVal}(\text{private}\ bty,\ n,\ \omega_2) = [v_0,\ ...,\ v_{n-1}]\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e])\ \Downarrow_{ra1}^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)}$$

Location-tracking Public Array Read Private Index

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \qquad \gamma(x) = (l,\ \text{public const}\ bty*) \\ \sigma_1(l) = (\omega,\ \text{public const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public const}\ bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public const}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \qquad \text{Label}(e, \gamma) = \text{private} \\ \sigma_1(l_1) = (\omega_1, \text{public}\ bty, n, \text{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, n)) \\ \text{DecodeVal}(\text{public}\ bty,\ n,\ \omega_1) = [v_0,\ ...,\ v_{n-1}] \qquad v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge \text{encrypt}(v_m) \\ (bty = \text{int}) \vee (bty = \text{float})\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e])\ \Downarrow_{ra2}^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)}$$

Location-tracking Public Array Read Entire Array

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{public const}\ bty*) \qquad \sigma_1(l) = (\omega, \text{public const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public const}\ bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public const}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \qquad \text{Label}(e, \gamma) = \text{public} \\ \sigma(l_1) = (\omega_1, \text{public}\ bty, n, \text{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, n)) \\ \text{DecodeVal}(\text{public}\ bty, n, \omega_1) = [v_0, ..., v_{n-1}]\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x)\ \Downarrow_{ra4}^t\ (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [v_0, ..., v_{n-1}])}$$

Location-tracking Private Array Read Entire Array

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{private const}\ bty*) \quad \sigma_1(l) = (\omega, \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private const}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \qquad \text{Label}(e, \gamma) = \text{private} \\ \sigma(l_1) = (\omega_1, \text{private}\ bty, n, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n)) \\ \text{DecodeVal}(\text{private}\ bty, n, \omega_1) = [v_0, ..., v_{n-1}]\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x)\ \Downarrow_{ra5}^t\ (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [v_0, ..., v_{n-1}])}$$

Figure 4.16: Location-tracking SMC^2 semantic rules for reading from an array at a public or private index and reading an entire array.

Location-tracking Public Array Write Public Value Public Index

$$\frac{\begin{array}{c}\text{Label}(e_1,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \qquad \gamma(x) = (l,\ \text{public const}\ bty*) \\ v \neq \text{skip} \qquad \sigma_2(l) = (\omega, \text{public const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public const}\ bty*, \text{public}, 1)) \\ \text{Label}(e_2,\gamma) = \text{public} \qquad \text{DecodePtr}(\text{public const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \sigma_2(l_1) = (\omega_1, \text{public}\ bty, n, \text{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, n)) \\ 0 \leq i \leq n-1 \qquad \text{DecodeVal}(\text{public}\ bty, n, \omega_1) = [v_0, ..., v_{n-1}] \qquad [v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\Big(\frac{v}{v_i}\Big) \\ \text{T_UpdateVal}(\sigma_2,\ l_1,\ [v'_0, ..., v'_{n-1}],\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{public}\ bty) = (\sigma_3,\ \Delta_3)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2)\ \Downarrow_{wa}^t\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Array Write Private Value Public Index

$$\frac{\begin{array}{c}\text{Label}(e_1,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \qquad \gamma(x) = (l,\ \text{private const}\ bty*) \\ \text{Label}(e_2,\gamma) = \text{private} \qquad \sigma_2(l) = (\omega, \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1)) \\ v \neq \text{skip} \qquad \text{DecodePtr}(\text{private const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \sigma_2(l_1) = (\omega_1, \text{private}\ bty, n, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n)) \\ 0 \leq i \leq n-1 \qquad \text{DecodeVal}(\text{private}\ bty, n, \omega_1) = [v_0, ..., v_{n-1}] \qquad [v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\Big(\frac{v}{v_i}\Big) \\ \text{T_UpdateVal}(\sigma_2,\ l_1,\ [v'_0, ..., v'_{n-1}],\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{private}\ bty) = (\sigma_3,\ \Delta_3)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2)\ \Downarrow_{wa4}^t\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Array Write Public Value Public Index

$$\frac{\begin{array}{c}\text{Label}(e_1,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \qquad \gamma(x) = (l,\ \text{private const}\ bty*) \\ \sigma_2(l) = (\omega, \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1)) \\ v \neq \text{skip} \qquad \text{DecodePtr}(\text{private const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \sigma_2(l_1) = (\omega_1, \text{private}\ bty, n, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n)) \qquad (bty = \text{int}) \vee (bty = \text{float}) \\ \text{DecodeVal}(\text{private}\ bty, n, \omega_1) = [v_0, ..., v_{n-1}] \qquad 0 \leq i \leq n-1 \qquad [v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\Big(\frac{v}{v_i}\Big) \\ \text{T_UpdateVal}(\sigma_2,\ l_1,\ [v'_0, ..., v'_{n-1}],\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{private}\ bty) = (\sigma_3,\ \Delta_3) \qquad \text{Label}(e_2,\gamma) = \text{public}\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2)\ \Downarrow_{wa1}^t\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Array Write Public Value Private Index

$$\frac{\begin{array}{c}\text{Label}(e_1,\gamma) = \text{private} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \qquad \gamma(x) = (l,\ \text{private const}\ bty*) \\ \sigma_2(l) = (\omega, \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1)) \qquad \text{Label}(e_2,\gamma) = \text{public} \\ v \neq \text{skip} \qquad \text{DecodePtr}(\text{private const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \sigma_2(l_1) = (\omega_1, \text{private}\ bty, n, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n)) \qquad (bty = \text{int}) \vee (bty = \text{float}) \\ \text{DecodeVal}(bty,\ n,\ \omega_1) = [v_0, ..., v_{n-1}] \qquad \forall v_m \in [v_0, ..., v_{n-1}] \quad v'_m = ((i = m) \wedge v)\ \vee\ (\neg(i = m) \wedge v_m) \\ \text{T_UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{private}\ bty) = (\sigma_3, \Delta_3)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2)\ \Downarrow_{wa2}^t\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Array Write Private Value Private Index

$$\frac{\begin{array}{c}\text{Label}(e_1,\gamma) = \text{private} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \qquad \gamma(x) = (l,\ \text{private const}\ bty*) \\ \sigma_2(l) = (\omega, \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1)) \\ (bty = \text{int}) \vee (bty = \text{float}) \qquad v \neq \text{skip} \qquad \text{DecodePtr}(\text{private const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \sigma_2(l_1) = (\omega_1, \text{private}\ bty, n, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n)) \\ \text{DecodeVal}(bty,\ n,\ \omega_1) = [v_0, ..., v_{n-1}] \qquad \forall v_m \in [v_0, ..., v_{n-1}] \quad v'_m = ((i = m) \wedge v)\ \vee\ (\neg(i = m) \wedge v_m) \\ \text{T_UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}],\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{private}\ bty) = (\sigma_3,\ \Delta_3) \qquad \text{Label}(e_2,\gamma) = \text{private}\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2)\ \Downarrow_{wa3}^t\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Figure 4.17: Location-tracking SMC^2 semantic rules for writing to an array at an index.

Location-tracking Public Array Read Out-of-Bounds Public Index

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \qquad \gamma(x) = (l,\ \text{public const}\ bty*) \\ \sigma_1(l) = (\omega,\ \text{public const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public const}\ bty*, \text{public}, 1)) \qquad \text{Label}(e, \gamma) = \text{public} \\ \text{DecodePtr}(\text{public const}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \qquad (i < 0) \vee (i \geq n) \\ \sigma_1(l_1) = (\omega_1, \text{public}\ bty, n, \text{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, n)) \qquad \text{ReadOOB}(i, n, l_1, \text{public}\ bty, \sigma_1) = (v, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e])\ \Downarrow_{rao}^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)}$$

Location-tracking Private Array Read Out-of-Bounds Public Index

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \qquad \gamma(x) = (l,\ \text{private const}\ bty*) \\ \sigma_1(l) = (\omega,\ \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1)) \qquad \text{Label}(e, \gamma) = \text{public} \\ \text{DecodePtr}(\text{private const}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \qquad (i < 0) \vee (i \geq n) \\ \sigma_1(l_1) = (\omega_1, \text{private}\ bty, n, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n)) \quad \text{ReadOOB}(i, n, l_1, \text{private}\ bty, \sigma_1) = (v, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e])\ \Downarrow_{rao1}^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)}$$

Location-tracking Public Array Write Out-of-Bounds Public Index Public Value

$$\frac{\begin{array}{c}\text{Label}(e_1, \gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \qquad \gamma(x) = (l, \text{public const}\ bty*) \\ \text{Label}(e_2, \gamma) = \text{public} \qquad \sigma_2(l) = (\omega, \text{public const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public const}\ bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \sigma_2(l_1) = (\omega_1,\ \text{public}\ bty,\ n,\ \text{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, n)) \qquad v \neq \text{skip} \\ (i < 0) \vee (i \geq n) \qquad \text{T_WriteOOB}(v,\ i,\ n,\ l_1,\ \text{public}\ bty,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc}) = (\sigma_3,\ \Delta_3, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2)\ \Downarrow_{wao}^t\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Array Write Out-of-Bounds Public Index Private Value

$$\frac{\begin{array}{c}\text{Label}(e_1, \gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \qquad \gamma(x) = (l, \text{private const}\ bty*) \\ \text{Label}(e_2, \gamma) = \text{private} \qquad \sigma_2(l) = (\omega, \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1)) \\ v \neq \text{skip} \qquad \text{DecodePtr}(\text{private const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \sigma_2(l_1) = (\omega_1,\ \text{private}\ bty,\ n,\ \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n)) \\ (i < 0) \vee (i \geq n) \qquad \text{T_WriteOOB}(v,\ i,\ n,\ l_1,\ \text{private}\ bty,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc}) = (\sigma_3,\ \Delta_3, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2)\ \Downarrow_{wao2}^t\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Array Write Out-of-Bounds Public Value Public Index

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)\ \Downarrow_e^t\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \qquad \gamma(x) = (l, \text{private const}\ bty*) \\ v \neq \text{skip} \qquad \sigma_2(l) = (\omega, \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1)) \\ \text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public} \qquad \text{DecodePtr}(\text{private const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \sigma_2(l_1) = (\omega_1,\ \text{private}\ bty,\ n,\ \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n)) \\ (i < 0) \vee (i \geq n) \qquad \text{T_WriteOOB}(\text{encrypt}(v),\ i,\ n,\ l_1,\ \text{private}\ bty,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc}) = (\sigma_3,\ \Delta_3, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2)\ \Downarrow_{wao1}^t\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Figure 4.18: Location-tracking SMC^2 semantic rules for reading and writing out-of-bounds for arrays.

Location-tracking Public Pointer Declaration

$$\frac{\begin{array}{c}(ty = \text{public } bty*) \vee ((ty = bty*) \wedge ((bty = \text{char}) \vee (bty = \text{void}))) \quad (\text{acc} = 0) \wedge (\text{bid} = none) \\ l = \phi() \quad \text{GetIndirection}(*) = i \quad \omega = \text{EncodeVal}(\text{public } bty*, [1, [(l_{default}, 0)], [1], i]) \\ \gamma_1 = \gamma[x \rightarrow (l, \text{public } bty*)] \quad \sigma_1 = \sigma[l \rightarrow (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))]\end{array}}{(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})}$$

Location-tracking Private Pointer Declaration

$$\frac{\begin{array}{c}((ty = bty*) \vee (ty = \text{private } bty*)) \wedge ((bty = \text{int}) \vee (bty = \text{float})) \quad (\text{acc} = 0) \wedge (\text{bid} = none) \\ l = \phi() \quad \text{GetIndirection}(*) = i \quad \omega = \text{EncodePtr}(\text{private } bty*, [1, [(l_{default}, 0)], [1], i]) \\ \gamma_1 = \gamma[x \rightarrow (l, \text{private } bty*)] \quad \sigma_1 = \sigma[l \rightarrow (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))]\end{array}}{(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp1} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})}$$

Location-tracking Private Pointer Declaration (Inside a Private - Conditioned If Else Branch)

$$\frac{\begin{array}{c}((ty = bty*) \vee (ty = \text{private } bty*)) \wedge ((bty = \text{int}) \vee (bty = \text{float})) \quad \text{GetIndirection}(*) = i \\ \gamma_1 = \gamma[x \rightarrow (l, \text{private } bty*)] \quad l = \phi() \quad \text{EncodePtr}(\text{private } bty*, [1, [(l_{default}, 0)], [1], i]) = \omega \\ \sigma_1 = \sigma[l \rightarrow (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))] \\ (\text{acc} > 0) \wedge ((\text{bid} = then) \vee (\text{bid} = else)) \quad \overline{\chi}_1 = l :: \overline{\chi}[\text{acc}]\end{array}}{(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp2} (\gamma_1, \sigma_1, \Delta, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})}$$

Location-tracking Public Pointer Read Single Location

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{public } bty*) \quad \sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]\end{array}}{(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l_1, \mu_1))}$$

Location-tracking Private Pointer Read Single Location

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1)) \\ \text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]\end{array}}{(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp2} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l_1, \mu_1))}$$

Location-tracking Private Pointer Read Multiple Locations

$$\frac{\begin{array}{c}\gamma(x) = (l, \text{private } bty*) \quad \sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ (bty = \text{int}) \vee (bty = \text{float}) \quad \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]\end{array}}{(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp1} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}, \bar{j}, i])}$$

Location-tracking Public Pointer Write

$$\frac{\begin{array}{c}\text{Label}(e, \gamma) = \text{public} \quad (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_e (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_e, \mu_e)) \\ \gamma(x) = (l, \text{public } bty*) \quad \sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i] \\ \text{T_UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], \text{public } bty, i], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty*) = (\sigma_2, \Delta_2, 1)\end{array}}{(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wp} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})}$$

Location-tracking Private Pointer Write

$$\frac{\begin{array}{c}\text{Label}(e, \gamma) = \text{public} \quad (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_e (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_e, \mu_e)) \\ \gamma(x) = (l, \text{private } bty*) \quad \sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i] \\ \text{T_UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*) = (\sigma_2, \Delta_2, 1)\end{array}}{(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wp1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})}$$

Location-tracking Private Pointer Write Multiple Locations

$$\frac{\begin{array}{c}(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_e (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}, \bar{j}, i]) \quad (bty = \text{int}) \vee (bty = \text{float}) \\ \gamma(x) = (l, \text{private } bty*) \quad \text{T_UpdatePtr}(\sigma_1, (l, 0), [\alpha, \bar{l}, \bar{j}, i], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*) = (\sigma_2, \Delta_2, 1)\end{array}}{(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})}$$

Figure 4.19: Location-tracking SMC^2 semantic rules for pointer declarations, reading, and writing.

Location-tracking Public Pointer Dereference Write Public Value

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \qquad \gamma(x) = (l,\ \text{public}\ bty*) \\ \text{Label}(e,\gamma) = \text{public} \qquad \sigma_1(l) = (\omega,\ \text{public}\ bty*,\ 1, \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1)) \\ (\text{acc} = 0) \wedge (\text{bid} = none) \qquad \text{DecodePtr}(\text{public}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ \text{public}\ bty,\ 1] \\ v \neq \text{skip} \qquad \text{T_UpdateOffset}(\sigma_1,\ (l_1, \mu_1),\ v, \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc}, \text{private}\ bty) = (\sigma_2, \Delta_2, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x =\ e) \Downarrow_{wdp}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Public Pointer Dereference Write Higher Level Indirection

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_e, \mu_e)) \qquad \text{Label}(e,\gamma) = \text{public} \\ \gamma(x) = (l,\ \text{public}\ bty*) \qquad \sigma_1(l) = (\omega,\ \text{public}\ bty*,\ 1, \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1)) \\ \text{DecodePtr}(\text{public}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i] \qquad i > 1 \qquad (\text{acc} = 0) \wedge (\text{bid} = none) \\ \text{T_UpdatePtr}(\sigma_1,\ (l_1, \mu_1),\ [1,\ [(l_e, \mu_e)],\ [1],\ i-1], \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ , \text{public}\ bty*) = (\sigma_2, \Delta_2, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow_{wdp1}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Pointer Dereference Write Private Value

$$\frac{\begin{array}{c}\text{Label}(e,\gamma) = \text{private} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \\ \gamma(x) = (l,\ \text{private}\ bty*) \qquad \sigma_1(l) = (\omega,\ \text{private}\ bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, \alpha)) \\ (bty = \text{int}) \vee (bty = \text{float}) \qquad v \neq \text{skip} \qquad \text{DecodePtr}(\text{private}\ bty*,\ \alpha,\ \omega) = [\alpha,\ \overline{l},\ \overline{j},\ 1] \\ \text{T_UpdatePriv}(\sigma_1,\ \alpha,\ \overline{l},\ \overline{j},\ \text{private}\ bty,\ v,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc}) = (\sigma_2,\ \Delta_2, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow_{wdp3}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Pointer Dereference Write Public Value

$$\frac{\begin{array}{c}\text{Label}(e,\gamma) = \text{public} \qquad (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \\ \gamma(x) = (l,\ \text{private}\ bty*) \qquad \sigma_1(l) = (\omega,\ \text{private}\ bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, \alpha)) \\ (bty = \text{int}) \vee (bty = \text{float}) \qquad v \neq \text{skip} \qquad \text{DecodePtr}(\text{private}\ bty*,\ \alpha,\ \omega) = [\alpha,\ \overline{l},\ \overline{j},\ 1] \\ \text{T_UpdatePriv}(\sigma_1,\ \alpha,\ \overline{l},\ \overline{j},\ \text{private}\ bty,\ \text{encrypt}(v),\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc}) = (\sigma_2,\ \Delta_2, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow_{wdp4}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Pointer Dereference Write Value Higher Level Indirection

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_e, \mu_e)) \qquad \text{Label}(e,\gamma) = \text{private} \\ \gamma(x) = (l,\ \text{private}\ bty*) \qquad \sigma_1(l) = (\omega,\ \text{private}\ bty*,\ \alpha, \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, \alpha)) \\ \text{DecodePtr}(\text{private}\ bty*,\ \alpha,\ \omega) = [\alpha,\ \overline{l},\ \overline{j},\ i] \qquad i > 1 \\ \text{T_UpdatePrivPtr}(\sigma_1,\ [\alpha, \overline{l},\ \overline{j}, i], [1, [(l_e, \mu_e)], [1], i-1], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private}\ bty*) = (\sigma_2, \Delta_2, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow_{wdp2}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Location-tracking Private Pointer Dereference Write Value Higher Level Indirection Multiple Location

$$\frac{\begin{array}{c}(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow_e^t (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [\alpha, \overline{l}_e, \overline{j}_e, i-1]) \\ \gamma(x) = (l,\ \text{private}\ bty*) \qquad \sigma_1(l) = (\omega,\ \text{private}\ bty*,\ \alpha, \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, \alpha)) \\ \text{DecodePtr}(\text{private}\ bty*,\ \alpha,\ \omega) = [\alpha,\ \overline{l},\ \overline{j},\ i] \qquad i > 1 \qquad \text{Label}(e,\gamma) = \text{private} \\ \text{T_UpdatePrivPtr}(\sigma_1,\ [\alpha, \overline{l},\ \overline{j}, i], [\alpha, \overline{l}_e, \overline{j}_e, i-1], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private}\ bty*) = (\sigma_2, \Delta_2, 1)\end{array}}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow_{wdp5}^t (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})}$$

Figure 4.20: Location-tracking SMC^2 semantic rules for pointer dereference write.

Location-tracking Public Pointer Dereference Single Location

$$\frac{\gamma(x) = (l,\ \text{public}\ bty*) \qquad \sigma(l) = (\omega,\ \text{public}\ bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1)) \qquad \text{DecodePtr}(\text{public}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1] \qquad \text{DerefPtr}(\sigma, \text{public}\ bty, (l_1, \mu_1)) = (v, 1)}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x)\ \Downarrow^{t}_{rdp}\ (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)}$$

Location-tracking Public Pointer Dereference Single Location Higher Level Indirection

$$\frac{\gamma(x) = (l,\ \text{public}\ bty*) \qquad \sigma(l) = (\omega_1,\ \text{public}\ bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1)) \qquad \text{DecodePtr}(\text{public}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i] \qquad \text{DerefPtrHLI}(\sigma, \text{public}\ bty*, (l_1, \mu_1)) = ([1,\ [(l_2, \mu_2)],\ [1],\ i-1], 1)}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x)\ \Downarrow^{t}_{rdp1}\ (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_2, \mu_2))}$$

Location-tracking Private Pointer Dereference Single Level Indirection

$$\frac{\gamma(x) = (l,\ \text{private}\ bty*) \qquad \sigma(l) = (\omega,\ \text{private}\ bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, \alpha)) \qquad \text{DecodePtr}(\text{private}\ bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1] \qquad \text{Retrieve_vals}(n, \bar{l}, \bar{j}, \text{private}\ bty, \sigma) = (v, 1) \qquad (bty = \text{int}) \vee (bty = \text{float})}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x)\ \Downarrow^{t}_{rdp2}\ (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)}$$

Location-tracking Private Pointer Dereference Higher Level Indirection

$$\frac{\gamma(x) = (l,\ \text{private}\ bty*) \qquad \sigma(l) = (\omega,\ \text{private}\ bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, \alpha)) \qquad (bty = \text{int}) \vee (bty = \text{float}) \qquad \text{DecodePtr}(\text{private}\ bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i] \qquad i > 1 \qquad \text{DerefPrivPtr}(\alpha,\ \bar{l},\ \bar{j},\ \text{private}\ bty*,\ \sigma) = ((\alpha',\ \bar{l}',\ \bar{j}'), 1)}{(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x)\ \Downarrow^{t}_{rdp3}\ (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [\alpha',\ \bar{l}',\ \bar{j}',\ i-1])}$$

Figure 4.21: Location-tracking SMC^2 semantic rules for pointer dereference read.

4.1.2 Algorithms

In this section, we will present and discuss the algorithms that were modified for use in the Location-tracking SMC^2 semantics. All other algorithms definitions remain the same, and can be found in Section 3.1.6. Algorithm 84 through Algorithm 89 are used in place of their corresponding algorithms from Basic SMC^2 in order to handle location-tracking within private-conditioned branches; they check if a location is already tracked, and update or add it appropriately before performing the update in memory. We use green text here to highlight which portions have been added or modified from their corresponding Basic SMC^2 algorithms. Algorithms 92 and 93 are the new private-conditioned **`if else`** helper algorithms – none of the previous Basic SMC^2 private-conditioned **`if else`** helper algorithms are used in this semantics.

84	T_UpdateVal	$::= f : (\sigma,\ l,\ v,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, a\ bty) \rightarrow (\sigma,\ \Delta)$
85	T_UpdatePtr	$::= f : (\sigma,\ (l, \mu),\ [\alpha,\ \bar{l},\ \bar{j},\ i],\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, ty) \rightarrow (\sigma,\ \Delta, j)$
86	T_UpdateOffset	$::= f : (\sigma, (l, \mu), v, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty) \rightarrow (\sigma, \Delta, j)$
87	T_UpdatePriv	$::= f : (\sigma, \alpha, \bar{l}, \bar{j}, ty, v, \Delta, \overline{\chi}, \text{bid}, \text{acc}) \rightarrow (\sigma, \Delta, j)$
88	T_UpdatePrivPtr	$::= f : (\sigma, [\alpha, \bar{l}, \bar{j}, i], [\alpha, \bar{l}, \bar{j}, i], \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty) \rightarrow (\sigma, \Delta, j)$
89	T_WriteOOB	$::= f : (v, i, n, l, ty, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}) \rightarrow (\sigma, \Delta, j)$
90	T_SetBytes	$::= f : ((l, \mu), a\ bty*, [\alpha,\ \bar{l},\ \bar{j},\ i], \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}) \rightarrow (\sigma_2, \Delta_2)$
91	T_UpdateLocationMap	$::= f : (ty, \alpha, \omega_1, \omega_f, \Delta, \overline{\chi}, \text{bid}, \text{acc}) \rightarrow \Delta_f$
92	T_restore	$::= f : (\sigma, \Delta, \text{acc}) \rightarrow \sigma$
93	T_resolve	$::= f : (\sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, res) \rightarrow (\sigma, \Delta)$

Algorithm 84 $(\sigma_2,\ \Delta_2) \leftarrow \text{T_UpdateVal}(\sigma,\ l,\ v,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, a\ bty)$

```
σ1[l → (ω1, a bty, n, PermL(Freeable, a bty, a, n))] = σ
if ((bid = then) ∧ (l ∉ χ̄[acc]) ∧ (a = private)) then
    if (l ∉ Δ[acc]) then
        v1 = DecodeVal(private bty, n, ω1)
        Δ2 = (l → (v1, v, v1)) :: Δ[acc]
    else
        Δ = (l → (v_orig, v_then, v_orig)) :: Δ1[acc]
        Δ2 = (l → (v_orig, v, v_orig)) :: Δ1[acc]
    end if
    Δ = Δ2
else if ((bid = else) ∧ (l ∉ χ̄[acc]) ∧ (a = private)) then
    if (l_m ∉ Δ[acc]) then
        v1 = DecodeVal(ty, n, ω1)
        Δ2 = (l → (v1, v1, v)) :: Δ[acc]
    else
        Δ = (l → (v_orig, v_then, v_else)) :: Δ1[acc]
        Δ2 = (l → (v_orig, v_then, v)) :: Δ1[acc]
    end if
    Δ = Δ2
else
    Δ2 = Δ
end if
ω2 = EncodeVal(a bty, v)
σ2 = σ1[l → (ω2, a bty, n, PermL(Freeable, a bty, a, n))]
return (σ2, Δ2)
```

Algorithm 84 (T_UpdateVal) is used to update regular (int or float) values in memory. This algorithm corresponds to Basic SMC^2 Algorithm 47, and like UpdateVal, it takes as input memory σ, the memory block identifier of the location we will be updating, the value to store into memory, and the type to store it as. This algorithm has the additional input arguments of location map Δ, local variable tracker $\overline{\chi}$, branch identifier bid, and the accumulator acc. T_UpdateVal first removes the original mapping from memory and

proceeds to check if we are in a private-conditioned branch, and if so checks that this location is not that of a local variable (as we do not need to track modifications to local variables, whose use will not persist beyond the scope of this branch). If both of these conditions are true, we check whether or not this location is already being tracked. If it is not, we need to add a mapping for this location with the original value and the new value. If we are inside the **`then`** branch, we add the new value as the **`then`** value and the original value as both the **`original`** and **`else`** values, in case this location does not get modified within the **`else`** branch. If we are inside the **`else`** branch, we add the original value as both the **`original`** and **`then`** values (as it was not modified within the **`then`** branch), and the new value as the **`else`** value. If this location is already being tracked, we simply update the new value for the branch that we are currently in. Finally, it encodes the value as the specified type, then inserts the new mapping with the updated byte data back into memory. It then returns the updated memory and location map.

Algorithm 85 $(\sigma_2,\ \Delta_2, j) \leftarrow \text{T_UpdatePtr}(\sigma, (l, \mu), [\alpha_{new}, \bar{l}_{new}, \bar{j}_{new}, i], \Delta, \overline{\chi}, \text{bid}, \text{acc}, a\ bty*)$

$j = 0$
$\sigma_1[l \rightarrow (\omega_1,\ ty_1,\ \alpha_3,\ \text{PermL}(\text{Freeable}, ty_1, a_1, \alpha_3)] = \sigma$
if $(\mu = 0) \wedge (a\ bty* = ty_1)$ **then**
  if $((\text{bid} = then) \wedge (l_m \notin \overline{\chi}[\text{acc}]) \wedge (a = \text{private}))$ **then**
    if $(l \notin \Delta[\text{acc}])$ **then**
      $[\alpha_3,\ \bar{l}_1,\ \bar{j}_1,\ i_2] = \text{DecodePtr}(ty,\ n_3,\ \omega_1)$
      $\Delta_2 = (l \rightarrow ([\alpha_3, \bar{l}_1, \bar{j}_1, i], [\alpha_{new}, \bar{l}_{new}, \bar{j}_{new}, i], [\alpha_3, \bar{l}_1, \bar{j}_1, i])) :: \Delta[\text{acc}])$
    else
      $\Delta = (l \rightarrow ([\alpha_{orig}, \bar{l}_{orig}, \bar{j}_{orig}, i], [n_{then}, \bar{l}_{then}, \bar{j}_{then}, i], [\alpha_{orig}, \bar{l}_{orig}, \bar{j}_{orig}, i])) :: \Delta_1[\text{acc}]$
      $\Delta_2 = (l \rightarrow ([\alpha_{orig}, \bar{l}_{orig}, \bar{j}_{orig}, i], [\alpha_{new}, \bar{l}_{new}, \bar{j}_{new}, i], [\alpha_{orig}, \bar{l}_{orig}, \bar{j}_{orig}, i])) :: \Delta_1[\text{acc}]$
    end if
    $\Delta = \Delta_2$
  else if $((\text{bid} = else) \wedge (l_m \notin \overline{\chi}[\text{acc}]) \wedge (a = \text{private}))$ **then**
    if $(l_m \notin \Delta[\text{acc}])$ **then**
      $[\alpha_3, \bar{l}, \bar{j}, i] = \text{DecodePtr}(ty,\ \alpha_3,\ \omega_1)$
      $\Delta_2 = (l \rightarrow ([\alpha_3, \bar{l}_1, \bar{j}_1, i], [\alpha_3, \bar{l}_1, \bar{j}_1, i], [\alpha_{new}, \bar{l}_{new},\ \bar{j}_{new}, i])) :: \Delta[\text{acc}]$
    else
      $\Delta = (l \rightarrow ([\alpha_{orig}, \bar{l}_{orig}, \bar{j}_{orig}, i], [\alpha_{then}, \bar{l}_{then}, \bar{j}_{then}, i], [\alpha_{else}, \bar{l}_{else}, \bar{j}_{else}, i])) :: \Delta_1[\text{acc}]$
      $\Delta_2 = (l \rightarrow ([\alpha_{orig}, \bar{l}_{orig}, \bar{j}_{orig}, i], [\alpha_{then}, \bar{l}_{then}, \bar{j}_{then}, i], [\alpha_{new}, \bar{l}_{new}, \bar{j}_{new}, i])) :: \Delta_1[\text{acc}]$
    end if
    $\Delta = \Delta_2$
  else if $(\text{bid} = none)$ **then**
    $\Delta_2 = \Delta$
  else
    ERROR
  end if
  $\omega_2 = \text{EncodePtr}(a\ bty*, [\alpha_{new}, \bar{l}_{new}, \bar{j}_{new}, i])$
  $\sigma_2 = \sigma_1[l \rightarrow (\omega_2, ty_1, \alpha_{new}, \text{PermL}(\text{Freeable}, ty_1, a_1, \alpha_{new}))]$
else
  $(\sigma_2, \Delta_2) = \text{T_SetBytes}((l, \mu), a\ bty*, [\alpha,\ \bar{l},\ \bar{j},\ i], \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc})$
end if
return $(\sigma_2,\ \Delta_2, j)$

Algorithm 85 ($\mathrm{T_UpdatePtr}$) is used to update the pointer data structure for a pointer. This algorithm corresponds to Basic SMC^2 Algorithm 48, and like $\mathrm{UpdatePtr}$, it takes as input memory σ, the location (memory block identifier and offset) we will be updating, the value to store into memory, and the type to store the value as. This algorithm has the additional input arguments of location map Δ, local variable tracker $\overline{\chi}$, branch identifier bid, and the accumulator acc. First, we extract the given memory block identifier's mapping in memory. If the given offset is 0 and the given pointer type matches the type in that mapping, we encode the pointer data structure into its byte representation and add a new mapping to memory with the new byte data, and set the tag to 1, indicating that we performed a *well-aligned* update to memory. It proceeds to check if we are in a private-conditioned branch, checking and adding to or modifying the location map as we did for Algorithm 84, simply with pointer data structures as the values. Otherwise, if the update is not *well-aligned*, we call $\mathrm{T_SetBytes}$ to perform the update to memory at this location. Finally, it returns the updated memory, location map, and tag indicating alignment.

Algorithm 86 ($\mathrm{T_UpdateOffset}$) is designed to update a value at an offset within a memory block, and is used by semantic rule Location-tracking Public Pointer Dereference Write Public Value. This algorithm corresponds to Basic SMC^2 Algorithm 51, and like $\mathrm{UpdateOffset}$, it takes as input memory, a location, a value, and a type. This algorithm has the additional input arguments of location map Δ, local variable tracker $\overline{\chi}$, branch identifier bid, and the accumulator acc. First, we check that we are not trying to update the default location, which is not valid (i.e., this would cause a segmentation fault). If we are, we return with the tag -1, which will not allow the statement that triggered this to resolve any further. We extract the memory block we are looking going to be updating. Next, we check if the memory block is of the expected type, the offset is 0, and the number of locations is 1 to see if this is a simple update. We proceed to check if we are in a private-conditioned branch, checking and adding to or modifying the location map as we did for Algorithm 84. We then encode the value into its byte representation and add the updated mapping into the final memory and set the tag to be 1, as we have made a *well-aligned* update to memory. If we are not at offset 0, and the memory block happens to be a block of array data, we can check if our update will be aligned by checking if the type is the same, then if the byte-offset of the pointer aligns with a value of the given type within the block by using the modulo operation, and also that it is within the range of the current block based on the given type and the number of locations in the block. If all of these elements are true, we will have a *well-aligned* update to memory. We use Algorithm 90 ($\mathrm{T_SetBytes}$) to perform the update here, as that algorithm facilitates proper insertion of the byte representation for a value into a larger block or across

Algorithm 86 $(\sigma_f,\ \Delta_f,\ j) \leftarrow$ T_UpdateOffset$(\sigma,\ (l,\mu),\ v,\ \Delta, \overline{\chi}, \text{bid}, \text{acc},\ a\ bty)$

1: **if** $(l_{default} = l)$ **then**
2: ERROR
3: **return** $(\sigma, -1)$
4: **end if**
5: $j = 0$
6: $\sigma_f[l \rightarrow (\omega_1,\ ty_1,\ \alpha,\ \text{PermL}(\text{Freeable}, ty_1, a_1, \alpha))] = \sigma$
7: **if** $(a\ bty = ty_1) \wedge (\mu = 0) \wedge (\alpha = 1)$ **then**
8: **if** $((\text{bid} = then) \wedge (l \notin \overline{\chi}[\text{acc}]) \wedge (a = \text{private}))$ **then**
9: **if** $(l \notin \Delta[\text{acc}])$ **then**
10: $v_1 = \text{DecodeVal}(\text{private}\ bty,\ n,\ \omega_1)$
11: $\Delta_f = (l \rightarrow (v_1,\ v,\ v_1)) :: \Delta[\text{acc}]$
12: **else**
13: $\Delta = (l \rightarrow (v_{orig},\ v_{then},\ v_{orig})) :: \Delta_1[\text{acc}]$
14: $\Delta_f = (l \rightarrow (v_{orig},\ v,\ v_{orig})) :: \Delta_1[\text{acc}]$
15: **end if**
16: $\Delta = \Delta_f$
17: **else if** $((\text{bid} = else) \wedge (l \notin \overline{\chi}[\text{acc}]) \wedge (a = \text{private}))$ **then**
18: **if** $(l_m \notin \Delta[\text{acc}])$ **then**
19: $v_1 = \text{DecodeVal}(ty,\ n,\ \omega_1)$
20: $\Delta_f = (l \rightarrow (v_1,\ v_1,\ v)) :: \Delta[\text{acc}]$
21: **else**
22: $\Delta = (l \rightarrow (v_{orig},\ v_{then},\ v_{else})) :: \Delta_1[\text{acc}]$
23: $\Delta_f = (l \rightarrow (v_{orig},\ v_{then},\ v)) :: \Delta_1[\text{acc}]$
24: **end if**
25: $\Delta = \Delta_f$
26: **else**
27: $\Delta_f = \Delta$
28: **end if**
29: $\omega_2 = \text{EncodeVal}(a\ bty,\ v)$
30: $\sigma_f = \sigma_f[l \rightarrow (\omega_2,\ ty_1,\ 1,\ \text{PermL}(\text{Freeable}, \widehat{ty}_1, a, 1))]$
31: $j = 1$
32: **else**
33: **if** $(a\ bty = ty_1) \wedge (\mu \,\%\, \tau(ty_1) = 0) \wedge (\frac{\mu}{\tau(ty_1)} < \alpha)$ **then**
34: $j = 1$
35: **end if**
36: $(\sigma_f, \Delta_f) = \text{T_SetBytes}((l,\mu), a\ bty, v, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc})$
37: **end if**
38: **return** (σ_f, Δ_f, j)

blocks. Finally, we return the updated memory, location map, and tag.

Algorithm 87 (T_UpdatePriv) is used during private pointer dereference writes at the first level of indirection, as shown in Figure 3.22. It facilitates the proper handling of private pointer data, particularly when there are multiple locations. This algorithm corresponds to Basic SMC^2 Algorithm 50, but has the additional input arguments of location map Δ, local variable tracker $\overline{\chi}$, branch identifier bid, and the accumulator acc, which it passes along to the algorithms that will perform the updates and checks to ensure tracking is handled properly within private-conditioned branches. First, we check if the default location is in the location list. If it is, then one of the possible locations for the pointer is an uninitialized location, and this would cause a segmentation fault at runtime. We return tag -1 if this is the case, and the program will get stuck

Algorithm 87 $(\sigma_1,\ \Delta_1, j) \leftarrow$ T_UpdatePriv$(\sigma,\ \alpha,\ \bar{l},\ \bar{j},\ \text{private } bty,\ v, \Delta, \overline{\chi}, \text{bid}, \text{acc})$

1: $j = 1$
2: **if** $(l_{default}, 0) \in \bar{l}$ **then**
3: ERROR
4: **return** $(\sigma, -1)$
5: **else if** $(\alpha = 1)$ **then**
6: $[(l_1, \mu_1)] = \bar{l}$
7: $(\sigma_1, \Delta_1, j) =$ T_UpdateOffset$(\sigma, (l_1, \mu_1), v, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty)$
8: **else**
9: **for all** $(l_m, \mu_m) \in \bar{l}$ **do**
10: $(\omega_m,\ ty_m,\ n,\ \text{PermL}(\text{Freeable}, ty_m, a_m, n)) = \sigma(l_m)$
11: **if** $(\mu_m = 0) \wedge (ty_m = \text{private } bty) \wedge (n = 1)$ **then**
12: $v_m = \text{DecodeVal}(\text{private } bty,\ \omega_m)$
13: $v'_m = (\bar{j}[m] \wedge v) \vee (\neg\, \bar{j}[m] \wedge v_m)$
14: $(\sigma_1, \Delta_1) =$ T_UpdateVal$(\sigma,\ l_m,\ v'_m, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty)$
15: $(\sigma, \Delta) = (\sigma_1, \Delta_1)$
16: **else if** $(\mu \,\%\, \tau(\text{private } bty) = 0) \wedge (\frac{\mu}{\tau(\text{private } bty)} < n) \wedge (ty_m = \text{private } bty)$ **then**
17: $[v_0, ..., v_{n-1}] = \text{DecodeVal}(\text{private } bty,\ n,\ \omega_m)$
18: $v'_{\mu_m} = (\bar{j}[m] \wedge v) \vee (\neg\, \bar{j}[m] \wedge v_{\mu_m})$
19: $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{v'_{\mu_m}}{v_{\mu_m}}\right)$
20: $(\sigma_1, \Delta_1) =$ T_UpdateVal$(\sigma,\ l_m,\ [v'_0, ..., v'_{n-1}], \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty)$
21: $(\sigma, \Delta) = (\sigma_1, \Delta_1)$
22: **else**
23: $(\sigma_1, \Delta_1) =$ T_SetBytes$((l_m, \mu_m), \text{private } bty, v, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc})$
24: $j = 0$
25: $(\sigma, \Delta) = (\sigma_1, \Delta_1)$
26: **end if**
27: **end for**
28: **end if**
29: **return** (σ_1, Δ_1, j)

and be unable to evaluate further. When there is a single location, it calls Algorithm 86 (T_UpdateOffset) for that location to update the value stored there and returns the updated memory and alignment tag. When there are multiple locations, it iterates through all the locations and updates each at location. If the location is aligned, it will privately update the value based on the tag for the location, and then update the location with this value. If the location is not aligned, then the alignment tag is set to 0 and the bytes in that location are overwritten with the new value that we are storing in memory. If the update is aligned at all locations, then the tag will be returned as 1; otherwise, 0.

Algorithm 88 (T_UpdatePrivPtr) is used during private pointer dereference writes at a level of indirection greater than 1, as shown in Figure 3.23. It facilitates the proper handling of private pointer data, particularly when there are multiple locations. This algorithm corresponds to Basic SMC^2 Algorithm 49, but has the additional input arguments of location map Δ, local variable tracker $\overline{\chi}$, branch identifier bid, and the accumulator acc. These arguments are passed along to other algorithms to properly handle the checks and updates to location map Δ when the updates occur. First, we check if the default location is in the location

Algorithm 88

$(\sigma_f, \Delta_f, j) \leftarrow$ T_UpdatePrivPtr$(\sigma, [\alpha, \bar{l}, \bar{j}, i], [\alpha_e, \bar{l}_e, \bar{j}_e, i-1], \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*)$

```
1: j = 1
2: if (l_default, 0) ∈ l̄ then
3:     ERROR
4:     j = −1
5: else if (α = 1) then
6:     [(l, μ)] = l̄
7:     (σ_f, Δ_f, j) ← T_UpdatePtr(σ, (l, μ), [α_e, l̄_e, j̄_e, i − 1], Δ, χ̄, bid, acc, private bty*)
8: else
9:     for all (l_m, μ_m) ∈ l̄ do
10:        σ_1[l_m → (ω_m, ty_m, n_m, PermL(Freeable, ty_m, a_m, n_m))] = σ
11:        if (μ_m = 0) ∧ (ty_m = private bty*) then
12:            [α_m, l̄_m, j̄_m, i − 1] = DecodePtr(ty, 1, ω_m)
13:            [α'_m, l̄'_m, j̄'_m] = CondAssign([α_e, l̄_e, j̄_e], [α_m, l̄_m, j̄_m], j_m)
14:            (σ_f, Δ_f, j_1) ← T_UpdatePtr(σ, (l_m, μ_m), [α'_m, l̄'_m, j̄'_m, i − 1], Δ, χ̄, bid, acc, private bty*)
15:            j = j ∧ j_1
16:            (σ, Δ) = (σ_f, Δ_f)
17:        else
18:            (σ_f, Δ_f) = T_SetBytes((l_m, μ_m), a bty*, [1, [(l_new, μ_new)], [1], i], σ, Δ, χ̄, bid, acc)
19:            j = 0
20:        end if
21:    end for
22: end if
23: return (σ_f, Δ_f, j)
```

list. If it is, then one of the possible locations for the pointer is an uninitialized location, and this would cause a segmentation fault at runtime. We return tag -1 if this is the case, and the program will get stuck and be unable to evaluate further. When there is a single location, it calls Algorithm 85 (T_UpdatePtr) for that location to update the location stored there, and returns the updated memory and alignment tag. When there are multiple locations, it iterates through all the locations and updates each location. If the location is aligned, it will combine the two location lists and privately update the tag list using Algorithm 78, then update the location with the new location and tag lists. If the location is not aligned, then the alignment tag is set to 0 and the bytes in that location are overwritten with the new location list that we are storing in memory. If the update is aligned at all locations, then the tag will be returned as 1; otherwise, 0.

Algorithm 89 (T_WriteOOB) is designed to store a value of the given type from memory as though it was at index i of the array in memory block l. This algorithm corresponds to Basic SMC^2 Algorithm 58, and like T_WriteOOB, it takes as input the value to write in memory v, the out of bounds index i, the number of values in the array n, the memory block of the array data l, the type of elements in the array $a\ bty$, and memory σ. This algorithm has the additional input arguments of location map Δ, local variable tracker $\overline{\chi}$, branch identifier bid, and the accumulator acc. It then iterates through memory until it finds the position that

would be for index i, encodes value v as the expected type, and places its byte representation into memory starting at that position. Before updating the memory, it proceeds to check if we are in a private-conditioned branch, checking and adding to or modifying the location map as we did for Algorithm 84, ensuring that all updates to all locations end up getting tracked. It is important to note here that index i will be public, as we do not overshoot the bounds of an array when we have a private index. As the algorithm iterates through memory, if all locations we iterate over are of the same type as the expected type, and the location we are writing the value to is also the expected type, then it will return tag 1, indicating that our read was *well-aligned*. Otherwise, tag 0 will be returned. We currently only show the algorithm handling overshooting in the positive direction, however, it can trivially extended to grab the previous blocks and iterate backwards through memory to handle a negative index as well.

Algorithm 90 ($\mathrm{T_SetBytes}$) is designed to store a value into a location in memory that may not be *well-aligned*. This algorithm corresponds to Basic SMC^2 Algorithm 65. It takes as input the location, the type to encode the byte representation of the value as, the value, the memory, location map Δ, local variable tracker $\overline{\chi}$, branch identifier bid, and the accumulator acc. It returns the updated memory and location map. Before it updates any location, it uses Algorithm 91 to check if we are in a private-conditioned branch and modify the location map as we did for Algorithm 84. We separate this functionality into a separate algorithm for due to the length and complexity of $\mathrm{T_SetBytes}$ as well as the need for the same functionality to be repeated several times within $\mathrm{T_SetBytes}$.

It is worthwhile to note here that we never change the privacy label or type for the location, we simply encode the value into a byte representation based on it's expected type (not that of the location). This prevents any unintentional encryption of public values or decryption of private values. When we later read from this location, we will again read from it as the type we are expecting to be there rather than the type that is there - this may result in garbage values being used in a program that was not ensured to be correct by the programmer, but it prevents any information leakage about private data when stored in an incorrect position. This algorithm first encodes the value as the given type, then attempts to store it into the current location. If it will not fully fit in the current location, it loops until all bytes have been properly stored into memory. For an in depth description of the functionality of this algorithm, please see Basic SMC^2 Algorithm 65.

Algorithm 91 ($\mathrm{T_UpdateLocationMap}$) is designed to ensure proper tracking of values for each location when we are inside private-conditioned branches. This algorithm is only called from within Algorithm 90, and its functionality is specific to the needs of that algorithm. It first decodes the bytes into values, with v_f

being that value that is being stored into the location and v_1 being the value that was stored in the location originally. It then proceeds to check if we are in a private-conditioned branch, and if so checks that this location is not that of a local variable (as we do not need to track modifications to local variables, whose use will not persist beyond the scope of this branch). If both of these conditions are true, we check whether or not this location is already being tracked. If it is not, we need to add a mapping for this location with the original value and the new value. If we are inside the **`then`** branch, we add the new value as the **`then`** value and the original value as both the **`original`** and **`else`** values, in case this location does not get modified within the **`else`** branch. If we are inside the **`else`** branch, we add the original value as both the **`original`** and **`then`** values (as it was not modified within the **`then`** branch), and the new value as the **`else`** value. If this location is already tracked, we simply update the new value for the branch that we are currently in.

Algorithm 92 (T_restore) is used within the private-conditioned **`if else`** statement after the **`then`** branch evaluation is complete to restore the original values for all locations before we begin evaluating the **`else`** branch. This algorithm iterates through the entire location map for the current level of nesting, handling the values at each location based on whether they are singular values, lists of values, or pointer data structures. For each type, remove its mapping from memory, encode the original value into bytes based on the type, then store it back into memory.

Algorithm 93 (T_resolve) is used within the private-conditioned **`if else`** statement after the **`else`** branch evaluation is complete to privately resolve which values are the true values and place them into memory. It first looks up the result of the private conditional, then iterates through all mappings within the current level of nesting. The algorithm handles the resolution of the values for each location based on their type – this is identical to how we resolved the true values in Basic SMC^2, taking the **`then`** and **`else`** values and the result of the conditional and performing bitwise operations to privately find the true singular value, and using Algorithm 78 to find the new list of locations and tag for pointer data structures. We use T_UpdateVal and T_UpdatePtr to ensure that, if we are within a nested private-conditioned **`if else`** statement, the next outer level of nesting will be updated to reflect the changes that occurred within this level. It is important to note here that the branch identifier that is passed into this algorithm is that of which we entered into this private-conditioned **`if else`** statement with, and thus it reflects the branch (if any) that we are inside from the outer scope.

Algorithm 89 $(\sigma,\ \Delta,\ j) \leftarrow$ T_WriteOOB$(v,\ i,\ n,\ l,\ a\ bty,\ \sigma,\ \Delta, \overline{\chi}, \text{bid}, \text{acc})$

$\omega_v = \text{EncodeVal}(a\ bty, v)$
$n_b = (i - n) \cdot \tau(a\ bty)$
$j = 1$
while $(n_b > 0) \vee (|\omega_v| > 0)$ **do**
  $l = \text{GetBlock}(l)$
  $\sigma_1[l \rightarrow (\omega,\ ty_1,\ \alpha,\ \text{PermL}(\text{Freeable}, ty_1, a_1, \alpha))] = \sigma$
  if $(ty_1 \neq a\ bty)$ **then**
    $j = 0$
  end if
  if $(n_b < \tau(ty_1) \cdot \alpha)$ **then**
    if $(|\omega_v| > \tau(ty_1) \cdot \alpha - n_b)$ **then**
      $\omega_1 = \omega[0 : n_b] + \omega_v + \omega[|\omega_v| + n_b :]$
      $\omega_v = [\,]$
    else if $(|\omega_v| = \tau(ty_1) \cdot \alpha - n_b)$ **then**
      $\omega_1 = \omega[0 : n_b] + \omega_v$
      $\omega_v = [\,]$
    else
      $\omega_1 = \omega[0 : n_b] + \omega_v[0 : \tau(ty_1) \cdot \alpha - n_b - 1]$
      $\omega_v = \omega_v[\tau(ty_1) \cdot \alpha - n_b :]$
    end if
    if $((\text{bid} = then) \wedge (l \notin \overline{\chi}[\text{acc}]) \wedge (a = \text{private}))$ **then**
      $v_f = \text{DecodeVal}(\text{private}\ bty,\ \alpha,\ \omega_1)$
      if $(l \notin \Delta[\text{acc}])$ **then**
        $v_1 = \text{DecodeVal}(\text{private}\ bty,\ \alpha, \omega)$
        $\Delta = (l \rightarrow (v_1,\ v_f,\ v_1)) :: \Delta[\text{acc}]$
      else
        $(l \rightarrow (v_{orig},\ v_{then},\ v_{orig})) :: \Delta_1[\text{acc}] = \Delta$
        $\Delta = (l \rightarrow (v_{orig},\ v_f,\ v_{orig})) :: \Delta_1[\text{acc}]$
      end if
    else if $((\text{bid} = else) \wedge (l \notin \overline{\chi}[\text{acc}]) \wedge (a = \text{private}))$ **then**
      $v_f = \text{DecodeVal}(ty,\ \alpha,\ \omega_1)$
      if $(l_m \notin \Delta[\text{acc}])$ **then**
        $v_1 = \text{DecodeVal}(\text{private}\ bty,\ \alpha, \omega)$
        $\Delta = (l \rightarrow (v_1,\ v_1,\ v_f)) :: \Delta[\text{acc}]$
      else
        $(l \rightarrow (v_{orig},\ v_{then},\ v_{else})) :: \Delta_1[\text{acc}] = \Delta$
        $\Delta = (l \rightarrow (v_{orig},\ v_{then},\ v_f)) :: \Delta_1[\text{acc}]$
      end if
    end if
    $\sigma = \sigma_1[l \rightarrow (\omega_1,\ ty_1,\ \alpha,\ \text{PermL}(\text{Freeable}, ty_1, a_1, \alpha))]$
  end if
  $n_b = max(0, n_b - \tau(ty_1) \cdot \alpha)$
end while
return $(\sigma,\ \Delta,\ j)$

Algorithm 90 $(\sigma_f, \Delta_f) \leftarrow \text{T_SetBytes}((l, \mu), ty, v, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc})$

$\sigma_f[l \rightarrow (\omega,\ ty_1,\ \alpha,\ \text{PermL}(\text{Freeable}, ty_1, a, \alpha))] = \sigma$
$\omega_v = \text{NULL}$
if $ty = a\ bty$ **then**
    $\omega_v = \text{EncodeVal}(ty, v)$
else
    $\omega_v = \text{EncodePtr}(ty, v)$
end if
$n_l = \tau(ty_1) \cdot \alpha - \mu$
if $(\tau(ty) < n_l - 1)$ **then**
    $\omega_f = \omega[0 : \mu - 1] + \omega_v + \omega[\mu + \tau(ty) :]$
    $\Delta_f = \text{T_UpdateLocationMap}(ty, \alpha, \omega, \omega_f, \Delta, \overline{\chi}, \text{bid}, \text{acc})$
    $\sigma_f = \sigma_f[l \rightarrow (\omega_f,\ ty_1,\ \alpha,\ \text{PermL}(\text{Freeable}, ty_1, a, \alpha))]$
else if $(\tau(ty) = n_l - 1)$ **then**
    $\omega_f = \omega[0 : \mu - 1] + \omega_v$
    $\Delta_f = \text{T_UpdateLocationMap}(ty, \alpha, \omega, \omega_f, \Delta, \overline{\chi}, \text{bid}, \text{acc})$
    $\sigma_f = \sigma_f[l \rightarrow (\omega_f,\ ty_1,\ \alpha,\ \text{PermL}(\text{Freeable}, ty_1, a, \alpha))]$
else
    $\omega_f = \omega[0 : \mu - 1] + \omega_v[0 : n_l - 1]$
    $\Delta_f = \text{T_UpdateLocationMap}(ty, \alpha, \omega, \omega_f, \Delta, \overline{\chi}, \text{bid}, \text{acc})$
    $\sigma_f = \sigma_f[l \rightarrow (\omega_f,\ ty_1,\ \alpha,\ \text{PermL}(\text{Freeable}, ty_1, a, \alpha))]$
    $\omega_v = \omega_v[n_l :]$
    $n_v = \tau(ty) - n_l$
    while $(n_v > 0)$ **do**
        $l = \text{GetBlock}(l)$
        $\sigma_f[l \rightarrow (\omega_c,\ ty_c,\ \alpha_c,\ \text{PermL}(\text{Freeable}, ty_c, a, \alpha_c))] = \sigma_f$
        $n_c = \tau(ty_c) \cdot \alpha_c$
        if $(n_v < n_c)$ **then**
            $\omega_f = \omega_v + \omega_c[0 : n_v]$
        else if $(n_v = n_c)$ **then**
            $\omega_f = \omega_v$
        else
            $\omega_f = \omega_v[0 : n_c - 1]$
            $\omega_v = \omega_v[n_c :]$
        end if
        $n_v = n_v - n_c$
        $\Delta_f = \text{T_UpdateLocationMap}(ty, \alpha_c, \omega, \omega_f, \Delta_f, \overline{\chi}, \text{bid}, \text{acc})$
        $\sigma_f = \sigma_f[l \rightarrow (\omega_f,\ ty_c,\ \alpha_c,\ \text{PermL}(\text{Freeable}, ty_c, a, \alpha_c))]$
    end while
end if
return (σ_f, Δ_f)

Algorithm 91 $\Delta_f \leftarrow$ T_UpdateLocationMap$(ty, \alpha, \omega_1, \omega_f, \Delta, \overline{\chi}, \text{bid}, \text{acc})$

```
1: if (ty = a bty) then
2:     v_f = DecodeVal(ty, α, ω_f)
3:     v_1 = DecodeVal(ty, α, ω_1)
4: else
5:     v_f = DecodePtr(ty, α, ω_f)
6:     v_1 = DecodePtr(ty, α, ω_1)
7: end if
8: if ((bid = then) ∧ (l ∉ χ̄[acc]) ∧ (a = private)) then
9:     if (l ∉ Δ[acc]) then
10:        Δ_f = (l → (v_1, v_f, v_1)) :: Δ[acc]
11:    else
12:        (l → (v_orig, v_then, v_orig)) :: Δ_1[acc] = Δ
13:        Δ_f = (l → (v_orig, v_f, v_orig)) :: Δ_1[acc]
14:    end if
15: else if ((bid = else) ∧ (l ∉ χ̄[acc]) ∧ (a = private)) then
16:    if (l_m ∉ Δ[acc]) then
17:        Δ_f = (l → (v_1, v_1, v_f)) :: Δ[acc]
18:    else
19:        (l → (v_orig, v_then, v_else)) :: Δ_1[acc] = Δ
20:        Δ_f = (l → (v_orig, v_then, v_f)) :: Δ_1[acc]
21:    end if
22: end if
23: return Δ_f
```

Algorithm 92 $\sigma_2 \leftarrow$ T_restore$(\sigma,\ \Delta,\ \text{acc})$

```
for all (l → β) ∈ Δ[acc] do
    if β = (v_orig, v_then, v_else) then
        σ_1[l → (ω, private bty, 1, PermL(Freeable, private bty, private, 1))] = σ
        ω_1 = EncodeVal(private bty, v_orig)
        σ_2 = σ_1[l → (ω_1, private bty, 1, PermL(Freeable, private bty, private, 1))]
    else if β = ([v_orig_0, ..., v_orig_n], [v_then_0, ..., v_then_n], [v_else_0, ..., v_else_n]) then
        σ_1[l → (ω, private bty, n, PermL(Freeable, private bty, private, n))] = σ
        ω_1 = EncodeVal(private bty, [v_orig_0, ..., v_orig_n])
        σ_2 = σ_1[l → (ω_1, private bty, n, PermL(Freeable, private bty, private, n))]
    else if β = ([α_orig, l̄_orig, j̄_orig, i], [α_then, l̄_then, j̄_then, i], [α_else, l̄_else, j̄_else, i]) then
        σ_1[l → (ω, private bty*, 1, PermL(Freeable, private bty*, private, 1))] = σ
        ω_1 = EncodePtr(ty, [α_orig, l̄_orig, j̄_orig, i])
        σ_2 = σ_1[l → (ω_1, private bty*, α_orig, PermL(Freeable, private bty, private, α_orig))]
    end if
    σ = σ_2
end for
return σ_2
```

Algorithm 93 $(\sigma_1,\ \Delta_1) \leftarrow \text{T_resolve}(\sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ res_{\text{acc}})$

$(l_{res}, \text{private int}) = \gamma(res_{\text{acc}})$
$(\omega_{res}, \text{private int}, 1, \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1)) = \sigma(l_{res})$
$n_{res} = \text{DecodeVal}(\text{private } bty, 1, \omega_{res})$
for all $(l \rightarrow \beta) \in \Delta[\text{acc}]$ **do**
  if $\beta = (v_{orig}, v_{then}, v_{else})$ **then**
    $v_{final} = (n_{res} \wedge v_{then}) \vee (\neg\, n_{res} \wedge v_{else})$
    $(\sigma_1, \Delta_1) = \text{T_UpdateVal}(\sigma,\ l,\ v_{final}, \Delta, \overline{\chi}, \text{bid}, \text{acc} - 1)$
  else if $\beta = ([v_{orig_0}, ..., v_{orig_n}], [v_{then_0}, ..., v_{then_n}], [v_{else_0}, ..., v_{else_n}])$ **then**
    for all $m \in [0, ..., n]$ **do**
      $v_{final_m} = (n_{res} \wedge v_{then_m}) \vee (\neg\, n_{res} \wedge v_{else_m})$
    end for
    $(\sigma_1, \Delta_1) = \text{T_UpdateVal}(\sigma, l_x, [v_{f_0}, ..., v_{f_{n-1}}], \Delta, \overline{\chi}, \text{bid}, \text{acc} - 1)$
  else if $\beta = ([\alpha_{orig}, \bar{l}_{orig}, \bar{\bar{j}}_{orig}, i], [\alpha_{then}, \bar{l}_{then}, \bar{\bar{j}}_{then}, i], [\alpha_{else}, \bar{l}_{else}, \bar{\bar{j}}_{else}, i])$ **then**
    $[n_{final}, \bar{l}_{final}, \bar{\bar{j}}_{final}] = \text{CondAssign}([\alpha_{then}, \bar{l}_{then}, \bar{\bar{j}}_{then}], [\alpha_{else}, \bar{l}_{else}, \bar{\bar{j}}_{else}], n_{res})$
    $(\sigma_1,\ \Delta_1) = \text{T_UpdatePtr}(\sigma, l, [n_{final}, \bar{l}_{final}, \bar{\bar{j}}_{final}, ty, i], \Delta, \overline{\chi}, \text{bid}, \text{acc} - 1)$
  end if
  $\sigma = \sigma_1$
  $\Delta = \Delta_1$
end for
return (σ_1, Δ_1)

4.2 Correctness

The most challenging result is correctness, which we discuss first. Once correctness is proven, noninterference follows from a standard argument, with some adaptations needed to deal with the fact that private data is encrypted and that we want to show indistinguishability of evaluation traces.

We show the correctness of the Location-tracking SMC^2 semantics with respect to the Vanilla C semantics. As usual we will do this by establishing a simulation relation between a Location-tracking SMC^2 program and a corresponding Vanilla C program. To do so we face two main challenges. First, we need to guarantee that the private operations in a Location-tracking SMC^2 program are reflected in the corresponding Vanilla C program and that the evaluation steps between the two programs correspond. To address the former issue, we define an *erasure function* $\mathbf{Erase}$ which translates a Location-tracking SMC^2 program into a Vanilla C program by erasing all labels and replacing all functions specific to Location-tracking SMC^2 with their public equivalents. This function also translates memory. As an example, let us consider `pmalloc`; in this case, we have $\mathbf{Erase}(\mathrm{pmalloc}(e,\ ty) = (\mathrm{malloc}(\mathrm{sizeof}(\mathbf{Erase}(ty)) \cdot \mathbf{Erase}(e))))$. That is, `pmalloc` is rewritten to use `malloc`, and since the given private type is now public we can use the sizeof function to find the size we will need to allocate. To address the latter issue, we have defined our operational semantics in terms of big-step evaluation judgments which allow the evaluation trees of the two programs to have a corresponding structure. In particular, notice how we designed the Private If Else rule to perform multiple operations at one step, guaranteeing that we have similar "synchronization points" in the two evaluation trees.

Second, we need to guarantee that at each evaluation step the memory used by a Location-tracking SMC^2 program corresponds to the one used by the Vanilla C program. In our setting, with explicit memory management, manipulations of pointers, and array overshooting, the latter becomes particularly challenging. To better understand the issue here, let us consider the the rule Private Free. Remember that our semantic model associates a pointer with a list of locations, and the Private Free rule frees the first location in the list, and relocates the content of that location if it is not the true location. Essentially, this rule may swap the content of two locations if the first location in the list is not the location intended to be freed and make the Location-tracking SMC^2 memory and the Vanilla C memory look quite different. To address this challenge in the proof of correctness, we use a *map*, denoted ψ, to track the swaps that happen when the rule Private Free is used. The simulation uses and modifies this map to guarantee that the two memories correspond.

Another related challenge comes from array overshooting. If, by overshooting an array, a program goes

over or into memory blocks of different types, we may end up in a situation where the locations in the Location-tracking SMC^2 memory are significantly different from the ones in the Vanilla C memory. This is mostly due to the size of private types being larger than their public counterpart. One option to address this problem would be to keep a more complex map between the two memories. However, this can result in a much more complex proof, for capturing a behavior that is faulty, in principle. Instead, we prefer to focus on situations where overshooting arrays are *well-aligned*, in the sense that they access only memory locations and blocks of the right type and size. An illustration of this is given in Figure 3.4.

Before stating our correctness, we need to introduce some notation. We use codes $[d_1, \ldots, d_n]$, $[\widehat{d}_1, \ldots, \widehat{d}_m]$ in evaluations (i.e., $\Downarrow_{[d_1,\ldots,d_n]}$) to describe the rules of the semantics that are applied in order to derive the result. We write $[d_1, \ldots, d_n] \cong [\widehat{d}_1, \ldots, \widehat{d}_m]$ to state that the Location-tracking SMC^2 codes $[d_1, \ldots, d_n]$ are in correspondence with the Vanilla C codes $[\widehat{d}_1, \ldots, \widehat{d}_m]$. Almost every Location-tracking SMC^2 rule is in one-to-one correspondence with a single Vanilla C rule within an execution trace (exceptions being private-conditioned branches and `pmalloc`).

We write $s \cong \widehat{s}$ to state that the Vanilla C configuration statement $\widehat{s}$ can be obtained by applying the erasure function to the Location-tracking SMC^2 statement s. Similarly, we can extend this notation to configuration by also using the map ψ. That is, we write $(\gamma, \sigma, \mathrm{acc}, s) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$ to state that the Vanilla C configuration $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$ can be obtained by applying the erasure function to the Location-tracking SMC^2 configuration $(\gamma, \sigma, \mathrm{acc}, s)$, and memory $\widehat{\sigma}$ can be obtained from σ by using the map ψ.

We state correctness in terms of evaluation trees, since we will use evaluation trees to prove a strong form of noninterference in the next subsection. We use capital greek letters Π, Σ to denote evaluation trees. In the Location-tracking SMC^2 semantics, we write $\Pi \triangleright (\gamma, \sigma, \mathrm{acc}, s) \Downarrow_{[d_1,\ldots,d_n]} (\gamma_1, \sigma_1, \mathrm{acc}_1, v)$, to stress that the evaluation tree Π proves as conclusion that configuration $(\gamma, \sigma, \mathrm{acc}, s)$ evaluates to configuration $(\gamma_1, \sigma_1, \mathrm{acc}_1, v)$ by means of the codes $[d_1, ..., d_n]$. Similarly, for the Vanilla C semantics. We then write $\Pi \cong_\psi \Sigma$ for the extension to evaluation trees of the congruence relation with map ψ.

We can now state our correctness result showing that if an Location-tracking SMC^2 program s can be evaluated to a value v, and the evaluation is well-aligned (it is an evaluation where all the overshooting of arrays are well-aligned), then the Vanilla C program $\hat{s}$ obtained by applying the erasure function to s, i.e., $s \cong \hat{s}$ can be evaluated to $\hat{v}$ where $v \cong \hat{v}$. This property can be formalized in terms of congruence:

Theorem 4.2.1 (Location-tracking Correctness)**.** *Given configurations* $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s)$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$, *and map* ψ, *such that* $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s) \cong^t_\psi (\widehat{\gamma}, \widehat{\sigma}, \square\, \widehat{s})$, *if* $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s) \Downarrow^t_{[d_1,\ldots,d_n]} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1,$

bid, acc, v_1) *for codes* $[d_1, ..., d_n] \in (SmcC \cup SmcCT) \setminus SmcCX$*, then there exists a derivation* $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square\, \widehat{s})$ $\Downarrow'_{[\widehat{d}_1,...,\widehat{d}_n]} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square\, \widehat{s}_1)$ *for codes* $[\widehat{d}_1, ..., \widehat{d}_m] \in VanC \setminus VanCX$ *and a map* ψ_1 *such that* $[d_1, ..., d_n] \cong^t [\widehat{d}_1, ..., \widehat{d}_m]$, $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, v_1) \cong^t_{\psi_1} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square\, \widehat{v}_1)$*, and* $\Pi \cong^t_{\psi_1} \Sigma$.

Proof. Proof Sketch: By induction over all Location-tracking SMC^2 semantic rules.

The bulk of the complexity of this proof lies with rules pertaining to Private If Else, handling of pointers, and freeing of memory. We first provide a brief overview of the intuition behind some assumptions we must make for the proof and reasoning behind the use of some of the elements of the rules; then we dive deeper into the details for the more complex cases. The majority of this proof follows very similarly to the proof of correctness for Basic SMC^2. The main difference is for rule Private If Else, which is now managed by location instead of by variable. We will discuss this portion last. The full proof is available in Section 4.2.2, with this theorem identical to Theorem 4.2.2.

First, we need to assume private indexing is within bounds. Otherwise, we will not be able to prove correctness, because when using private indexing we will not go out of the bounds of the array in Location-tracking SMC^2, whereas the Vanilla C equivalent would. We also need to assume that input files are congruent, otherwise we cannot reason over the data input functions.

Similarly, when reasoning about rules containing overshooting and offsets into memory blocks, we must assert that such operations are *well-aligned* by type (i.e., for overshooting, we can only assert correctness over memory blocks and elements of the same type, and for offsets, the offset must be aligned with the start of an element within the block, and the expected and actual types of memory the same). Going over or into memory blocks of different types could cause significantly different locations between Location-tracking SMC^2 and Vanilla C due to private types being larger in size. An illustration of this is shown in Figure 3.4.

The correctness of most semantic rules follows easily, with Private Free being a notable exception. We leverage the correctness of Algorithm 75 (PFree), to show that correctness follows due to the deterministic definitions of this algorithm and those used by this algorithm. In this case, we must also show that the locations that are swapped within this rule, which is done to hide the true location, are deterministic based on our memory model definition. We use ψ to map the swapped locations, enabling us to show that, if these swaps were reversed, we would once again have memories that are directly congruent. This concept of locations being ψ-congruent is particularly necessary when reasoning about pointers in other rule cases.

We make the assertion that $v \neq \text{skip}$ in some rules where skip is not allowed as a value - this is especially important for asserting that an expression cannot contain `pfree(`e`)` (as any expression containing `pfree(`e`)` would evaluate to skip) and thus that ψ could not have been modified over said evaluation.

Another common assertion we must make is that in a semantic rule, we do not accept a hard-coded location (l, μ), $(\widehat{l}, \widehat{\mu})$ as the starting statement s, $\widehat{s}$. Hard-coded locations could lead to evaluating locations that are not congruent to each other and therefore we would not be able to prove correctness over such statements. This makes it so we can easily

assert that our starting statements are congruent (i.e., $s \cong \widehat{s}$).

For all the rules using private pointers, we will rely upon the pointer data structure containing a set of locations and their associated tags, only one of which being the true location. With this proven to be the case, it is then clear that the true location indicated within the private pointer's data structure in Location-tracking SMC^2 will be ψ-congruent with the location given by the pointer data structure in Vanilla C. We define this correspondence between locations as location ψ-congruence - ensuring that memory block IDs are the same, and the position into the block is congruent (same if public, proportional if private

For rule Private Malloc, we must relate this rule to the sequence of Vanilla C rules for Malloc, Multiplication, and Size of Type. This is due to the definition of `pmalloc` as a helper that allows the user to write programs without knowing the size of private types. This case follows from the definition of translating the Location-tracking SMC^2 program to a Vanilla C program, $\mathbf{Erase}(\text{pmalloc}(e,\ ty) = (\text{malloc}(\text{sizeof}(\mathbf{Erase}(ty)) \cdot \mathbf{Erase}(e))))$.

For the Private If Else rule, we must reason that our end results in memory after executing both branches and resolving correctly match the end result of having only executed the intended branch. The cases for both of these rules will have two subcases - one for the conditional being true, and the other for false. For this rule, we will need to reason about the location map Δ, and that we will catch all modifications and properly add them to Δ if the location being modified is not already tracked. If a new mapping is added, we store the current value in v_{orig} (as this location has not yet been modified), and the new value into the appropriate position based on our current branch. To do this, we must reason about each of our update functions, and that they will catch and add all modifications appropriately. This behavior will be used to ensure the correctness during both restoration and resolution.

Then we can evaluate the **`then`** branch, which will result in the values that are correct for if the condition had been true - this holds by induction. For T_restore, we reason that we properly store the results of the **`then`** branch as well as the original values for each location, as we discussed above. Then we can evaluate the **`else`** branch, which will result in the values that are correct for if the condition had been false - this holds by induction. We will then reason about the correctness of T_resolve. It must be set up to correctly take the temporary variable for the condition for the branch, and the values stored from the **`then`** branch and the ending values for the **`else`** branch. For the resolution of pointers, we must also reason about Algorithm 78 (CondAssign), because the resolution of pointer data is more involved. By proving that this algorithm will correctly resolve the true locations for pointers, we will then have that the statements created by T_resolve will appropriately resolve all pointers.

□

$$\mathbf{Erase}(\gamma,\ \sigma,\ \Delta, \overline{\chi}, \mathbf{bid}, \mathbf{acc}, s) = (\mathbf{Erase}(\gamma,\ \sigma,\ [\,],\ [\,]),\ \square,\ \mathbf{Erase}(s))$$

(a) Erasure function over configurations.

Figure 4.22: The Erasure function from Location-tracking SMC^2 configurations to Vanilla C configurations.

The full erasure function is shown in subsection 3.2.1, Figure 3.31. The only difference is the update to the erasure function over configurations, replacing subfigure 3.31a with subfigure 4.22a. Figure 4.22 shows erasure over an entire Location-tracking SMC^2 configuration, calling Erase on the four-tuple of the environment, memory, and two empty maps needed as the base for the Vanilla C environment and memory; removing the location map, local variable map, branch indicator, and accumulator (i.e., replacing it with $\square$); and calling Erase on the statement.

For the Proof of Correctness over the Location-tracking SMC^2 semantics, several definitions, axioms, and lemmas remain unchanged from their Basic SMC^2 versions, as the elements used in defining and proving these did not change:

- Definitions: 3.2.1, 3.2.2, 3.2.3, 3.2.4, 3.2.5, 3.2.6, 3.2.7, 3.2.8, 3.2.9, 3.2.10, 3.2.11, 3.2.12, 3.2.13, 3.2.14, 3.2.15, 3.2.16, 3.2.17, 3.2.18, 3.2.19, 3.2.20, 3.2.25, 3.2.26, 3.2.22, 3.2.23
- Axioms: 3.2.1, 3.2.3, 3.2.4, 3.2.5
- Lemmas: 3.2.3, 3.2.5, 3.2.6, 3.2.7, 3.2.8, 3.2.9, 3.2.10, 3.2.11, 3.2.12, 3.2.22, 3.2.23, 3.2.24, 3.2.25, 3.2.26, 3.2.27, 3.2.28, 3.2.29, 3.2.30, 3.2.31, 3.2.32, 3.2.33, 3.2.34, 3.2.35, 3.2.36, 3.2.37, 3.2.38, 3.2.39, 3.2.40, 3.2.41, 3.2.42, 3.2.43, 3.2.44, 3.2.45, 3.2.46, 3.2.47, 3.2.48, 3.2.49, 3.2.50, 3.2.51, 3.2.57, 3.2.58, 3.2.59, 3.2.60, 3.2.13, 3.2.63, 3.2.14, 3.2.15, 3.2.16, 3.2.18, 3.2.19, 3.2.62, 3.2.73

4.2.1 Supporting Metatheory

Definition 4.2.1. A Location-tracking SMC^2 configuration and a Vanilla C configuration are *ψ-congruent*, in symbols $(\gamma,\ \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc},\ s) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s})$, if and only if $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $s \cong_\psi \widehat{s}$.

Definition 4.2.2. A Location-tracking SMC^2 evaluation code and a Vanilla C evaluation code are *congruent*, in symbols $d \cong \widehat{d}$, if and only if $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s) \Downarrow_c (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \mathrm{bid}_1, \mathrm{acc}, v)$ and $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s}) \Downarrow_d (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v})$ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$ and $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \mathrm{bid}_1, \mathrm{acc}, v) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v})$ by Definition 4.2.1.

$SmcCT$	$VanC$	$SmcCT$	$VanC$	$SmcCT$	$VanC$
$\Downarrow^t_{d2}$	$\Downarrow'_d$	$\Downarrow^t_{dp2}$	$\Downarrow'_{dp}$	$\Downarrow^t_{da2}$	$\Downarrow'_{da}$

Figure 4.23: Table of Location-tracking SMC^2-specific evaluation codes in $SmcCT$ and their congruent Vanilla C evaluation codes in $VanC$. These evaluation codes are in addition to the SMC^2 evaluation codes shown in Figure 3.30.

Definition 4.2.3. A SMC^2 derivation $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow^t_{[d_1,\ldots,d_n]} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, v)$ and a Vanilla C derivation $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s}) \Downarrow'_{[\widehat{d}_1,\ldots,\widehat{d}_m]} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v})$ are *ψ-congruent*, in symbols $\Pi \cong'_\psi \Sigma$, if and only if given initial map ψ and ψ' derived from evaluating Π, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$, $[d_1, ..., d_n] \cong [\widehat{d}_1, ..., \widehat{d}_m]$, $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, v) \cong'_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v})$.

Lemma 4.2.1. *Given an initial map ψ, environment γ, memory σ, accumulator acc, and expression e, if $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$ such that $v \neq \text{skip}$, then $\text{pfree}(e_1) \notin e$ and the ending map ψ_1 is equivalent to ψ.*

Proof. By definition of SMC^2 rule pfree, skip is returned from the evaluation of $\text{pfree}(e_1)$. Therefore, by case analysis of the rules, if $v \neq \text{skip}$, then $\text{pfree}(e_1) \notin e$. By Definition 3.2.11, ψ is only modified after the execution of function pfree; therefore we have that $\psi_1 = \psi$. □

Lemma 4.2.2. *Given configuration $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s)$, if $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow^t_{c_1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, v)$, then $(l, \mu) \notin s$.*

Proof. Proof by contradiction using all semantic rules. □

Lemma 4.2.3. *Given map ψ and configuration $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s)$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$ such that $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $s \cong \widehat{s}$, if $(l, \mu) \notin s$, then $s \cong_\psi \widehat{s}$ and $(\gamma, \sigma, \text{acc}, s) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$.*

Proof. By Definition 3.2.10, if $s \cong \widehat{s}$ then $\text{Erase}(s) = \widehat{s}$. Given $s \cong \widehat{s}$ and $(l, \mu) \notin s$, we have for all $v_i \in s$, $\widehat{v}_i \in \widehat{s}$, $v_i \cong \widehat{v}_i$, and therefore by Definition 3.2.17 we have $v_i \in s$, $\widehat{v}_i \in \widehat{s}$, $v_i \cong_\psi \widehat{v}_i$. Therefore, by Definition 3.2.18 we have $s \cong_\psi \widehat{s}$ and by Definition 4.2.1 we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$. □

Lemma 4.2.4. *Given map ψ, location map Δ_1, local variable tracker $\overline{\chi}$, branch identifier bid, accumulator acc, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, value $v, \widehat{v}$, and type $a\ bty$, $\widehat{bty}$, if $\text{T_UpdateVal}(\sigma_1, l, v, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, a\ bty) = (\sigma_2, \Delta_2)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l \cong_\psi \widehat{l}$, $v \cong_\psi \widehat{v}$, and $a\ bty \cong \widehat{bty}$, then $\text{UpdateVal}(\widehat{\sigma}_1, \widehat{l}, \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.*

Proof. By definition of T_UpdateVal, UpdateVal, and Erase. □

Lemma 4.2.5. *Given map ψ, location map Δ_1, local variable tracker $\overline{\chi}$, branch identifier bid, accumulator acc, environment γ, $\widehat{\gamma}$, memory σ_1, $\widehat{\sigma}_1$, location (l, μ), $(\widehat{l}, \widehat{\mu})$, value v, $\widehat{v}$, and type $a\ bty$, $\widehat{bty}$, if $\text{T_UpdateOffset}(\sigma_1, (l, \mu), v, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, a\ bty) = (\sigma_2, \Delta_2, j)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$, $v = \widehat{v}$, and $a\ bty \cong \widehat{bty}$, then $\text{UpdateOffset}(\widehat{\sigma}_1, (\widehat{l}, \widehat{\mu}), \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, \widehat{j})$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $j \cong j'$.*

Proof. By definition of Algorithm T_UpdateOffset, UpdateOffset, and Erase, as well as Definition 3.2.13 and 3.2.3. □

Lemma 4.2.6. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, location map Δ_1, local variable tracker $\overline{\chi}$, branch identifier* bid*, accumulator* acc*, location $(l, \mu), (\widehat{l}, \widehat{\mu})$, pointer data structure $[\alpha, \overline{l}, \overline{j}, i]$, $[1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, and type $a\ bty*, \widehat{bty}*$, if* T_UpdatePtr$(\sigma, (l, \mu), [\alpha, \overline{l}, \overline{j}, i], \Delta, \overline{\chi}, \text{bid}, \text{acc}, a\ bty*) = (\sigma_1, \Delta_1, j)$, $a\ bty* \cong \widehat{bty}*$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$, *and* $[\alpha, \overline{l}, \overline{j}, i] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, *then* UpdatePtr$(\widehat{\sigma}, (\widehat{l}, \widehat{\mu}), [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], i], \widehat{bty}*) = (\widehat{\sigma}_1, \widehat{j})$ *such that* $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ *and* $j \cong \widehat{j}$.

Proof. By definition of T_UpdatePtr, UpdatePtr, and Erase, as well as Definition 3.2.14, 3.2.13, and 3.2.3. □

Lemma 4.2.7. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, number of locations α, memory block identifier list $\overline{l}$, location $(\widehat{l}_1, \widehat{\mu}_1)$, tag list $\overline{j}, \widehat{\overline{j}}$, level of indirection i, type* private bty, $\widehat{bty}$*, location map Δ_1, local variable tracker $\overline{\chi}$, branch identifier* bid*, accumulator* acc*, and value $v, \widehat{v}$, if* T_UpdatePriv$(\sigma_1, \alpha, \overline{l}, \overline{j}, \text{private } bty, v, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_2, \Delta_2, j)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, DeclassifyPtr$([\alpha, \overline{l}, \overline{j}, i], \text{private } bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, private $bty \cong \widehat{bty}$, *and* $v \cong_\psi \widehat{v}$, *then* UpdateOffset$(\widehat{\sigma}_1, (l_1, \widehat{\mu}_1), \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$ *such that* $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ *and* $j \cong j'$.

Proof. By definition of T_UpdatePriv, UpdateOffset, and Erase. □

Lemma 4.2.8. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, location $(\widehat{l}_1, \widehat{\mu}_1)$, and pointer data structure $[\alpha, \overline{l}, \overline{j}, i]$, $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1]$, $[1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1]$, location map Δ_1, local variable tracker $\overline{\chi}$, branch identifier* bid*, accumulator* acc*, and type* private $bty*, \widehat{bty}*$, *if* UpdatePrivPtr$(\sigma_1, [\alpha, \overline{l}, \overline{j}, i], [\alpha_e, \overline{l}_e, \overline{j}_e, i-1], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*) = (\sigma_2, \Delta_2, j)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, DeclassifyPtr$([\alpha, \overline{l}, \overline{j}, i], \text{private } bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, private $bty* \cong \widehat{bty}*$ *and* $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1] \cong_\psi [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1]$, *then* UpdatePtr$(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1], \widehat{bty}*) = (\widehat{\sigma}_2, \widehat{j})$ *such that* $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ *and* $j = \widehat{j}$.

Proof. By definition of T_UpdatePrivPtr, UpdatePtr, and Erase, as well as Definition 3.2.14 and 3.2.13. □

Lemma 4.2.9. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, type $a\ bty, \widehat{bty}$, value $v, \widehat{v}$, array index $i, \widehat{i}$, size $n, \widehat{n}$, location map Δ_1, local variable tracker $\overline{\chi}$, branch identifier* bid*, and accumulator* acc*, if* WriteOOB$(v, i, n, l, a\ bty, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_2, \Delta_2, j)$, $v \cong_\psi \widehat{v}$, $i = \widehat{i}$, $n = \widehat{n}$, $l = \widehat{l}$, $a\ bty \cong \widehat{bty}$, *and* $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, *then* WriteOOB$(\widehat{v}, \widehat{i}, \widehat{n}, \widehat{l}, \widehat{bty}, \widehat{\sigma}_1) = (\widehat{\sigma}_2, \widehat{j})$ *such that* $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ *and* $j \cong \widehat{j}$.

Proof. Proof Idea:

By definition of T_WriteOOB, if the number returned with the updated memory is 1, then the out of bounds access was *well-aligned* by Definition 3.2.2. Therefore, when we iterate over the ψ*-congruent* Vanilla C memory, the resulting out of bounds access will also be *well-aligned*. We use the definition of T_WriteOOB, WriteOOB, and Erase to help prove this. □

Lemma 4.2.10. *Given evaluation trace* $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$, $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v})$, *if* $(l, \mu) \neq v$ *and* Erase$(e) = \widehat{e}$, *then* $e \cong_\psi \widehat{e}$ *for any possible map* ψ.

Proof. By Definition 3.2.10 and case analysis of SMC^2 semantic rules. □

Lemma 4.2.11. *Given an initial map ψ, environment γ, memory σ, location map Δ_1, local variable tracker $\overline{\chi}$, branch identifier* bid*, accumulator* acc*, and stmt s, if $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow_s^t (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, v)$ and $\text{pfree}(e) \notin s$, then the ending map ψ_1 is equivalent to ψ.*

Proof. By definition of ψ. □

Lemma 4.2.12. *Given ψ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$, if $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow_c (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}_1, \text{acc}, v)$ and $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s}) \Downarrow_d (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v})$ such that $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}_1, \text{acc}, v) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \widehat{v})$, then $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$.*

Proof. Proof Idea:

Proof by induction over congruent evaluations. Using the definition of function Erase, we show that with every rule that adds to γ or adds to or modifies σ maintains both $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$ and $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ by Definition 3.2.15. □

Lemma 4.2.13. *Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{private int } res_\text{acc} = n)$ where $\text{Label}(n,\ \gamma) = \text{private}$, if $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{private int } res_\text{acc}) \Downarrow_{ds}^t (\gamma_1,\ \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$ then $\gamma_1 = \gamma :: \gamma_A$ such that $\gamma_A = [res_\text{acc} \rightarrow (\text{private int},\ l)]$ and $\sigma_1 = \sigma :: \sigma_A$ such that $\sigma_A = [l \rightarrow (\text{EncodeVal}(\text{private int}, n),\ \text{private int},\ 1,\ \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$.*

Proof.

Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{private int } res_\text{acc} = n)$ where $\text{Label}(n,\ \gamma) = \text{private}$, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{private int } res_\text{acc} = n) \Downarrow_{ds}^t (\gamma_1,\ \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc},\ \text{skip})$ by rule Declaration Assignment if $(\gamma,\ \sigma,\ \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{private int } res_\text{acc}) \Downarrow_{d1}^t (\gamma_1,\ \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc},\ \text{skip})$ and $(\gamma_1,\ \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc},\ res_\text{acc} = n) \Downarrow_{w2}^t (\gamma_1,\ \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc},\ \text{skip})$.

Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{private int } res_\text{acc})$, by rule Private Declaration we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{private int } res_\text{acc}) \Downarrow_{d1}^t (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$ where $l = \phi()$, $\gamma_1 = \gamma[res_\text{acc} \rightarrow (l,\ \text{private int})]$, $\omega = \text{EncodeVal}(\text{private int}, \text{NULL})$, $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private int}, 1,\ \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$. By Axiom 3.2.2, we have $res_\text{acc} \notin \gamma$. By definition of ϕ, we have $l \notin \sigma$. Therefore, we have $\gamma_1 = \gamma[res_\text{acc} \rightarrow (\text{private int},\ l)]$ and $\sigma_1 = \sigma[l \rightarrow (\omega,\ \text{private int},\ 1,\ \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$.

Given $(\gamma_1,\ \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc},\ res_\text{acc} = n)$ where $\text{Label}(res_\text{acc}, \gamma_1) = \text{Label}(n,\ \gamma_1) = \text{private}$, by rule Write Private Variable we have $(\gamma_1,\ \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc},\ res_\text{acc} = n) \Downarrow_{w2}^t (\gamma_1,\ \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc},\ \text{skip})$ where $\gamma_1(res_\text{acc}) = (l,\ \text{private int})$ and $\text{T_UpdateVal}(\sigma_1,\ l,\ n,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{private } bty) = \sigma_2$. By definition

of T_UpdateVal, we have $\sigma_1 = \sigma_3[l \rightarrow (\omega,\ \text{private int},\ 1,\ \text{PermL}(\text{Freeable},\ \text{private int},\ \text{private},\ 1))]$ and $\sigma_2 = \sigma_3[l \rightarrow (\omega',\ \text{private int},\ 1,\ \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$. Given $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private int}, 1, \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$, we have $\sigma_3 = \sigma$ and therefore $\sigma_2 = \sigma[l \rightarrow (\omega',\ \text{private int},\ 1,\ \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$.

Given $\gamma_1 = \gamma[res_\text{acc} \rightarrow (\text{private int},\ l)]$, we can conclude that $\gamma_1 = \gamma :: \gamma_A$ where $\gamma_A = [res_\text{acc} \rightarrow (\text{private int},\ l)]$.

Given $\sigma_2 = \sigma[l \rightarrow (\omega',\ \text{private int},\ 1,\ \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$, we can conclude that $\sigma_2 = \sigma :: \sigma_A$ where $\sigma_A = [l \rightarrow (\omega',\ \text{private int},\ 1,\ \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$.

Therefore, we can conclude $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{private int}\ res = n) \Downarrow^t_{ds} (\gamma :: \gamma_A,\ \sigma :: \sigma_A,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$ by rule Declaration Assignment. □

Lemma 4.2.14. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, location map Δ, local variable tracker $\overline{\chi}$, branch identifier* bid*, accumulator* acc*, and statement* $\text{private int}\ res_{\text{acc}+1} = n$*, if* $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ *and* $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{private int}\ res_{\text{acc}+1} = n) \Downarrow^t_{d_1} (\gamma_1,\ \sigma_1,\ \Delta,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$ *then* $(\gamma_1,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$.

Proof. By definition of Erase. □

Lemma 4.2.15. *Given an initial map ψ, environment γ, memory σ, accumulator* acc*, if* $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}+1,\ s) \Downarrow^t_{c_1} (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc}+1,\ v)$ *then* $\text{pfree}(e) \notin s$ *and* $\psi_1 = \psi$.

Proof. Proof by contradiction over the semantics showing that when $\text{acc} > 0$, $\text{pfree}(e)$ cannot be executed and therefore $\text{pfree}(e) \notin s$ if $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}+1,\ s) \Downarrow^t_{c_1} (\gamma_1,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}+1,\ v)$. □

Lemma 4.2.16. *Given original memory σ, updated memory σ_1, temporary memory σ_A, statement s, updated environment γ_1, and accumulator* acc*, if for all locations l modified by s, the location map $\Delta[\text{acc}]$ maintains the original value for l from the starting memory σ, and the only differences between memory σ and σ_1 that can occur are stored in the memory blocks with identifiers l and* $\text{T_restore}(\sigma_1 :: \sigma_A,\ \Delta,\ \text{acc}) = \sigma_2$*, then $\sigma_2 = \sigma :: \sigma_A$ such that $\forall l \in \Delta[\text{acc}]$, the original value remains unchanged in $\Delta[\text{acc}]$, $\Delta[\text{acc}]$ is updated with the modified values for l from the execution of the* **`then`** *branch, and x is updated to its original value from σ.*

Proof. By definition of Algorithm T_restore. □

Lemma 4.2.17. *Given map ψ, private condition result variable name $res_{\text{acc}+1}$, accumulator* acc*, starting environment $\gamma, \widehat{\gamma}$, location map Δ, local variable tracker $\overline{\chi}$, branch identifier* bid*,* **`then`** *memory $\sigma_t, \widehat{\sigma}_t$,* **`else`** *memory $\sigma_e, \widehat{\sigma}_e$,*

and temporary variable environment σ_A*, if* T_resolve($\sigma_e :: \sigma_A$, Δ_1, $\overline{\chi}$, bid, acc $+ 1$, $res_{\text{acc}+1}$) $= (\sigma_f, \Delta_2)$, $res_{\text{acc}+1}$ $\neq_{\text{private}}$ encrypt(0), $DMap_1$ *stores all modifications made to any variable within the* **`then`** *branch, and* $(\gamma, \sigma_t) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_t)$*, then* $\sigma_f = \sigma_t :: \sigma_A$ *such that* $(\gamma, \sigma_f) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_t)$.

Proof. By definition of Algorithm T_resolve, we have $\sigma_f = \sigma_t :: \sigma_A$. By Definition 3.2.15 we have $(\gamma, \sigma_t :: \sigma_A) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_t)$. □

Lemma 4.2.18. *Given map* ψ*, private condition result variable name* $res_{\text{acc}+1}$*, accumulator* acc*, starting environment* $\gamma, \widehat{\gamma}$*, location map* Δ*, local variable tracker* $\overline{\chi}$*, branch identifier* bid*,* **`then`** *memory* $\sigma_t, \widehat{\sigma}_t$*,* **`else`** *memory* $\sigma_e, \widehat{\sigma}_e$*, and temporary variable environment* σ_A*, if* T_resolve($\sigma_e :: \sigma_A$, Δ_1, $\overline{\chi}$, bid, acc $+ 1$, $res_{\text{acc}+1}$) $= (\sigma_f, \Delta_2)$ $res_{\text{acc}+1}$ $=_{\text{private}}$ encrypt(0), $DMap_1$ *stores all modifications made to any variable within the* **`else`** *branch,* $(\gamma, \sigma_e) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_e)$ *then* $\sigma_f = \sigma_e :: \sigma_A$ *such that* $(\gamma, \sigma_f) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_e)$.

Proof. By definition of Algorithm T_resolve, we have $\sigma_f = \sigma_e :: \sigma_A$. By Definition 3.2.15 we have $(\gamma, \sigma_e :: \sigma_A) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_e)$. □

4.2.2 Proof of Correctness

Theorem 4.2.2 (Location-tracking Correctness)**.** *Given configurations* $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s)$*,* $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s})$*, and map* ψ*, such that* $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \cong^t_\psi (\widehat{\gamma}, \widehat{\sigma}, \square\, \widehat{s})$*, if* $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow^t_{[d_1,\ldots,d_n]} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, v_1)$ *for codes* $[d_1, \ldots, d_n] \in (SmcC \cup SmcCT) \setminus SmcCX$*, then there exists a derivation* $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square\, \widehat{s}) \Downarrow'_{[\widehat{d}_1,\ldots,\widehat{d}_n]} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square\, \widehat{s}_1)$ *for codes* $[\widehat{d}_1, \ldots, \widehat{d}_m] \in VanC \setminus VanCX$ *and a map* ψ_1 *such that* $[d_1, \ldots, d_n] \cong^t [\widehat{d}_1, \ldots, \widehat{d}_m]$*,* $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, v_1) \cong^t_{\psi_1} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square\, \widehat{v}_1)$*, and* $\Pi \cong^t_{\psi_1} \Sigma$.

Proof.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \Downarrow^t_{ltt1} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \Downarrow^t_{ltt1} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$ by Location-tracking SMC^2 rule Private Less Than True, we have Label$(e_1, \gamma) =$ Label$(e_2, \gamma) =$ private, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2)$, $n_1 <_{\text{private}} n_2$, and encrypt$(1) = n_3$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 < e_2 \cong_\psi \widehat{e}_1 < \widehat{e}_2$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \Downarrow^t_{ltt1} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$, by Lemma 4.2.2 we have $(l, \mu) \notin e_1 < e_2$. Therefore, by Lemma 3.2.3, we have $e_1 < e_2 \cong \widehat{e}_1 < \widehat{e}_2$. By Definition 3.2.10 we have $e_1 < e_2 =$ Erase$(e_1) <$ Erase$(e_2) = \widehat{e}_1 < \widehat{e}_2$, and therefore $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$, $(l, \mu) \notin e_1 < e_2$, and ψ such

that $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ and $c_1 \cong d_1$. Given $n_1 \neq \mathrm{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$. Therefore, by Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$.

Given $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_2)$, ψ, $(l, \mu) \notin e_1 < e_2$, and $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$ and ψ_2 such that $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$ and $c_2 \cong d_2$. Given $n_2 \neq \mathrm{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$ and therefore $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$. Therefore, by Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_2 \cong_\psi \widehat{n}_2$.

Given $n_1 <_{\mathrm{private}} n_2$, $n_1 \cong_\psi \widehat{n}_1$, and $n_2 \cong_\psi \widehat{n}_2$, by Lemma 3.2.5 we have $\widehat{n}_1 < \widehat{n}_2$.

Given $\mathrm{encrypt}(1) = n_3$ and $\mathrm{Erase}(\mathrm{encrypt}(1)) = 1$ by Definition 3.2.17 we have $n_3 \cong_\psi 1$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$, and $\widehat{n}_1 < \widehat{n}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{ltt} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ 1)$ by Vanilla C rule Less Than True.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_3 \cong_\psi 1$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ 1)$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1 < e_2) \Downarrow^t_{ltt1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{ltt}$ $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ 1)$, $\Pi \cong_\psi \Sigma$, and $ltt1 \cong ltt$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1 < e_2) \Downarrow^t_{ltf1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1 < e_2) \Downarrow^t_{ltt1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1 == e_2) \Downarrow^t_{eqt1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1 < e_2) \Downarrow^t_{ltt1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1 == e_2) \Downarrow^t_{eqf1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1 < e_2) \Downarrow^t_{ltt1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1! = e_2) \Downarrow^t_{nef1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1 < e_2) \Downarrow^t_{ltt1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 != e_2) \Downarrow^t_{net1} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$ by Location-tracking SMC^2 rule Public-Private Less Than True, we have $\text{Label}(e_1, \gamma) = \text{public}$, $\text{Label}(e_2, \gamma) = \text{private}$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2)$, $\text{encrypt}(n_1) <_{\text{private}} n_2$, and $\text{encrypt}(1) = n_3$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 < e_2 \cong_\psi \widehat{e}_1 < \widehat{e}_2$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$, by Lemma 4.2.2 we have $(l, \mu) \notin e_1 < e_2$. Therefore, by Lemma 3.2.3, we have $e_1 < e_2 \cong \widehat{e}_1 < \widehat{e}_2$. By Definition 3.2.10 we have $e_1 < e_2 = \text{Erase}(e_1) < \text{Erase}(e_2) = \widehat{e}_1 < \widehat{e}_2$, and therefore $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$, $(l, \mu) \notin e_1 < e_2$, and ψ such that $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ and $c_1 \cong d_1$. Given $n_1 \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$. Therefore, by Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$, and by definition of Erase, we have $n_1 = \widehat{n}_1$.

Given $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2)$, ψ, $(l, \mu) \notin e_1 < e_2$, and $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$ and ψ_2 such that $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$ and $c_2 \cong d_2$. Given $n_2 \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$, therefore $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$. Therefore, by Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_2 \cong_\psi \widehat{n}_2$.

Given $\text{encrypt}(n_1) <_{\text{private}} n_2$, $n_1 = \widehat{n}_1$, and $n_2 \cong_\psi \widehat{n}_2$, by Definition 3.2.10 we have $\text{Erase}(\text{encrypt}(n_1)) = \widehat{n}_1$, and therefore $\text{encrypt}(n_1) \cong_\psi \widehat{n}_1$ by Definition 3.2.17. Therefore by Lemma 3.2.5 we have $\widehat{n}_1 < \widehat{n}_2$.

Given $\text{encrypt}(1) = n_3$ and $\text{Erase}(\text{encrypt}(1)) = 1$ by Definition 3.2.17 we have $n_3 \cong_\psi 1$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$, and $\widehat{n}_1 < \widehat{n}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{ltt} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ 1)$ by Vanilla C rule Less Than True.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_3 \cong_\psi 1$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ 1)$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{ltt}$ $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ 1)$, $\Pi \cong_\psi \Sigma$ and $ltt2 \cong ltt$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltf2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 == e_2) \Downarrow^t_{eqt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 == e_2) \Downarrow^t_{eqf2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 != e_2) \Downarrow^t_{net2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 != e_2) \Downarrow^t_{nef2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt3} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltf3} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 == e_2) \Downarrow^t_{eqt3} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 == e_2) \Downarrow^t_{eqf3} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 != e_2) \Downarrow^t_{net3} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 != e_2) \Downarrow^t_{nef3} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \Downarrow^t_{ltt} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ 1)$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \Downarrow^t_{ltt} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ 1)$ by Location-tracking SMC^2 rule Public Less Than True, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2)$, and $n_1 <_{\text{public}} n_2$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 < \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 < e_2 \cong_\psi \widehat{e}_1 < \widehat{e}_2$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \Downarrow^t_{ltt} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$, by Lemma 4.2.2 we have $(l, \mu) \notin e_1 < e_2$. Therefore, by Lemma 3.2.3, we have $e_1 < e_2 \cong \widehat{e}_1 < \widehat{e}_2$. By Definition 3.2.10 we have $e_1 < e_2 = \text{Erase}(e_1) < \text{Erase}(e_2) = \widehat{e}_1 < \widehat{e}_2$, and therefore $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$, $(l, \mu) \notin e_1 < e_2$, and ψ such that $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ and $c_1 \cong d_1$. Given $n_1 \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \Delta_1, \overline{\chi}, \text{bid}, \square,\ \widehat{n}_1)$. Therefore, by Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$, and by definition of Erase, we have $n_1 = \widehat{n}_1$.

Given $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2)$, ψ, $(l, \mu) \notin e_1 < e_2$, and $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$ and ψ_2 such that $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$ and $c_2 \cong d_2$. Given $n_2 \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$, therefore $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$. Therefore, by Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_2 \cong_\psi \widehat{n}_2$. Given

$\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n_2, \gamma) = \text{public}$, and by definition of Erase, we have $n_2 = \widehat{n}_2$.

Given $n_1 <_{\text{public}} n_2$, $n_1 = \widehat{n}_1$, and $n_2 = \widehat{n}_2$, we have $\widehat{n}_1 < \widehat{n}_2$.

By Definition 3.2.17, we have $1 \cong_\psi 1$. Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $1 \cong_\psi 1$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ 1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}_1 < \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{n}_1)$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ \widehat{n}_2)$, and $\widehat{n}_1 < \widehat{n}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{ltt} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ 1)$ by Vanilla C rule Less Than True.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $1 \cong_\psi 1$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ 1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ 1)$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 < e_2) \Downarrow^t_{ltt} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ 1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}_1 < \widehat{e}_2) \Downarrow'_{ltt} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ 1)$, $\Pi \cong_\psi \Sigma$, and $ltt \cong ltt$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltf} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, 0)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 == e_2) \Downarrow^t_{eqt} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, 1)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 == e_2) \Downarrow^t_{eqf} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, 0)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 != e_2) \Downarrow^t_{net} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, 1)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 != e_2) \Downarrow^t_{nef} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, 0)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi},\ \text{bid}, \text{acc}, 1)$.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 + e_2) \Downarrow^t_{bp} (\gamma,\ \sigma_2\ \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1{+}e_2) \Downarrow^t_{bp} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3)$ by Location-tracking SMC^2 rule Public Addition, we have $\mathrm{Label}(e_1, \gamma) = \mathrm{Label}(e_2, \gamma) = \mathrm{public}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_1)$, $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_2)$, and $n_1 +_{\mathrm{public}} n_2 = n_3$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1 + e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 + e_2 \cong_\psi \widehat{e}_1 + \widehat{e}_2$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1 + e_2) \Downarrow^t_{bp} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3)$, by Lemma 4.2.2 we have $(l, \mu) \notin e_1 + e_2$. Therefore, by Lemma 3.2.3, we have $e_1 + e_2 \cong \widehat{e}_1 + \widehat{e}_2$. By Definition 3.2.10 we have $e_1 + e_2 = \mathrm{Erase}(e_1) + \mathrm{Erase}(e_2) = \widehat{e}_1 + \widehat{e}_2$, and therefore $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_1)$, ψ, $(l, \mu) \notin e_1 + e_2$, and $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ and $c_1 \cong d_1$. Given $n_1 \neq \mathrm{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$. Therefore, by Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$. By Definition 3.2.17 we have $n_1 \cong \widehat{n}_1$. Given $\mathrm{Label}(e_1, \gamma) = \mathrm{public}$, we have $\mathrm{Label}(n_1, \gamma) = \mathrm{public}$, and by definition of Erase, we have $n_1 = \widehat{n}_1$.

Given $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_2)$, ψ, $(l, \mu) \notin e_1 + e_2$, and $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$ and ψ_2 such that $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$ and $c_2 \cong d_2$. Given $n_2 \neq \mathrm{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$, therefore $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$. Therefore, by Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_2 \cong_\psi \widehat{n}_2$. By Definition 3.2.17 we have $n_2 \cong \widehat{n}_2$. Given $\mathrm{Label}(e_2, \gamma) = \mathrm{public}$, we have $\mathrm{Label}(n_2, \gamma) = \mathrm{public}$, and by definition of Erase, we have $n_2 = \widehat{n}_2$.

Given $n_1 +_{\mathrm{public}} n_2 = n_3$, $n_1 = \widehat{n}_1$, and $n_2 = \widehat{n}_2$, we have $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$ where $n_3 = \widehat{n}_3$. Therefore by Definition 3.2.17 we have $n_3 \cong_\psi \widehat{n}_3$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$, and $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{bp} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_3)$ by Vanilla C rule Addition.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_3 \cong_\psi \widehat{n}_3$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_3)$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1{+}e_2) \Downarrow^t_{bp} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{bp}$

$(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_3)$, $\Pi \cong_\psi \Sigma$, and $bp \cong bp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 - e_2) \Downarrow^t_{bs} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 \cdot e_2) \Downarrow^t_{bm} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 + e_2) \Downarrow^t_{bp1} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 + e_2) \Downarrow^t_{bp1} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$ by Location-tracking SMC^2 rule Private Addition, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2)$, and $n_1 +_{\text{private}} n_2 = n_3$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 + e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 + e_2 \cong_\psi \widehat{e}_1 + \widehat{e}_2$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 + e_2) \Downarrow^t_{bp1} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$, by Lemma 4.2.2 we have $(l, \mu) \notin e_1 + e_2$. Therefore, by Lemma 3.2.3, we have $e_1 + e_2 \cong \widehat{e}_1 + \widehat{e}_2$. By Definition 3.2.10 we have $e_1 + e_2 = \text{Erase}(e_1) + \text{Erase}(e_2) = \widehat{e}_1 + \widehat{e}_2$, and therefore $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, ψ, $(l, \mu) \notin e_1 + e_2$, and $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ and $c_1 \cong d_1$. Given $n_1 \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$. Therefore, by Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$. By Definition 3.2.17 we have $n_1 \cong \widehat{n}_1$.

Given $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2)$, ψ, $(l, \mu) \notin e_1 + e_2$, and $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$ and ψ_2 such that $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$ and $c_2 \cong d_2$. Given $n_2 \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$, therefore $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$. Therefore, by Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_2 \cong_\psi \widehat{n}_2$. By Definition 3.2.17 we have $n_2 \cong \widehat{n}_2$.

Given $n_1 +_{\text{private}} n_2 = n_3$, $n_1 \cong \widehat{n}_1$, and $n_2 \cong \widehat{n}_2$, by Definition 3.2.10 we have $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$ where $n_3 \cong_\psi \widehat{n}_3$ by Definition 3.2.17.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_2)$, and $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{bp} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_3)$ by Vanilla C rule Addition.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n_3 \cong_\psi \widehat{n}_3$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_3)$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1{+}e_2) \Downarrow^t_{bp1} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{bp}$ $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n}_3)$, $\Pi \cong_\psi \Sigma$, and $bp1 \cong bp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1{-}e_2) \Downarrow^t_{bs1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1{+}e_2) \Downarrow^t_{bp1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1{\cdot}e_2) \Downarrow^t_{bm1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1{+}e_2) \Downarrow^t_{bp1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1{+}e_2) \Downarrow^t_{bp2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1{+}e_2) \Downarrow^t_{bp2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$ by Location-tracking SMC^2 rule Public-Private Addition, we have $\text{Label}(e_1, \gamma) = \text{public}$, $\text{Label}(e_2, \gamma) = \text{private}$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_2)$, and $\text{encrypt}(n_1) +_{\text{private}} n_2 = n_3$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 + e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1 + \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 + e_2 \cong_\psi \widehat{e}_1 + \widehat{e}_2$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1 + e_2) \Downarrow^t_{bp2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n_3)$, by Lemma 4.2.2 we have $(l, \mu) \notin e_1 + e_2$. Therefore, by Lemma 3.2.3, we have $e_1 + e_2 \cong \widehat{e}_1 + \widehat{e}_2$. By Definition 3.2.10 we have $e_1 + e_2 = \text{Erase}(e_1) + \text{Erase}(e_2) = \widehat{e}_1 + \widehat{e}_2$, and therefore $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, ψ, $(l, \mu) \notin e_1 + e_2$, and $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$ $\cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ and $c_1 \cong d_1$. Given $n_1 \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi},$

$\mathrm{bid}, \mathrm{acc}, n_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$. Therefore, by Definition 4.2.1 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$. By Definition 3.2.17 we have $n_1 \cong \widehat{n}_1$. Given $\mathrm{Label}(e_1, \gamma) = \mathrm{public}$, we have $\mathrm{Label}(n_1, \gamma) = \mathrm{public}$, and by definition of Erase, we have $n_1 = \widehat{n}_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_2)$, ψ, $(l, \mu) \notin e_1 + e_2$, and $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$ and ψ_2 such that $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_2) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$ and $c_2 \cong d_2$. Given $n_2 \neq \mathrm{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$, therefore $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$. Therefore, by Definition 4.2.1 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $n_2 \cong_\psi \widehat{n}_2$. By Definition 3.2.17 we have $n_2 \cong \widehat{n}_2$.

Given $\mathrm{encrypt}(n_1) +_{\mathrm{private}} n_2 = n_3$, $n_1 = \widehat{n}_1$, and $n_2 \cong \widehat{n}_2$, by Definition 3.2.10 we have $\mathrm{Erase}(\mathrm{encrypt}(n_1)) = \widehat{n}_1$, and therefore $\mathrm{encrypt}(n_1) \cong \widehat{n}_1$. By Definition 3.2.10, we have $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$ where $n_3 \cong \widehat{n}_3$. By Definition 3.2.17 we have $n_3 \cong_\psi \widehat{n}_3$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2)$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$, $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_2)$, and $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{bp} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_3)$ by Vanilla C rule Addition.

Given $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $n_3 \cong_\psi \widehat{n}_3$, by Definition 4.2.1 we have $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_3)$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e_1{+}e_2) \Downarrow^t_{bp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1 + \widehat{e}_2) \Downarrow'_{bp}$ $(\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n}_3)$, $\Pi \cong_\psi \Sigma$, and $bp2 \cong bp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e_1{-}e_2) \Downarrow^t_{bs2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e_1{+}e_2) \Downarrow^t_{bp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e_1{+}e_2) \Downarrow^t_{bp3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e_1{+}e_2) \Downarrow^t_{bp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e_1{-}e_2) \Downarrow^t_{bs3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e_1{+}e_2) \Downarrow^t_{bp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e_1{\cdot}e_2) \Downarrow^t_{bm2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e_1{+}e_2) \Downarrow^t_{bp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1{\cdot}e_2) \Downarrow^t_{bm3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, e_1{+}e_2) \Downarrow^t_{bp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow^t_{iep} (\gamma, \sigma_6, \Delta_6, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow^t_{iep} (\gamma,\ \sigma_6, \Delta_6, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private If Else, we have $\text{Label}(e, \gamma) = \text{private}$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n)$, $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{private int } res_{\text{acc}+1} = n) \Downarrow^t_{c_2} (\gamma_1,\ \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc},\ \text{skip})$, $\Delta_2 = \Delta_1.push([\,])$, $\overline{\chi}_2 = \overline{\chi}_1.push([\,])$, $(\gamma_1,\ \sigma_2, \Delta_2, \overline{\chi}_2, then, \text{acc}+1, s_1) \Downarrow^t_{c_3} (\gamma_2,\ \sigma_3, \Delta_3, \overline{\chi}_3, then, \text{acc}+1,\ \text{skip})$, $\text{T_restore}(\sigma_3, \Delta_3, \text{acc}+1) = \sigma_4$, $(\gamma_1,\ \sigma_4, \Delta_3, \overline{\chi}_2, else, \text{acc}+1,\ s_2) \Downarrow^t_{c_4} (\gamma_3,\ \sigma_5, \Delta_4, \overline{\chi}_4, else, \text{acc}+1,\ \text{skip})$, $\text{T_resolve}(\sigma_5, \Delta_4, \overline{\chi}, \text{bid}, \text{acc}+1, res_{\text{acc}+1}) = (\sigma_6, \Delta_5)$, and $\Delta_6 = \Delta_5.pop()$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{if } (\widehat{e})\widehat{s}_1 \text{ else } \widehat{s}_2)$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{if } (e)\ s_1 \text{ else } s_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{if } (\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{if } (e)\ s_1 \text{ else } s_2 \cong_\psi \text{if } (\widehat{e})\widehat{s}_1 \text{ else } \widehat{s}_2$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow^t_{iep} (\gamma,\ \sigma_6, \Delta_6, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{if } (e)\ s_1 \text{ else } s_2$. Therefore, by Lemma 3.2.3, we have $\text{if } (e)\ s_1 \text{ else } s_2 \cong \text{if } (\widehat{e})\widehat{s}_1 \text{ else } \widehat{s}_2$. By Definition 3.2.10, we have $\text{Erase}(\text{if } (e)\ s_1 \text{ else } s_2) = \text{if } (\text{Erase}(e))\ \text{Erase}(s_1) \text{ else } \text{Erase}(s_2)$, $\text{Erase}(e) = \widehat{e}$, $\text{Erase}(s_1) = \widehat{s}_1$, and $\text{Erase}(s_2) = \widehat{s}_2$. Therefore, we have $e \cong \widehat{e}$, $s_1 \cong \widehat{s}_1$, and $s_2 \cong \widehat{s}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, ψ, and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e})$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n)$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square, \widehat{n})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square, \widehat{n})$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$. Therefore, by Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_\psi \widehat{n}$.

Given $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private int } res_{\text{acc}+1} =\ n) \Downarrow^t_{c_2} (\gamma_1,\ \sigma_2, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.13 we have $\gamma_1 = \gamma :: \gamma_A$ such that $\gamma_A = [res_\text{acc} \rightarrow (\text{private int}, l_{res})]$ and $\sigma_2 = \sigma_1 :: \sigma_A$ such that $\sigma_A = [l_{res} \rightarrow (\text{EncodeVal}(\text{private int}, n), \text{private int}, 1, \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$. By Lemma 4.2.14, we have $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $(\gamma :: \gamma_A, \sigma_1 :: \sigma_A) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$. Given $\text{pfree}(e) \notin \text{private int } res_{\text{acc}+1} = n$, by Definition 3.2.11 ψ is not updated over the evaluation $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{private int } res_{\text{acc}+1} =\ n) \Downarrow^t_{c_2} (\gamma_1,\ \sigma_2, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc},\ \text{skip})$.

Given $(\gamma_1,\ \sigma_2,\ \Delta_2, \overline{\chi}_2, \mathit{then}, \mathrm{acc}+1,\ s_1) \Downarrow^t_{c_3} (\gamma_2,\ \sigma_3,\ \Delta_3, \overline{\chi}_3, \mathit{then}, \mathrm{acc}+1,\ \mathrm{skip})$, $\gamma_2 = \gamma :: \gamma_A$, and $\sigma_2 = \sigma_1 :: \sigma_A$, $(\gamma :: \gamma_A, \sigma_1 :: \sigma_A) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $s_1 \cong \widehat{s}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{s}_1)$ such that $(\gamma_1,\ \sigma_2,\ \Delta_2, \overline{\chi}_2, \mathrm{bid}, \mathrm{acc}+1,\ s_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{s}_1)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{s}_1) \Downarrow'_{d_3} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \Box,\ \mathrm{skip})$ and ψ_2 such that $(\gamma_2,\ \sigma_3,\ \Delta_3, \overline{\chi}_3, \mathit{then}, \mathrm{acc}+1,\ \mathrm{skip}) \cong_{\psi_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \Box,\ \mathrm{skip})$. By Lemma 4.2.15, we have $\mathrm{pfree}(e) \notin s_1$ and $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma_2,\ \sigma_3) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$. By Lemma 3.2.68, we have $\gamma_2 = \gamma'_2 :: \gamma_A$ and $\sigma_3 = \sigma'_3 :: \sigma_A$. By Lemma 3.2.73 we have $(\gamma'_2, \sigma'_3) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_2)$.

Given T_restore$(\sigma_3,\ \Delta_3,\ \mathrm{acc}+1) = \sigma_4$ and $\sigma_3 = \sigma'_3 :: \sigma_A$, by Lemma 4.2.16 we have $\sigma_4 = \sigma :: \sigma_A$ such that $\forall l \in \Delta[\mathrm{acc}]$, $\Delta[\mathrm{acc}]$ is updated with the modified values for l from the execution of the **then** branch, and x is updated to its original value from σ. By Definition 3.2.15, we have $(\gamma_2, \sigma_4) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$. By Lemma 4.2.12, we have $(\gamma, \sigma_4) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$. By Definition 3.2.15, we have $(\gamma :: \gamma_A, \sigma_4) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and given $\gamma_2 = \gamma :: \gamma_A$ we have $(\gamma_2, \sigma_4) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$.

Given $(\gamma_1,\ \sigma_4, \Delta_3, \overline{\chi}_2, \mathit{else}, \mathrm{acc}+1,\ s_2) \Downarrow^t_{c_4} (\gamma_3,\ \sigma_5, \Delta_4, \overline{\chi}_4, \mathit{else}, \mathrm{acc}+1,\ \mathrm{skip})$, $s_2 \cong \widehat{s}_2$, and $(\gamma_1, \sigma_4) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, by Lemma 4.2.3 we have $(\gamma_1,\ \sigma_4, \Delta_3, \overline{\chi}_2, \mathit{else}, \mathrm{acc}+1,\ s_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{s}_2)$. By the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{s}_2) \Downarrow'_{d_4} (\widehat{\gamma}_2,\ \widehat{\sigma}_3,\ \Box,\ \mathrm{skip})$ and ψ_3 such that $(\gamma_3,\ \sigma_5, \Delta_4, \overline{\chi}_4, \mathit{else}, \mathrm{acc}+1,\ \mathrm{skip}) \cong_{\psi_3} (\widehat{\gamma}_2,\ \widehat{\sigma}_3,\ \Box,\ \mathrm{skip})$. By Lemma 4.2.15, we have $\mathrm{pfree}(e) \notin s_2$ and $\psi_3 = \psi$. By Definition 4.2.1 we have $(\gamma_3,\ \sigma_5) \cong_\psi (\widehat{\gamma}_2,\ \widehat{\sigma}_3)$. By Lemma 3.2.68, we have $\gamma_3 = \gamma'_3 :: \gamma_A$ and $\sigma_5 = \sigma'_5 :: \sigma_A$. By Lemma 3.2.73 we have $(\gamma'_3, \sigma'_5) \cong_\psi (\widehat{\gamma}_2, \widehat{\sigma}_3)$. By Lemma 4.2.12, we have $(\gamma, \sigma'_5) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Subcase $n \neq_{\mathrm{private}} \mathrm{encrypt}(0)$

Given $n \cong_\psi \widehat{n}$ and $n \neq_{\mathrm{private}} \mathrm{encrypt}(0)$, we have $\widehat{n} \neq 0$.

Given T_resolve$(\sigma_5,\ \Delta_4,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc}+1,\ \mathit{res}_{\mathrm{acc}+1}) = (\sigma_6, \Delta_5)$, by Lemma 4.2.17 we have $\sigma_6 = \sigma'_4 :: \sigma_A$ and $(\gamma, \sigma_6) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \mathrm{if}\ (\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{n})$, $\widehat{n} \neq 0$, and $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{s}_1) \Downarrow'_{d_3} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \Box,\ \mathrm{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \mathrm{if}\ (\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2) \Downarrow'_{iet} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ \mathrm{skip})$ by Vanilla C rule If Else True.

Given $(\gamma,\ \sigma_7) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_6, \Delta_6, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ \mathrm{skip})$.

Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{if}\ (e)\ s_1\ \text{else}\ s_2) \Downarrow^t_{iep} (\gamma,\ \sigma_6,\ \Delta_6,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{if}\ (\widehat{e})\ \widehat{s}_1\ \text{else}\ \widehat{s}_2) \Downarrow'_{iet} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $iep \cong iet$ by Definition 4.2.2.

Subcase $n =_{\text{private}} \text{encrypt}(0)$

Given $n \cong_\psi \widehat{n}$ and $n =_{\text{private}} \text{encrypt}(0)$, we have $\widehat{n} = 0$.

Given $\text{T_resolve}(\sigma_5,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc}+1,\ res_{\text{acc}+1}) = (\sigma_6, \Delta_5)$, by Lemma 4.2.18 we have $\sigma_6 = \sigma'_5 :: \sigma_A$ and $(\gamma, \sigma_6) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{if}\ (\widehat{e})\ \widehat{s}_1\ \text{else}\ \widehat{s}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$, $\widehat{n} = 0$, and $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{s}_2) \Downarrow'_{d_4} (\widehat{\gamma}_2,\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{if}\ (\widehat{e})\ \widehat{s}_1\ \text{else}\ \widehat{s}_2) \Downarrow'_{ief} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ by Vanilla C rule If Else True.

Given $(\gamma,\ \sigma_7) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_6, \Delta_6, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{if}\ (e)\ s_1\ \text{else}\ s_2) \Downarrow^t_{iep} (\gamma,\ \sigma_6, \Delta_6, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{if}\ (\widehat{e})\ \widehat{s}_1\ \text{else}\ \widehat{s}_2) \Downarrow'_{ief} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $iep \cong ief$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{if}\ (e)\ s_1\ \text{else}\ s_2) \Downarrow^t_{iet} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{if}\ (e)\ s_1\ \text{else}\ s_2) \Downarrow^t_{iet} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule If Else True, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n)$, $n \neq 0$, and $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ s_1) \Downarrow^t_{c_2} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc},\ \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{if}\ (\widehat{e})\widehat{s}_1\ \text{else}\ \widehat{s}_2)$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{if}\ (e)\ s_1\ \text{else}\ s_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{if}\ (\widehat{e})\widehat{s}_1\ \text{else}\ \widehat{s}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{if}\ (e)\ s_1\ \text{else}\ s_2 \cong_\psi \text{if}\ (\widehat{e})\ \widehat{s}_1\ \text{else}\ \widehat{s}_2$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{if}\ (e)\ s_1\ \text{else}\ s_2) \Downarrow^t_{iet} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{if}\ (e)\ s_1\ \text{else}\ s_2$. Therefore, by Lemma 3.2.3 we have $\text{if}\ (e)\ s_1\ \text{else}\ s_2 \cong \text{if}\ (\widehat{e})\ \widehat{s}_1\ \text{else}\ \widehat{s}_2$. By Definition 3.2.10, we have $\text{Erase}(\text{if}\ (e)\ s_1\ \text{else}\ s_2) = \text{if}\ (\text{Erase}(e)\ \text{Erase}(s_1)\ \text{else}\ \text{Erase}(s_2)$, $\text{Erase}(e) = \widehat{e}$, $\text{Erase}(s_1) = \widehat{s}_1$, and $\text{Erase}(s_2) = \widehat{s}_2$. Therefore, we have $e \cong \widehat{e}$, $s_1 \cong \widehat{s}_1$, and $s_2 \cong \widehat{s}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, ψ, $(l, \mu) \notin \text{if}\ (e)\ s_1\ \text{else}\ s_2$, and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,$

Δ, $\overline{\chi}$, bid, acc, e). Given (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, n), by the inductive hypothesis we have ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}$) $\Downarrow'_{d_1}$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{n}$) and ψ_1 such that (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, n) $\cong_{\psi_1}$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{n}$) and $c_1 \cong d_1$. Given $n \neq$ skip, by Lemma 4.2.1 we have $\psi_1 = \psi$, therefore (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, n) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{n}$). By Definition 4.2.1 we have (γ, σ_1) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$) and $n \cong_\psi \widehat{n}$. By Definition 3.2.17 we have $n \cong \widehat{n}$. Given Label(e, γ) = public, we have Label(n, γ) = public and therefore $n = \widehat{n}$.

Given $n \neq 0$ and $n = \widehat{n}$, we have $\widehat{n} \neq 0$.

Given (γ, σ_1) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$), ψ, $(l, \mu) \notin$ if (e) s_1 else s_2, and $s_1 \cong \widehat{s}_1$, by Lemma 4.2.3 we have (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, s_1) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{s}_1$). Given (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, s_1) $\Downarrow^t_{c_2}$ (γ_1, σ_2, Δ_2, $\overline{\chi}_1$, bid, acc, skip), by the inductive hypothesis, we have ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{s}_1$) $\Downarrow'_{d_2}$ ($\widehat{\gamma}_1$, $\widehat{\sigma}_2$, $\square$, skip) and ψ_2 such that (γ_1, σ_2, Δ_2, $\overline{\chi}_1$, bid, acc, skip) $\cong_{\psi_2}$ ($\widehat{\gamma}_1$, $\widehat{\sigma}_2$, $\square$, skip) and $c_2 \cong d_2$. By Definition 4.2.1, we have (γ_1, σ_2) $\cong_{\psi_2}$ ($\widehat{\gamma}_1$, $\widehat{\sigma}_2$).

Given ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, if ($\widehat{e}$) $\widehat{s}_1$ else $\widehat{s}_2$), ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}$) $\Downarrow'_{d_1}$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{n}$), $\widehat{n} \neq 0$, and ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{s}_1$) $\Downarrow'_{d_2}$ ($\widehat{\gamma}_1$, $\widehat{\sigma}_2$, $\square$, skip), we have $\Sigma \triangleright$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, if ($\widehat{e}$) $\widehat{s}_1$ else $\widehat{s}_2$) $\Downarrow'_{iet}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, skip) by Vanilla C rule If Else True.

Given (γ, σ_2) $\cong_{\psi_2}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$), by Definition 4.2.1 we have (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, skip) $\cong_{\psi_2}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, skip). Therefore, we have (γ, σ, Δ, $\overline{\chi}$, bid, acc, if (e) s_1 else s_2) $\Downarrow^t_{iet}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, skip) $\cong_{\psi_2}$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, if ($\widehat{e}$) $\widehat{s}_1$ else $\widehat{s}_2$) $\Downarrow'_{iet}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, skip), $\Pi \cong_{\psi_2} \Sigma$, and $iet \cong iet$ by Definition 4.2.2.

Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, if (e) s_1 else s_2) $\Downarrow^t_{ief}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, skip)
This case is similar to Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, if (e) s_1 else s_2) $\Downarrow^t_{iet}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, skip).

Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, &x) $\Downarrow^t_{loc}$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $(l, 0)$)

Given $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, &x) $\Downarrow^t_{loc}$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $(l, 0)$) by Location-tracking SMC^2 rule Address Of, we have $\gamma(x) = (l, ty)$.

Given ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, &$\widehat{x}$) and ψ such that (γ, σ, Δ, $\overline{\chi}$, bid, acc, &x) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, &$\widehat{x}$), by Definition 4.2.1 we have (γ, σ) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$) and &$x \cong_\psi$ &$\widehat{x}$. By Definition 3.2.18 we have Erase(&x) = & Erase(x) and Erase(x) = $\widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l,\ ty)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{ty})$ such that $l = \widehat{l}$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, and $ty \cong \widehat{ty}$ by Lemma 3.2.14.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \&\widehat{x})$ and $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{ty})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \&\widehat{x}) \Downarrow'_{loc} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}, 0))$ by Vanilla C rule Address Of.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$, by Definition 4.2.1 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}, 0))$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \&x) \Downarrow^t_{loc} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \&\widehat{x}) \Downarrow'_{loc} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}, 0))$, $\Pi \cong_\psi \Sigma$, and $loc \cong loc$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{sizeof}(ty)) \Downarrow^t_{ty} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{sizeof}(ty)) \Downarrow^t_{ty} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$ by Location-tracking SMC^2 rule Size of Type, we have $n = \tau(ty)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{sizeof}(\widehat{ty}))$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{sizeof}(ty)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{sizeof}(\widehat{ty}))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{sizeof}(ty) \cong_\psi \text{sizeof}(\widehat{ty})$. By Definition 3.2.18 we have $\text{Erase}(\text{sizeof}(ty)) = \text{sizeof}(\text{Erase}(ty))$ and $\text{Erase}(ty) = \widehat{ty}$. Therefore, we have $ty \cong \widehat{ty}$.

Given $n = \tau(ty)$, $ty \cong \widehat{ty}$ and $\text{Label}(ty, \gamma) = \text{public}$, we have $\widehat{n} = \tau(\widehat{ty})$ and $n = \widehat{n}$ by Lemma 3.2.48. By Definition 3.2.17 we have $n \cong_\psi \widehat{n}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{sizeof}(\widehat{ty}))$ and $\widehat{n} = \tau(\widehat{ty})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{sizeof}(\widehat{ty})) \Downarrow'_{ty} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{n})$ by Vanilla C rule Size of Type.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $n \cong_\psi \widehat{n}$, by Definition 4.2.1 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{n})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{sizeof}(ty)) \Downarrow^t_{ty} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{sizeof}(\widehat{ty})) \Downarrow'_{ty} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{n})$, $\Pi \cong_\psi \Sigma$ and $ty \cong ty$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{while}\ (e)\ s) \Downarrow^t_{wle} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{while}\ (e)\ s) \Downarrow^{t}_{wle} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule While End, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n)$, $\text{Label}(e, \gamma) = \text{public}$, and $n\ =\ 0$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \text{while}\ (\widehat{e})\ \widehat{s})$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{while}\ (e)\ s) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \Box, \text{while}\ (\widehat{e})\ \widehat{s})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{while}\ (e)\ s \cong_{\psi} \text{while}\ (\widehat{e})\ \widehat{s}$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{while}\ (e)\ s) \Downarrow^{t}_{wle} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{while}\ (e)\ s$. Therefore, by Lemma 3.2.3 we have $\text{while}\ (e)\ s \cong \text{while}\ (\widehat{e})\ \widehat{s}$. By Definition 3.2.10 we have $\text{Erase}(\text{while}\ (e)\ \widehat{s}) = \text{while}\ (\text{Erase}(e))\ \text{Erase}(s)$, $\text{Erase}(e) = \widehat{e}$, and $\text{Erase}(s) = \widehat{s}$. Therefore, we have $e \cong \widehat{e}$ and $s \cong \widehat{s}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, $e \cong \widehat{e}$, $(l, \mu) \notin \text{while}\ (e)\ s$, and ψ, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}) \cong_{\psi} (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{n})$ such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{n})$ and $c_1 \cong d_1$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1\ =\ \psi$, therefore $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{n})$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_{\psi} \widehat{n}$. By Definition 3.2.17 we have $n \cong \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$.

Given $n\ =\ 0$ and $n = \widehat{n}$, we have $\widehat{n}\ =\ 0$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \text{while}\ (\widehat{e})\ \widehat{s})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{n})$, and $\widehat{n}\ =\ 0$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \Box, \text{while}\ (\widehat{e})\ \widehat{s}) \Downarrow'_{wle} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \text{skip})$ by Vanilla C rule While End.

Given $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{while}\ (e)\ s) \Downarrow^{t}_{wle} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \text{while}\ (\widehat{e})\ \widehat{s}) \Downarrow'_{wle} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \text{skip})$, $\Pi \cong_{\psi} \Sigma$, and $wle \cong wle$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{while}\ (e)\ s) \Downarrow^{t}_{wlc} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{while}\ (e)\ s) \Downarrow^{t}_{wlc} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule While Continue, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n)$, $n \neq 0$, $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ s) \Downarrow^{t}_{c_2} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc},\ \text{skip})$, and $(\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc}, \text{while}\ (e)\ s) \Downarrow^{t}_{c_3} (\gamma_2, \sigma_3, \Delta_3, \overline{\chi}_2, \text{bid}, \text{acc},\ \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{while}\ (\widehat{e})\ \widehat{s})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{while}\ (e)\ s) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{while}\ (\widehat{e})\ \widehat{s})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{while}\ (e)\ s \cong_{\psi} \text{while}\ (\widehat{e})\ \widehat{s}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{while}\ (e)\ s) \Downarrow^{t}_{wlc} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{while}\ (e)\ s$. Therefore, by Lemma 3.2.3 we have $\text{while}\ (e)\ s \cong \text{while}\ (\widehat{e})\ \widehat{s}$. By Definition 3.2.10 we have $\text{Erase}(\text{while}\ (e)\ \widehat{s}) = \text{while}\ (\text{Erase}(e))\ \text{Erase}(s)$, $\text{Erase}(e) = \widehat{e}$, and $\text{Erase}(s) = \widehat{s}$. Therefore, we have $e \cong_{\psi} \widehat{e}$ and $s \cong_{\psi} \widehat{s}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, $e \cong \widehat{e}$, $(l, \mu) \notin \text{while}\ (e)\ s$, and ψ, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_{\psi} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and $c_1 \cong d_1$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$, therefore $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_{\psi} \widehat{n}$. By Definition 3.2.17 we have $n \cong \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$.

Given $n \neq 0$ and $n = \widehat{n}$, we have $\widehat{n} \neq 0$.

Given $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, $s \cong \widehat{s}$, $(l, \mu) \notin \text{while}\ (e)\ s$, and ψ, by Lemma 4.2.3 we have $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{s})$. Given $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s) \Downarrow^{t}_{c_2} (\gamma_1, \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{s}) \Downarrow'_{d_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ such that $(\gamma_1, \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ and $c_2 \cong d_2$. By Definition 4.2.1, we have $(\gamma_1, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$. By Lemma 4.2.12, we have $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$, $\text{while}\ (e)\ s \cong \text{while}\ (\widehat{e})\ \widehat{s}$, and $(l, \mu) \notin \text{while}\ (e)\ s$, by Lemma 4.2.3 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{while}\ (e)\ s) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{while}\ (\widehat{e})\ \widehat{s})$. Given $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{while}\ (e)\ s) \Downarrow^{t}_{c_3} (\gamma_2,\ \sigma_3,\ \Delta_3,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{while}\ (\widehat{e})\ \widehat{s}) \Downarrow'_{d_3} (\widehat{\gamma}_2,\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ such that $(\gamma_2, \sigma_3,\ \Delta_3,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_3} (\widehat{\gamma}_2,\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ and $c_3 \cong d_3$. By Definition 4.2.1, we have $(\gamma_2, \sigma_3) \cong_{\psi_3} (\widehat{\gamma}_2,\ \widehat{\sigma}_3)$. By Lemma 4.2.12, we have $(\gamma,\ \sigma_3) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{while}\ (\widehat{e})\ \widehat{s})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$, $\widehat{n} \neq 0$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{s}) \Downarrow'_{d_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, and $(\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{while}\ (e)\ s) \Downarrow'_{d_3} (\widehat{\gamma}_2, \widehat{\sigma}_3, \square, \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \text{while}\ (\widehat{e})\ \widehat{s}) \Downarrow'_{wlc} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule While Continue.

Given $(\gamma,\ \sigma_3) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{while}\ (e)\ s) \Downarrow^{t}_{wlc} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{while}$

$(\widehat{e})\ \widehat{s}) \Downarrow'_{wlc} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, $\Pi \cong_{\psi_3} \Sigma$, and $wlc \cong wlc$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1;s_2) \Downarrow^t_{ss} (\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ v)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1;s_2) \Downarrow^t_{ss} (\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ v)$ by Location-tracking SMC^2 rule Statement Sequencing, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1) \Downarrow^t_{c_1} (\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$ and $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ s_2) \Downarrow^t_{c_2} (\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ v)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1;\ \widehat{s}_2)$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1;s_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1;\ \widehat{s}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $s_1;s_2 \cong_\psi \widehat{s}_1;\ \widehat{s}_2$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1;s_2) \Downarrow^t_{ss} (\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ v)$, by Lemma 4.2.2 we have $(l, \mu) \notin s_1;s_2$. Therefore, by Lemma 3.2.3 we have $s_1;s_2 \cong \widehat{s}_1;\ \widehat{s}_2$. By Definition 3.2.10 we have $\text{Erase}(s_1;s_2) = \text{Erase}(s_1);\text{Erase}(s_2)$, $\text{Erase}(s_1) = \widehat{s}_1$, and $\text{Erase}(s_2) = \widehat{s}_2$. Therefore, we have $s_1 \cong_\psi \widehat{s}_1$ and $s_2 \cong \widehat{s}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, $s_1 \cong \widehat{s}_1$, and $(l, \mu) \notin s_1;s_2$, by Lemma 4.2.3 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1) \Downarrow^t_{c_1} (\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1) \Downarrow'_{d_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$ and ψ_1 such that $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1, \square, \text{skip})$ and $c_1 \cong d_1$. By Definition 4.2.1, we have $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, $s_2 \cong \widehat{s}_2$ and $(l, \mu) \notin s_1;s_2$, by Lemma 4.2.3 we have we have $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ s_2) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{s}_2)$. Given $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ s_2) \Downarrow^t_{c_2} (\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ v)$, by the inductive hypothesis, we have $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{s}_2) \Downarrow'_{d_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ such that $(\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ v) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and $c_2 \cong d_2$. By Definition 4.2.1, we have $(\gamma_2, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_2, \widehat{\sigma}_2)$ and $v \cong_{\psi_2} \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1;\ \widehat{s}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1) \Downarrow'_{d_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$, and $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{s}_2) \Downarrow'_{d_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \widehat{s}_1; \widehat{s}_2) \Downarrow'_{ss} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ by Vanilla C rule Statement Sequencing.

Given $(\gamma_2, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2)$ and $v \cong_{\psi_2} \widehat{v}$, by Definition 4.2.1 we have $(\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ v) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1;\ s_2) \Downarrow^t_{ss} (\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ v) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1;\ \widehat{s}_2) \Downarrow'_{ss} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, $\Pi \cong_{\psi_2} \Sigma$, and $ss \cong ss$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ (e)) \Downarrow^t_{ep} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v)$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ (e)) \Downarrow^t_{ep} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v)$ by Location-tracking SMC^2 rule Parentheses, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{e}))$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ (e)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{e}))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $(e) \cong_\psi (\widehat{e})$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ (e)) \Downarrow^t_{ep} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v)$, by Lemma 4.2.2 we have $(l, \mu) \notin (e)$. Therefore, by Lemma 3.2.3 we have $(e) \cong (\widehat{e})$. By Definition 3.2.10 we have $\text{Erase}((e)) = (\text{Erase}(e))$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, $(l, \mu) \notin (e)$, and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ and $c_1 \cong d_1$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_{\psi_1} \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{e}))$ and $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{e})) \Downarrow'_{ep} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ by Vanilla C rule Parentheses.

Given $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_{\psi_1} \widehat{v}$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ (e)) \Downarrow^t_{ep} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{e})) \Downarrow'_{ep} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\Pi \cong_{\psi_1} \Sigma$, and $ep \cong ep$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \{s\}) \Downarrow^t_{sb} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \{s\}) \Downarrow^t_{sb} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Statement Block, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ s) \Downarrow^t_{c_1} (\gamma_1,\ \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc},\ \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \{\widehat{s}\})$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \{s\}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \{\widehat{s}\})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\{s\} \cong_\psi \{\widehat{s}\}$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \{s\}) \Downarrow^t_{sb} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \{s\}$. Therefore, by Lemma 3.2.3 we have $\{s\} \cong \{\widehat{s}\}$. By Definition 3.2.10 we have $\text{Erase}(\{s\}) = \{\text{Erase}(s)\}$ and $\text{Erase}(s) = \widehat{s}$. Therefore, we have $s \cong \widehat{s}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, $(l, \mu) \notin \{s\}$, and $s \cong \widehat{s}$, by Lemma 4.2.3 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s) \cong (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s})$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s) \Downarrow^{t}_{c_1} (\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}) \Downarrow'_{d_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$ and ψ_1 such that $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$ and $c_1 \cong d_1$. By Definition 4.2.1, we have $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \{\widehat{s}\})$ and $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{s}) \Downarrow'_{d_1} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \{\widehat{s}\}) \Downarrow'_{sb} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \text{skip})$ by Vanilla C rule Statement Block.

Given $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \{s\}) \Downarrow^{t}_{sb} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \{\widehat{s}\}) \Downarrow'_{sb}$ $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \text{skip})$, $\Pi \cong_{\psi_1} \Sigma$, and $sb \cong sb$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (ty)\ e) \Downarrow^{t}_{cl} (\gamma,\ \sigma_3,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0))$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (ty)\ e) \Downarrow^{t}_{cl} (\gamma,\ \sigma_3,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0))$ by Location-tracking SMC^2 rule Cast Public Location, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0))$, $(ty = \text{public}\ bty*) \vee (ty = \text{char}*)$, $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega,\ \text{void}, n,\ \text{PermL}(\text{Freeable},\ \text{void},\ \text{public},\ n)\big)\big]$, and $\sigma_3 = \sigma_2\big[l \rightarrow \big(\omega, ty, \frac{n}{\tau(ty)},\ \text{PermL}\big(\text{Freeable},\ ty,\ \text{public},\ \frac{n}{\tau(ty)}\big)\big)\big]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (ty)\ e) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $(ty)\ e \cong_{\psi} (\widehat{ty})\ \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (ty)\ e) \Downarrow^{t}_{cl} (\gamma,\ \sigma_3,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0))$, by Lemma 4.2.2 we have $(l, \mu) \notin (ty)\ e$. Therefore, by Lemma 3.2.3 we have $(ty)\ e \cong (\widehat{ty})\ \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}((ty)\ e) = (\text{Erase}(ty))\ \text{Erase}(e)$, $\text{Erase}(ty) = \widehat{ty}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, $(l, \mu) \notin (ty)\ e$, and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_{\psi}$ $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$ by Definition 4.2.1. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0))$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}, 0))$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0))$ $\cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}, 0))$ and $c_1 \cong d_1$. Given $(l, 0) \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l, 0) \cong_{\psi} (\widehat{l}, 0)$. By Definition 3.2.13 we have $l \cong_{\psi} \widehat{l}$.

Given $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega,\ \text{void}, n,\ \text{PermL}(\text{Freeable},\ \text{void},\ \text{public}, n)\big)\big]$, $l \cong_{\psi} \widehat{l}$, and $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, by

Lemma 3.2.36 we have $\widehat{\sigma}_1 = \widehat{\sigma}_2\big[\widehat{l} \rightarrow \big(\widehat{\omega}, \text{void}, \widehat{n}, \text{PermL}(\text{Freeable}, \text{void}, \text{public}, \widehat{n})\big)\big]$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$, $ty \cong \widehat{ty}$, and $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_3 = \sigma_2\big[l \rightarrow \big(\omega, ty, \frac{n}{\tau(ty)}, \text{PermL}\big(\text{Freeable}, ty, \text{public}, \frac{n}{\tau(ty)}\big)\big)\big]$, $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$, $l \cong_\psi \widehat{l}$, $ty \cong \widehat{ty}$, $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.35 we have $\widehat{\sigma}_3 = \widehat{\sigma}_2\big[\widehat{l} \rightarrow \big(\widehat{\omega}, \widehat{ty}, \frac{\widehat{n}}{\tau(\widehat{ty})}, \text{PermL}\big(\text{Freeable}, \widehat{ty}, \text{public}, \frac{\widehat{n}}{\tau(\widehat{ty})}\big)\big)\big]$ such that $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e})$, $(\widehat{ty} = \widehat{bty*})$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, (\widehat{l}, 0))$, $\widehat{\sigma}_1 = \widehat{\sigma}_2\big[\widehat{l} \rightarrow \big(\widehat{\omega}, \text{void}, \widehat{n}, \text{PermL}(\text{Freeable}, \text{void}, \text{public}, \widehat{n})\big)\big]$, and $\widehat{\sigma}_3 = \widehat{\sigma}_2\big[\widehat{l} \rightarrow \big(\widehat{\omega}, \widehat{ty}, \frac{\widehat{n}}{\tau(\widehat{ty})}, \text{PermL}\big(\text{Freeable}, \widehat{ty}, \text{public}, \frac{\widehat{n}}{\tau(\widehat{ty})}\big)\big)\big]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}) \Downarrow'_{cl} (\widehat{\gamma}, \widehat{\sigma}_3, \square, (\widehat{l}, 0))$ by Vanilla C rule Cast Location.

Given $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$, by Definition 4.2.1 we have $(\gamma, \sigma_3, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3, \square, (\widehat{l}, 0))$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cl} (\gamma, \sigma_3, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}) \Downarrow'_{cl} (\widehat{\gamma}, \widehat{\sigma}_3, \square, (\widehat{l}, 0))$, $\Pi \cong_\psi \Sigma$, and $cl1 \cong cl$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cl1} (\gamma, \sigma_3, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cl1} (\gamma, \sigma_3, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$ by Location-tracking SMC^2 rule Case Private Location, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$, $(ty = \text{private int*}) \vee (ty = \text{private float*}) \vee (ty = \text{int*}) \vee (ty = \text{float*})$, $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega, \text{void}, n, \text{PermL}(\text{Freeable}, \text{void}, \text{private}, n)\big)\big]$, and $\sigma_3 = \sigma_2\big[l \rightarrow \big(\omega, ty, \frac{n}{\tau(ty)}, \text{PermL}\big(\text{Freeable}, ty, \text{private}, \frac{n}{\tau(ty)}\big)\big)\big]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e})$ and ψ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e})$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $(ty)\ e \cong_\psi (\widehat{ty})\ \widehat{e}$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cl1} (\gamma, \sigma_3, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$, by Lemma 4.2.2 we have $(l, \mu) \notin (ty)\ e$. Therefore, by Lemma 3.2.3 we have $(ty)\ e \cong (\widehat{ty})\ \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}((ty)\ e) = (\text{Erase}(ty))\ \text{Erase}(e)$, $\text{Erase}(ty) = \widehat{ty}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, \mu) \notin (ty)\ e$, and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e)$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, (\widehat{l}, 0))$ and ψ_1 such that $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0)) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, (\widehat{l}, 0))$ and $c_1 \cong d_1$. Given $(l, 0) \neq \text{skip}$, by Lemma 4.2.1 we have $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0)) \cong_\psi$

$(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}, 0))$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$.

Given $\sigma_1 = \sigma_2\big[l \to \big(\omega,\ \text{void}*,\ n,\ \text{PermL}(\text{Freeable},\ ty,\ \text{private},\ n)\big)\big]$, $l \cong_\psi \widehat{l}$, $ty \cong \widehat{ty}$, and $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, by Lemma 3.2.36 we have $\widehat{\sigma}_1 = \widehat{\sigma}_2\big[\widehat{l} \to \big(\widehat{\omega},\ \text{void}*,\ \widehat{n}, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, \widehat{n})\big)\big]$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, $\omega \cong_\psi \widehat{\omega}$, and $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$.

Given $\sigma_3 = \sigma_2\big[l \to \big(\omega,\ ty,\ \frac{n}{\tau(ty)},\ \text{PermL}\big(\text{Freeable},\ ty,\ \text{private},\ \frac{n}{\tau(ty)}\big)\big)\big]$, $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$, $l \cong_\psi \widehat{l}$, $ty \cong \widehat{ty}$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.35 we have $\widehat{\sigma}_3 = \widehat{\sigma}_2\big[\widehat{l} \to \big(\widehat{\omega},\ \widehat{ty},\ \frac{\widehat{n}}{\tau(\widehat{ty})}, \text{PermL}\big(\text{Freeable}, \widehat{ty}, \text{public}, \frac{\widehat{n}}{\tau(\widehat{ty})}\big)\big)\big]$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e})$, $(\widehat{ty} = \widehat{bty}*)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}, 0))$, $\widehat{\sigma}_1 = \widehat{\sigma}_2\big[\widehat{l} \to \big(\widehat{\omega},\ \text{void},\ \widehat{n}, \text{PermL}(\text{Freeable}, \text{void}, \text{public}, \widehat{n})\big)\big]$, and $\widehat{\sigma}_3 = \widehat{\sigma}_2\big[\widehat{l} \to \big(\widehat{\omega},\ \widehat{ty},\ \frac{\widehat{n}}{\tau(\widehat{ty})}, \text{PermL}\big(\text{Freeable},\ \widehat{ty}, \text{public},\ \frac{\widehat{n}}{\tau(\widehat{ty})}\big)\big)\big]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e}) \Downarrow'_{cl} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ (\widehat{l}, 0))$ by Vanilla C rule Cast Location.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_3,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ (\widehat{l}, 0))$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, (ty)\ e) \Downarrow^t_{cl1} (\gamma,\ \sigma_3,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ (\widehat{l}, 0))$, $\Pi \cong_\psi \Sigma$ and $cl \cong cl$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (ty)\ e) \Downarrow^t_{cv} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (ty)\ e) \Downarrow^t_{cv} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1)$ by Location-tracking SMC^2 rule Cast Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, $(ty = \text{public int}) \vee (ty = \text{public float})$, and $n_1 = \text{Cast}(\text{public}, ty, n)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (ty)\ e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $(ty)\ e \cong_\psi (\widehat{ty})\ \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, (ty)\ e) \Downarrow^t_{cv} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1)$, by Lemma 4.2.2 we have $(l, \mu) \notin (ty)\ e$. Therefore, by Lemma 3.2.3 we have $(ty)\ e \cong (\widehat{ty})\ \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}((ty)\ e) = (\text{Erase}(ty))\ \text{Erase}(e)$, $\text{Erase}(ty) = \widehat{ty}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, $(l, \mu) \notin (ty)\ e$, and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, by the inductive

hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and $c_1 \cong d_1$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_{\psi} \widehat{n}$. By Definition 3.2.17 we have $n \cong \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.10.

Given $n_1 = \text{Cast}(\text{public}, ty, n)$, $ty \cong \widehat{ty}$, and $n = \widehat{n}$, by Lemma 3.2.24 we have $\widehat{n}_1 = \text{Cast}(\text{public},\ \widehat{ty},\ \widehat{n})$ such that $n_1 = \widehat{n}_1$. By Definition 3.2.10 we have $n_1 \cong \widehat{n}_1$, and by Definition 3.2.17 we have $n_1 \cong_{\psi} \widehat{n}_1$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$, and $\widehat{n}_1 = \text{Cast}(\text{public},\ \widehat{ty},\ \widehat{n})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e}) \Downarrow'_{cv} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$ by Vanilla C rule Cast Value.

Given $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n_1 \cong_{\psi} \widehat{n}_1$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1) \cong (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ (ty)\ e) \Downarrow^t_{cv} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e}) \Downarrow'_{cv} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}_1)$, $\Pi \cong_{\psi} \Sigma$ and $cv \cong cv$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ (ty)\ e) \Downarrow^t_{cv1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ (ty)\ e) \Downarrow^t_{cv1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$ by Location-tracking SMC^2 rule Cast Private Value, we have $\text{Label}(e, \gamma) = \text{private}$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n)$, $(ty = \text{private int}) \vee (ty = \text{private float}) \vee (ty = \text{int}) \vee (ty = \text{float})$, and $n_1 = \text{Cast}(\text{private}, ty, n)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e})$ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ (ty)\ e) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{ty})\ \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $(ty)\ e \cong_{\psi} (\widehat{ty})\ \widehat{e}$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cv1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, by Lemma 4.2.2 we have $(l, \mu) \notin (ty)\ e$. Therefore, by Lemma 3.2.3 we have $(ty)\ e \cong (\widehat{ty})\ \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}((ty)\ e) = (\text{Erase}(ty))\ \text{Erase}(e)$, $\text{Erase}(ty) = \widehat{ty}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, $(l, \mu) \notin (ty)\ e$, and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and $c_1 \cong d_1$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_{\psi} \widehat{n}$.

Given $n_1 = \text{Cast}(\text{private}, ty, n)$, $ty \cong \widehat{ty}$, and $n \cong_\psi \widehat{n}$, by Lemma 3.2.25 we have $\widehat{n}_1 = \text{Cast}(\text{public}, \widehat{ty}, \widehat{n})$ such that $n_1 \cong_\psi \widehat{n}_1$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e})$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n})$, and $\widehat{n}_1 = \text{Cast}(\text{public}, \widehat{ty}, \widehat{n})$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}) \Downarrow'_{cv} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$ by Vanilla C rule Cast Value.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $n_1 \cong_\psi \widehat{n}_1$, by Definition 4.2.1 we have $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cv1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{ty})\ \widehat{e}) \Downarrow'_{cv} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n}_1)$, $\Pi \cong_\psi \Sigma$ and $cv1 \cong cv$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow^t_{inp} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1,e_2)) \Downarrow^t_{inp} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by Location-tracking SMC^2 rule SMC Input Public Value, we have $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$, $\text{acc} = 0$, $(\gamma\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n)$, $\gamma(x) = (l, \text{public}\ bty)$, $\text{InputValue}(x, n) = n_1$, and $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, x = n_1) \Downarrow^t_{c_3} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2))$ and ψ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2))$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $\text{smcinput}(e_1, e_2) \cong_\psi \text{mcinput}(\widehat{e}_1, \widehat{e}_2)$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow^t_{inp} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{smcinput}(e_1,e_2)$. Therefore, by Lemma 3.2.3 we have $\text{smcinput}(e_1,e_2) \cong \text{mcinput}(\widehat{e}_1, \widehat{e}_2)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcinput}(e_1, e_2)) = \text{mcinput}(\text{Erase}(e_1, e_2))$. By Definition 3.2.8, we have $\text{Erase}(e_1, e_2) = \text{Erase}(e_1), \text{Erase}(e_2))$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$ and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$, and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, Lemma 4.2.3 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1)$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{x})$ and ψ_1 such that $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{x})$ and $c_1 \cong d_1$. Given $x \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \cong_\psi (\gamma,\ \sigma_1,\ \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2)$. Given $(\gamma\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \widehat{n})$ and ψ_2 such that $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$ and $c_2 \cong d_2$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. By Definition 3.2.17 we have $n \cong \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.10.

Given $\gamma(x) = (l, \text{public}\ bty)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by $\text{public}\ bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\text{InputValue}(x, n) = n_1$, $x = \widehat{x}$, and $n = \widehat{n}$, by Axiom 3.2.4 and Lemma 3.2.26 we have $\text{InputValue}(\widehat{x},\ \widehat{n}) = \widehat{n}_1$ such that $n_1 \cong \widehat{n}_1$. By Definition 3.2.17 we have $n_1 \cong_\psi \widehat{n}_1$.

Given $x = \widehat{x}$ and $n_1 \cong \widehat{n}_1$, by Definition 3.2.10 we have $x = n_1 \cong \widehat{x} = \widehat{n}_1$, and by Definition 3.2.18 we have $x = n_1 \cong_\psi \widehat{x} = \widehat{n}_1$. Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, we have $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{x} = \widehat{n}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{x} = \widehat{n}_1) \cong_\psi (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ x = n_1)$ by Definition 4.2.1. Given $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ x = n_1) \Downarrow^t_{c_3} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{x} = \widehat{n}_1) \Downarrow'_{d_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ and ψ_3 such that $(\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ and $c_3 \cong d_3$. Given $\text{pfree}(e) \notin x = n_1$, by Definition 3.2.11 we have $\psi_3 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square, \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty})$, $\text{InputValue}(\widehat{x}, \widehat{n}) = \widehat{n}_1$, and $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{x} = \widehat{v}) \Downarrow'_{d_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2)) \Downarrow'_{inp} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, \text{skip})$ by Vanilla C rule Input Value.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{smcinput}(e_1,e_2)) \Downarrow^t_{inp} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2)) \Downarrow'_{inp} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, \text{skip})$, $\Pi \cong_\psi \Sigma$ and $inp \cong inp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{smcinput}(e_1,e_2)) \Downarrow^t_{inp3} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1,\ e_2)) \Downarrow^t_{inp3} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Input Private Variable, we have $\text{Label}(e_2, \gamma) = \text{private}$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid},$

acc, x), ($\gamma\ \sigma_1, \Delta_1, \overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c_2}$ (γ, $\sigma_2, \Delta_2, \overline{\chi}$, bid, acc, n), $\gamma(x) = (l, \text{private } bty)$, InputValue$(x, n) = n_1$, and ($\gamma$, σ_2, Δ_2, $\overline{\chi}$, bid, acc, $x = n_1$) $\Downarrow^t_{c_3}$ (γ, σ_3, Δ_3, $\overline{\chi}$, bid, acc, skip).

Given ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, mcinput$(\widehat{e}_1, \widehat{e}_2)$) and ψ such that (γ, σ, Δ, $\overline{\chi}$, bid, acc, smcinput(e_1, e_2)) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, mcinput$(\widehat{e}_1, \widehat{e}_2)$), by Definition 4.2.1 we have (γ, σ) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$) and smcinput(e_1, e_2) $\cong_\psi$ mcinput$(\widehat{e}_1, \widehat{e}_2)$. Given ($\gamma$, σ, Δ, $\overline{\chi}$, bid, acc, smcinput(e_1, e_2)) $\Downarrow^t_{inp3}$ (γ, σ_3, Δ_3, $\overline{\chi}$, bid, acc, skip), by Lemma 4.2.2 we have $(l, \mu) \notin$ smcinput(e_1, e_2). Therefore, by Lemma 3.2.3 we have smcinput$(e_1, e_2) \cong$ mcinput$(\widehat{e}_1, \widehat{e}_2)$. By Definition 3.2.10 we have Erase(smcinput(e_1, e_2)) = mcinput(Erase(e_1, e_2)). By Definition 3.2.8, we have Erase(e_1, e_2) = Erase(e_1), Erase(e_2)). By Definition 3.2.10 we have Erase$(e_1) = \widehat{e}_1$ and Erase$(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$, and $e_2 \cong \widehat{e}_2$.

Given (γ, σ) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$) and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}_1$) such that ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}_1$) $\cong_\psi$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, e_1). Given (γ, σ, Δ, $\overline{\chi}$, bid, acc, e_1) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, x), by the inductive hypothesis we have ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}_1$) $\Downarrow'_{d_1}$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{x}$) and ψ_1 such that (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, x) $\cong_{\psi_1}$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{x}$) and $c_1 \cong d_1$. Given $x \neq$ skip, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have (γ, σ_1) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$) and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given (γ, σ_1) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$) and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{e}_2$) such that ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{e}_2$) $\cong_\psi$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, e_2). Given ($\gamma\ \sigma_1$, Δ_1, $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c_2}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n), by the inductive hypothesis we have ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{e}_2$) $\Downarrow'_{d_2}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, $\widehat{n}$) and ψ_2 such that (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n) $\cong_{\psi_2}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, $\widehat{n}$) and $c_2 \cong d_2$. Given $n \neq$ skip, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have (γ, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$) and $n \cong_\psi \widehat{n}$. Given Label(e_2, γ) = public, we have Label(n, γ) = public and therefore $n = \widehat{n}$ by Definition 3.2.17 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{private } bty)$, ($\gamma$, σ) $\cong$ ($\widehat{\gamma}$, $\widehat{\sigma}$), and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by private $bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given InputValue$(x, n) = n_1$, $x = \widehat{x}$, and $n = \widehat{n}$, by Axiom 3.2.4 and Lemma 3.2.26 we have InputValue$(\widehat{x}, \widehat{n}) = \widehat{n}_1$ such that $n_1 \cong \widehat{n}_1$.

Given $x = \widehat{x}$ and $n_1 \cong \widehat{n}_1$, by Definition 3.2.10 we have $x = n_1 \cong \widehat{x} = \widehat{n}_1$. Given ($\gamma$, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$), by Lemma 4.2.3 we have ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, $\widehat{x} = \widehat{n}_1$) such that ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, $\widehat{x} = \widehat{n}_1$) $\cong_\psi$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, $x = n_1$). Given (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, $x = n_1$) $\Downarrow^t_{c_3}$ (γ, σ_3, Δ_3, $\overline{\chi}$, bid, acc, skip), by the inductive hypothesis, we have

$(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{x} = \widehat{n}_1) \Downarrow'_{d_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ and ψ_3 such that $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ and $c_3 \cong d_3$. Given $\text{pfree}(e) \notin$, we have $\psi_3 = \psi_1$. By Definition 4.2.1 we have $(\gamma,\ \sigma_3) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square, \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty})$, $\text{InputValue}(\widehat{x},\ \widehat{n}) = \widehat{n}_1$, and $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{x} = \widehat{v}) \Downarrow'_{d_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2)) \Downarrow'_{inp} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square, \text{skip})$ by Vanilla C rule Input Value.

Given $(\gamma,\ \sigma_3) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{smcinput}(e_1, e_2)) \Downarrow^t_{inp3} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2)) \Downarrow'_{inp} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, $\Pi \cong_{\psi} \Sigma$ and $inp3 \cong inp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp1} (\gamma,\ \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp1} (\gamma,\ \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule SMC Input Public 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ x)$, $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n)$, $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ e_3) \Downarrow^t_{c_3} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, $\gamma(x) = (l, \text{public const } bty*)$, $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, and $(\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, x = [m_0, ..., m_{n_1}]) \Downarrow^t_{c_4} (\gamma,\ \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1, \widehat{e}_2, e_3))$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2)) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{smcinput}(e_1, e_2, e_3) \cong_{\psi} \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp1} (\gamma,\ \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{smcinput}(e_1, e_2, e_3)$. Therefore, by Lemma 3.2.3 we have $\text{smcinput}(e_1, e_2, e_3) \cong \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcoutput}(e_1, e_2, e_3)) = \text{mcinput}(\text{Erase}(e_1, e_2, e_3))$. By Definition 3.2.8, we have $\text{Erase}(e_1, e_2, e_3) = \text{Erase}(e_1), \text{Erase}(e_2), \text{Erase}(e_3)$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$, $\text{Erase}(e_2) = \widehat{e}_2$, and $\text{Erase}(e_3) = \widehat{e}_3$. Therefore, we have $e_1 \cong \widehat{e}_1$, $e_2 \cong \widehat{e}_2$, and $e_3 \cong \widehat{e}_3$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ x)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and $c_1 \cong d_1$. Given $x \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_{\psi} \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$. Given $(\gamma\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \widehat{n})$ and ψ_2 such that $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$ and $c_2 \cong d_2$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $e_3 \cong \widehat{e}_3$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3)$ such that $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3)$. Given $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_3) \Downarrow^t_{c_3} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \Downarrow'_{d_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$ and ψ_3 such that $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$ and $c_3 \cong d_3$. Given $n_1 \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_3 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$ and $n_1 \cong_\psi \widehat{n}_1$. Given $\text{Label}(e_3, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$ and therefore $n_1 = \widehat{n}_1$ by Definition 3.2.18 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, $x = \widehat{x}$, $n = \widehat{n}$, $n_1 = \widehat{n}_1$, by Axiom 3.2.4 and Lemma 3.2.27 we have $\text{InputArray}(\widehat{x}, \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$ such that $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$.

Given $x = \widehat{x}$ and $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, by Definition 3.2.10 we have $x = [m_0, ..., m_{n_1}] \cong \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$. Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, we have $(\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}])$ such that $(\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \cong_\psi (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = [m_0, ..., m_{n_1}])$ by Lemma 4.2.3. Given $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = [m_0, ..., m_{n_1}]) \Downarrow^t_{c_4} (\gamma,\ \sigma_4,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \Downarrow'_{d_4} (\widehat{\gamma},\ \widehat{\sigma}_4\ \square,\ \text{skip})$ and ψ_4 such that $(\gamma,\ \sigma_4,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_4} (\widehat{\gamma},\ \widehat{\sigma}_4,\ \square, \text{skip})$ and $c_4 \cong d_4$. Given $\text{pfree}(e) \notin x = [m_0, ..., m_{n_1}]$, we have $\psi_4 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_4) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_4)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n})$, $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \Downarrow'_{d_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\text{InputArray}(\widehat{x}, \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, and $(\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \Downarrow'_{d_4} (\widehat{\gamma}, \widehat{\sigma}_4, \square, \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3)) \Downarrow'_{inp1} (\widehat{\gamma},\ \widehat{\sigma}_4,\ \square, \text{skip})$ by Vanilla C rule Input 1D Array.

Given $(\gamma,\ \sigma_4) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_4)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_4,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_4,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp1} (\gamma,\ \sigma_4,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)) \Downarrow'_{inp1} (\widehat{\gamma},\ \widehat{\sigma}_4,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $inp1 \cong inp1$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp4} (\gamma,\ \sigma_4,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp4} (\gamma,\ \sigma_4,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule SMC Input Private 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x)$, $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_3) \Downarrow^t_{c_3} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n_1)$, $\gamma(x) = (l, \text{private const } bty*)$, $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, and $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = [m_0, ..., m_{n_1}]) \Downarrow^t_{c_4} (\gamma,\ \sigma_4,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1, \widehat{e}_2, e_3))$ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcinput}(e_1, e_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{smcinput}(e_1,\ e_2,\ e_3) \cong_\psi \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp4} (\gamma,\ \sigma_4,\ \Delta_4,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{smcinput}(e_1, e_2, e_3)$. Therefore, by Lemma 3.2.3 we have $\text{smcinput}(e_1, e_2, e_3) \cong \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcoutput}(e_1,\ e_2,\ e_3)) = \text{mcinput}(\text{Erase}(e_1,\ e_2,\ e_3))$. By Definition 3.2.8, we have $\text{Erase}(e_1, e_2, e_3) = \text{Erase}(e_1), \text{Erase}(e_2), \text{Erase}(e_3)$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$, $\text{Erase}(e_2) = \widehat{e}_2$, and $\text{Erase}(e_3) = \widehat{e}_3$. Therefore, we have $e_1 \cong \widehat{e}_1$, $e_2 \cong \widehat{e}_2$, and $e_3 \cong \widehat{e}_3$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and $c_1 \cong d_1$. Given $x \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$. Given $(\gamma\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \widehat{n})$ and ψ_2 such that $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$ and $c_2 \cong d_2$. Given $\neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $e_3 \cong \widehat{e}_3$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3)$ such that $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ e_3) \cong (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3)$. Given $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ e_3) \Downarrow^t_{c_3} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \Downarrow'_{d_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$ and ψ_3 such that $(\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ n_1) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$ and $c_3 \cong d_3$. Given $n_1 \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_3 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$ and $n_1 \cong_\psi \widehat{n}_1$. Given $\text{Label}(e_3, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$ and therefore $n_1 = \widehat{n}_1$ by Definition 3.2.18 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, $x = \widehat{x}$, $n = \widehat{n}$, $n_1 = \widehat{n}_1$, by Axiom 3.2.4 and Lemma 3.2.27 we have $\text{InputArray}(\widehat{x}, \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$ such that $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$.

Given $x = \widehat{x}$ and $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, by Definition 3.2.10 we have $x = [m_0, ..., m_{n_1}] \cong \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$. Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}])$ such that $(\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \cong_\psi (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ x = [m_0, ..., m_{n_1}])$. Given $(\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ x = [m_0, ..., m_{n_1}]) \Downarrow^t_{c_4} (\gamma,\ \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \Downarrow'_{d_4} (\widehat{\gamma},\ \widehat{\sigma}_4\ \square, \text{skip})$ and ψ_4 such that $(\gamma,\ \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_{\psi_4} (\widehat{\gamma},\ \widehat{\sigma}_4,\ \square,\ \text{skip})$ and $c_4 \cong d_4$. Given $\text{pfree}(e) \notin x = [m_0, ..., m_{n_1}]$, we have $\psi_4 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_4) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_4)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \widehat{n})$, $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \Downarrow'_{d_3} (\widehat{\gamma},\ \widehat{\sigma}_3, \square, \widehat{n}_1)$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\text{InputArray}(\widehat{x}, \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, and $(\widehat{\gamma}, \widehat{\sigma}_3, \square, \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]) \Downarrow'_{d_4} (\widehat{\gamma}, \widehat{\sigma}_4, \square, \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3)) \Downarrow'_{out1} (\widehat{\gamma},\ \widehat{\sigma}_4,\ \square, \text{skip})$ by Vanilla C rule Input 1D Array.

Given $(\gamma,\ \sigma_4) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_4)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_4,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp4} (\gamma,\ \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcinput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)) \Downarrow'_{out1} (\widehat{\gamma},\ \widehat{\sigma}_4,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $inp4 \cong inp1$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{smcoutput}(e_1, e_2)) \Downarrow^t_{out} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcoutput}(e_1, e_2)) \Downarrow^t_{out} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule SMC Output Public Value, we have $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \sigma_1, \text{acc},\ x)$, $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, $\gamma(x) = (l,\ \text{public}\ bty)$, $\sigma_2(l) = (\omega,\ \text{public}\ bty,\ 1,\ \text{PermL}(\text{Freeable},\ \text{public}\ bty,\ \text{public},\ 1))$, $\text{DecodeVal}(\text{public}\ bty,\ 1, \omega) = n_1$, and $\text{OutputValue}(x,\ n,\ n_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcoutput}(\widehat{e}_1, \widehat{e}_2))$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcoutput}(e_1, e_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{smcoutput}(e_1,\ e_2) \cong_\psi \text{mcinput}(\widehat{e}_1,\ \widehat{e}_2)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcoutput}(e_1, e_2)) \Downarrow^t_{out} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{smcoutput}(e_1, e_2)$. Therefore, by Lemma 3.2.3 we have $\text{smcoutput}(e_1,\ e_2) \cong \text{mcoutput}(\widehat{e}_1, \widehat{e}_2)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcoutput}(e_1,\ e_2)) = \text{mcoutput}(\text{Erase}(e_1, e_2))$. By Definition 3.2.8, we have $\text{Erase}(e_1, e_2) = \text{Erase}(e_1), \text{Erase}(e_2))$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$ and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$, and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and $c_1 \cong d_1$. Given $x \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \cong_\psi (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)$. Given $(\gamma\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \widehat{n})$ and ψ_2 such that $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$ and $c_2 \cong d_2$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{public}\ bty)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by $\text{public}\ bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega, \text{public}\ bty, 1, \text{PermL}(\text{Freeable},\ \text{public}\ bty,\ \text{public},\ 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, bty, \text{public}, 1))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$.

Given $\mathrm{DecodeVal}(\text{public } bty, 1, \omega) = n_1$, $\text{public } bty \cong \widehat{bty}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.41 we have $\mathrm{DecodeVal}(\widehat{bty}, 1, \widehat{\omega}) = \widehat{n}_1$ and $n_1 \cong \widehat{n}_1$.

Given $\mathrm{OutputValue}(x, n, n_1)$, $x = \widehat{x}$, $n = \widehat{n}$, and $n_1 \cong \widehat{n}_1$, by Lemma 3.2.28 we have $\mathrm{OutputValue}(\widehat{x}, \widehat{n}, \widehat{n}_1)$ such that the corresponding output files are congruent.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2))$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{x})$, $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \mathrm{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))$, $\mathrm{DecodeVal}(\widehat{bty}, 1, \widehat{\omega}) = \widehat{v}$, and $\mathrm{OutputValue}(\widehat{x}, \widehat{n}, \widehat{v})$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2)) \Downarrow'_{out} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \mathrm{skip})$ by Vanilla C rule Output Value.

Given $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \square, \mathrm{skip})$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \mathrm{smcoutput}(e_1, e_2)) \Downarrow^t_{out} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2)) \Downarrow'_{out} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \mathrm{skip})$, $\Pi \cong_\psi \Sigma$, and $out \cong out$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \mathrm{smcoutput}(e_1, e_2)) \Downarrow^t_{out3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \mathrm{smcoutput}(e_1, e_2)) \Downarrow^t_{out3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by Location-tracking SMC[2] rule SMC Output Private Value, we have $\mathrm{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n)$, $\gamma(x) = (l, \text{private } bty)$, $\sigma_2(l) = (\omega, \text{private } bty, 1, \mathrm{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))$, $\mathrm{DecodeVal}(\text{private } bty, 1, \omega) = n_1$, and $\mathrm{OutputValue}(x, n, n_1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2))$ and ψ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \mathrm{smcoutput}(e_1, e_2)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2))$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $\mathrm{smcoutput}(e_1, e_2) \cong_\psi \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2)$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \mathrm{smcoutput}(e_1, e_2)) \Downarrow^t_{out3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \mathrm{smcoutput}(e_1, e_2)$. Therefore, by Lemma 3.2.3 we have $\mathrm{smcoutput}(e_1, e_2) \cong \mathrm{mcoutput}(\widehat{e}_1, \widehat{e}_2)$. By Definition 3.2.10 we have $\mathrm{Erase}(\mathrm{smcoutput}(e_1, e_2)) = \mathrm{mcoutput}(\mathrm{Erase}(e_1, e_2))$. By Definition 3.2.8, we have $\mathrm{Erase}(e_1, e_2) = \mathrm{Erase}(e_1), \mathrm{Erase}(e_2))$. By Definition 3.2.10 we have $\mathrm{Erase}(e_1) = \widehat{e}_1$ and $\mathrm{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$, and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$, by the inductive hypothesis we

have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and $c_1 \cong d_1$. Given $x \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$. Given $(\gamma\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \widehat{n})$ and ψ_2 such that $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$ and $c_2 \cong d_2$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{private } bty)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by $\text{private } bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, bty, \text{public}, 1))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$.

Given $\text{DecodeVal}(\text{private } bty, 1, \omega) = n_1$, $\text{private } bty \cong_\psi \widehat{bty}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, 1, \widehat{\omega}) = \widehat{n}_1$ and $n_1 \cong \widehat{n}_1$.

Given $\text{OutputValue}(x, n, n_1)$, $x = \widehat{x}$, $n = \widehat{n}$, and $n_1 \cong \widehat{n}_1$, by Lemma 3.2.28 we have $\text{OutputValue}(\widehat{x},\ \widehat{n},\ \widehat{n}_1)$ such that the corresponding output files are congruent.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcoutput}(\widehat{e}_1,\ \widehat{e}_2))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square, \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty})$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega},\ \widehat{bty},\ 1,\ \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))$, $\text{DecodeVal}(\widehat{bty},\ 1,\ \widehat{\omega}) = \widehat{n}_1$, and $\text{OutputValue}(\widehat{x}, \widehat{n}, \widehat{n}_1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcoutput}(\widehat{e}_1,\ \widehat{e}_2)) \Downarrow'_{out} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ by Vanilla C rule Output Value.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcoutput}(e_1, e_2)) \Downarrow^t_{out3} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcoutput}(\widehat{e}_1,\ \widehat{e}_2)) \Downarrow'_{out} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $out3 \cong out$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out1} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out1} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule SMC Output Public 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ x)$, $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n)$, $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ e_3) \Downarrow^t_{c_3} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_3(l) = (\omega,\ \text{public const } bty*, 1,\ \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_3(l_1) = (\omega_1, \text{public } bty, n_1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n_1))$, $\text{DecodeVal}(\text{public } bty, n_1, \omega_1) = [m_0, ..., m_{n_1}]$, and $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{smcoutput}(e_1, e_2, e_3)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{smcoutput}(e_1, e_2, e_3) \cong_\psi \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out1} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{smcoutput}(e_1, e_2, e_3)$. Therefore, by Lemma 3.2.3 we have $\text{smcoutput}(e_1, e_2, e_3) \cong \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcoutput}(e_1, e_2, e_3)) = \text{mcoutput}(\text{Erase}(e_1, e_2, e_3))$. By Definition 3.2.8, we have $\text{Erase}(e_1, e_2, e_3) = \text{Erase}(e_1), \text{Erase}(e_2), \text{Erase}(e_3)$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$, $\text{Erase}(e_2) = \widehat{e}_2$, and $\text{Erase}(e_3) = \widehat{e}_3$. Therefore, we have $e_1 \cong \widehat{e}_1$, $e_2 \cong \widehat{e}_2$, and $e_3 \cong \widehat{e}_3$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \cong_\psi (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ x)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and $c_1 \cong d_1$. Given $x \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \cong_\psi (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2)$. Given $(\gamma\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \widehat{n})$ and ψ_2 such that $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ n) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$ and $c_2 \cong d_2$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $e_3 \cong \widehat{e}_3$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \cong_\psi (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ e_3)$ by Definition 4.2.1. Given $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ e_3) \Downarrow^t_{c_3} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \Downarrow'_{d_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$ and ψ_3 such that $(\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$

$\cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$ and $c_3 \cong d_3$. Given $n_1 \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_3 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$ and $n_1 \cong_\psi \widehat{n}_1$. Given $\text{Label}(e_3, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$ and therefore $n_1 = \widehat{n}_1$ by Definition 3.2.18 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_3(l) = (\omega,\ \text{public const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_3(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{const } \widehat{bty}*,\ \text{public},\ 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{public const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0), [1], 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$. By Definition 3.2.13 we have $l_1 \cong_\psi \widehat{l}_1$.

Given $\sigma_3(l_1) = (\omega_1, \text{public } bty, n_1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n_1))$, $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, and $l_1 \cong_\psi \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_3(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}_1, \text{PermL}(\text{Freeable},\ bty,\ \text{public}, \widehat{n}_1))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n_1 = \widehat{n}_1$.

Given $\text{DecodeVal}(\text{public } bty, n_1, \omega_1) = [m_0, ..., m_{n_1}]$, $\text{public } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}_1, \widehat{\omega}_1) = [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}]$ and $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}]$.

Given $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$, $x = \widehat{x}$, $n = \widehat{n}$, and $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, by Lemma 3.2.29 we have $\text{OutputArray}(\widehat{x},\ \widehat{n}, [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}])$ such that the corresponding output files are congruent.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcoutput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$, $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \Downarrow'_{d_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_3(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)],\ [1],\ 1]$, $\widehat{\sigma}_3(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n}_2, \text{PermL}(\text{Freeable},\ \widehat{bty}, \text{public}, \widehat{n}_1))$, $\text{DecodeVal}(\widehat{bty},\ \widehat{n}_1, \widehat{\omega}_1) = [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}]$, and $\text{OutputArray}(\widehat{x}, \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)) \Downarrow'_{out1} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ by Vanilla C rule Output 1D Array.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out1} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,$

$\text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)) \Downarrow'_{out1} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $out1 \cong out1$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out4} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out4} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by Location-tracking SMC^2 rule SMC Output Private 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n)$, $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, e_3) \Downarrow^t_{c_3} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, n_1)$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_3(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_3(l_1) = (\omega_1, \text{private } bty, n_1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n_1))$, $\text{DecodeVal}(\text{private } bty, n_1, \omega_1) = [m_0, ..., m_{n_1}]$, and $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$ and ψ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3))$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $\text{smcoutput}(e_1, e_2, e_3) \cong_\psi \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out4} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{smcoutput}(e_1, e_2, e_3)$. Therefore, by Lemma 3.2.3 we have $\text{smcoutput}(e_1, e_2, e_3) \cong \text{mcoutput}(\widehat{e}_1, \widehat{e}_2, \widehat{e}_3)$. By Definition 3.2.10 we have $\text{Erase}(\text{smcoutput}(e_1, e_2, e_3)) = \text{mcoutput}(\text{Erase}(e_1, e_2, e_3))$. By Definition 3.2.8, we have $\text{Erase}(e_1, e_2, e_3) = \text{Erase}(e_1), \text{Erase}(e_2), \text{Erase}(e_3)$. By Definition 3.2.10 we have $\text{Erase}(e_1) = \widehat{e}_1$, $\text{Erase}(e_2) = \widehat{e}_2$, and $\text{Erase}(e_3) = \widehat{e}_3$. Therefore, we have $e_1 \cong \widehat{e}_1$, $e_2 \cong \widehat{e}_2$, and $e_3 \cong \widehat{e}_3$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1)$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{x})$ and ψ_1 such that $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{x})$ and $c_1 \cong d_1$. Given $x \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2)$. Given $(\gamma\, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n})$ and ψ_2 such that $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{n})$ and $c_2 \cong d_2$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $e_3 \cong \widehat{e}_3$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \cong_\psi (\gamma,\ \sigma_2,\ \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ e_3)$. Given $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ e_3) \Downarrow^t_{c_3} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ n_1)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \Downarrow'_{d_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$ and ψ_3 such that $(\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ n_1) \cong_{\psi_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$ and $c_3 \cong d_3$. Given $n_1 \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_3 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$ and $n_1 \cong_\psi \widehat{n}_1$. Given $\text{Label}(e_3, \gamma) = \text{public}$, we have $\text{Label}(n_1, \gamma) = \text{public}$ and therefore $n_1 = \widehat{n}_1$ by Definition 3.2.18 and Definition 3.2.10.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_3(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_3(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1,$
$\text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$. By Definition 3.2.13 we have $l_1 \cong_\psi \widehat{l}_1$.

Given $\sigma_3(l_1) = (\omega_1, \text{private } bty, n_1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n_1))$, $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, and $l_1 \cong_\psi \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_3(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}_1, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}_1))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n_1 = \widehat{n}_1$.

Given $\text{DecodeVal}(\text{private } bty, n_1, \omega_1) = [m_0,\ ...,\ m_{n_1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}_1, \widehat{\omega}_1) = [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}]$ and $[m_0,\ ...,\ m_{n_1}] \cong [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}]$.

Given $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$, $x = \widehat{x}$, $n = \widehat{n}$, and $[m_0, ..., m_{n_1}] \cong [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, by Lemma 3.2.29 we have $\text{OutputArray}(\widehat{x},\ \widehat{n}, [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}])$ such that the corresponding output files are congruent.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcoutput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{n})$, $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{e}_3) \Downarrow'_{d_3} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \widehat{n}_1)$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_3(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},$ $\text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_3(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}_2, \text{PermL}(\text{Freeable},$ $\widehat{bty}, \text{public}, \widehat{n}_1))$, $\text{DecodeVal}(\widehat{bty},\ \widehat{n}_1,\ \widehat{\omega}_1) = [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}]$, and $\text{OutputArray}(\widehat{x},\ \widehat{n},\ [\widehat{m}_0,\ ...,\ \widehat{m}_{\widehat{n}_1}])$, we have

$\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcoutput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3)) \Downarrow'_{out1} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ by Vanilla C rule Output 1D Array.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out4} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{mcoutput}(\widehat{e}_1,\ \widehat{e}_2,\ \widehat{e}_3)) \Downarrow'_{out1} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $out4 \cong out1$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})) \Downarrow^t_{df} (\gamma_1, \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})) \Downarrow^t_{df} (\gamma_1, \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Function Declaration, we have $(\text{acc} = 0) \wedge (\text{bid} = none)$, $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, and $\sigma_1 = \sigma[l \rightarrow (\text{NULL},\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun}(\text{public}))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}}))$ and ψ such that $(\gamma,\ \sigma,\ \text{acc},\ ty\ x(\overline{p})) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}}))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x(\overline{p}) \cong_\psi \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})) \Downarrow^t_{df} (\gamma_1, \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin ty\ x(\overline{p})$. Therefore, by Lemma 3.2.3 we have $ty\ x(\overline{p}) \cong \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})$. By Definition 3.2.10 we have $\text{Erase}(ty\ x(\overline{p})) = \text{Erase}(ty)\ \widehat{x}(\text{Erase}(\overline{p}))$ where $x = \widehat{x}$, $\text{Erase}(ty) = \widehat{ty}$, and $\text{Erase}(\overline{p}) = \widehat{\overline{p}}$ by Definition 3.2.9. Therefore, we have $ty \cong \widehat{ty}$ and $\overline{p} \cong \widehat{\overline{p}}$.

Given $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$ and $\overline{p} \cong \widehat{\overline{p}}$, by Lemma 3.2.30 we have $\text{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}}$ where $\overline{ty} \cong \widehat{\overline{ty}}$. Therefore, we have $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$ by Definition 3.2.6.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, $x = \widehat{x}$, $l = \widehat{l}$, $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$, and $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{\overline{ty}} \rightarrow \widehat{ty})]$ such that $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given $\sigma_1 = \sigma[l \rightarrow (\text{NULL}, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun}(\text{public}))]$, $l = \widehat{l}$, $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$, and $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\text{NULL},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \text{PermL_Fun}(\text{public}))]$ such that $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}}))$, $\widehat{l} = \phi()$, $\text{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}}$, $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{\overline{ty}} \rightarrow \widehat{ty})]$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\text{NULL},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \text{PermL_Fun}(\text{public}))]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})) \Downarrow'_{df} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$ by Vanilla C rule Function Declaration.

Given $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, by Definition 4.2.1 we have $(\gamma_1, \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})) \Downarrow^t_{df} (\gamma_1, \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})) \Downarrow'_{df}$ $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $df \cong df$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})\{s\}) \Downarrow^t_{fpd} (\gamma,\ \sigma_2,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})\{s\}) \Downarrow^t_{fpd} (\gamma,\ \sigma_2,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Pre-Declared Function Definition, we have $(\text{acc} = 0) \wedge (\text{bid} = none)$, $x \in \gamma$, $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $\text{CheckPublicEffects}(s, x, \gamma,\ \sigma) = n$, $\text{EncodeFun}(s, n, \overline{p}) = \omega$, $\sigma = \sigma_1[l \rightarrow (\text{NULL},\ \overline{ty} \rightarrow ty,\ 1, \text{PermL_Fun}(\text{public}))]$, and $\sigma_2 = \sigma_1[l \rightarrow (\omega,\ \overline{ty} \rightarrow ty,\ 1, \text{PermL_Fun}(\text{public}))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})\{s\}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x(\overline{p})\{s\} \cong_\psi \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})\{s\})$ $\Downarrow^t_{fpd} (\gamma,\ \sigma_2,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin ty\ x(\overline{p})\{s\}$. Therefore, by Lemma 3.2.3 we have $ty\ x(\overline{p})\{s\} \cong \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x(\overline{p})\{s\}) = \text{Erase}(ty\ x(\overline{p}))\ \{\text{Erase}(s)\}$, $\text{Erase}(ty\ x(\overline{p})) = \text{Erase}(ty)\ \widehat{x}\ (\text{Erase}(\overline{p}))$ where $x = \widehat{x}$, $\text{Erase}(ty) = \widehat{ty}$, $\text{Erase}(s) = \widehat{s}$, and $\text{Erase}(\overline{p}) = \widehat{\overline{p}}$ by Definition 3.2.9. Therefore, we have $ty \cong \widehat{ty}$ and $\overline{p} \cong \widehat{\overline{p}}$.

Given $x \in \gamma$, $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{\overline{ty}} \rightarrow \widehat{ty})$ such that $l = \widehat{l}$ by $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$ by Lemma 3.2.14 and $\widehat{x} \in \widehat{\gamma}$ by Lemma 3.2.33.

Given $\text{EncodeFun}(s, n, \overline{p}) = \omega$, $s \cong \widehat{s}$, and $\overline{p} \cong \widehat{\overline{p}}$, by Lemma 3.2.46 we have $\text{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}}) = \widehat{\omega}$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma = \sigma_1[l \rightarrow (\text{NULL},\ \overline{ty} \rightarrow ty,\ 1, \text{PermL_Fun}(\text{public}))]$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.36 we have $\widehat{\sigma} = \widehat{\sigma}_1[\widehat{l} \rightarrow (\text{NULL},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1, \text{PermL_Fun}(\text{public}))]$ and $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given $\sigma_2 = \sigma_1[l \rightarrow (\omega,\ \overline{ty} \rightarrow ty,\ 1, \text{PermL_Fun}(\text{public}))]$, $(\gamma,\ \sigma_1) \cong (\widehat{\gamma},\ \widehat{\sigma}_1)$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{\overline{ty}} \rightarrow \widehat{ty}, 1, \text{PermL_Fun}(\text{public}))]$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\})$, $x \in \widehat{\gamma}$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{\overline{ty}} \rightarrow \widehat{ty})$, $\widehat{\sigma} = \widehat{\sigma}_1[\widehat{l} \rightarrow (\text{NULL},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \text{PermL_Fun}(\text{public}))]$,

$\text{EncodeFun}(\widehat{s},\ \square,\ \widehat{\overline{p}}) = \widehat{\omega}$, and $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \text{PermL_Fun(public)})]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\})$ $\Downarrow'_{fpd}$ $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ by Vanilla C rule Pre-Declared Function Definition.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})\{s\}) \Downarrow^t_{fpd} (\gamma,\ \sigma_2,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}) \Downarrow'_{fpd}$ $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $fpd \cong fpd$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})\{s\}) \Downarrow^t_{fd} (\gamma_1, \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})\{s\}) \Downarrow^t_{fd} (\gamma_1, \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Function Definition, we have $l = \phi()$, $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$, $x \notin \gamma$, $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, $\text{CheckPublicEffects}(s, x, \gamma, \sigma) = n$, $\text{EncodeFun}(s, n, \overline{p}) = \omega$, and $\sigma_1 = \sigma[l \rightarrow (\omega, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\})$ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})\{s\}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x(\overline{p})\{s\} \cong_\psi \widehat{ty}\ \widehat{x}(\widehat{\overline{e}})\ \{\widehat{s}\}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})\{s\}) \Downarrow^t_{fd}$ $(\gamma_1, \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin ty\ x(\overline{p})\{s\}$. Therefore, by Lemma 3.2.3 we have $ty\ x(\overline{p})\{s\} \cong \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x(\overline{p})\{s\}) = \text{Erase}(ty\ x(\overline{p}))\ \{\text{Erase}(s)\}$, $\text{Erase}(ty\ x(\overline{p})) = \text{Erase}(ty)\ \widehat{x}\ (\text{Erase}(\overline{p}))$ where $x = \widehat{x}$, $\text{Erase}(ty) = \widehat{ty}$, $\text{Erase}(\overline{p}) = \widehat{\overline{p}}$, and $\text{Erase}(s) = \widehat{s}$ by Definition 3.2.9. Therefore, we have $ty \cong \widehat{ty}$, $\overline{p} \cong \widehat{\overline{p}}$, and $s \cong \widehat{s}$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$ and $\overline{p} \cong \widehat{\overline{p}}$, by Lemma 3.2.30 we have $\text{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}}$ where $\overline{ty} \cong \widehat{\overline{ty}}$.

Given $x \notin \gamma$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, by Lemma 3.2.32 we have $\widehat{x} \notin \widehat{\gamma}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, $x = \widehat{x}$, $l = \widehat{l}$, $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$, and $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{\overline{ty}} \rightarrow \widehat{ty})]$ such that $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given $\text{EncodeFun}(s, n, \overline{p}) = \omega$, $s \cong \widehat{s}$, and $\overline{p} \cong \widehat{\overline{p}}$, by Lemma 3.2.46 we have $\text{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}}) = \widehat{\omega}$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_1 = \sigma[l \to (\text{NULL}, \overline{ty} \to ty, 1, \text{PermL_Fun}(\text{public}))]$, $l = \widehat{l}$, $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$, and $\overline{ty} \to ty \cong \widehat{\overline{ty}} \to \widehat{ty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \to (\text{NULL},\ \widehat{\overline{ty}} \to \widehat{ty},\ 1, \text{PermL_Fun}(\text{public}))]$ such that $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\})$, $x \notin \widehat{\gamma}$, $\widehat{l} = \phi()$, $\text{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}}$, $\widehat{\gamma}_1 = \widehat{\gamma}[x \to (\widehat{l},\ \widehat{\overline{ty}} \to \widehat{ty})]$, $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \to (\widehat{\omega},\ \widehat{\overline{ty}} \to \widehat{ty},\ 1, \text{PermL_Fun}(\text{public}))]$, and $\text{EncodeFun}(\widehat{s},\ \square,\ \widehat{\overline{p}}) = \widehat{\omega}$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}) \Downarrow'_{fd} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$ by Vanilla C rule Function Definition.

Given $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, by Definition 4.2.1 we have $(\gamma_1, \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x(\overline{p})\{s\}) \Downarrow^t_{fd} (\gamma_1, \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\ \{\widehat{s}\}) \Downarrow'_{fd} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $fd \cong fd$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc}, x(\overline{e})) \Downarrow^t_{fc} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{NULL})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x(\overline{e})) \Downarrow^t_{fc} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{NULL})$ by Location-tracking SMC^2 rule Function Call No Return With Public Side Effects, we have $\gamma(x) = (l, \overline{ty} \to ty)$, $\sigma(l) = (\omega, \overline{ty} \to ty, 1, \text{PermL_Fun}(\text{public}))$, $\text{DecodeFun}(\omega) = (s, 1, \overline{p})$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, $\text{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1) \Downarrow^t_{c_1} (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$, and $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ s) \Downarrow^t_{c_2} (\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}(\widehat{\overline{e}}))$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x(\overline{e})) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}(\widehat{\overline{e}}))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x(\overline{e}) \cong_\psi \widehat{x}(\widehat{\overline{e}})$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x(\overline{e})) \Downarrow^t_{fc} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{NULL})$, by Lemma 4.2.2 we have $(l, \mu) \notin x(\overline{e})$. Therefore, by Lemma 3.2.3 we have $x(\overline{e}) \cong \widehat{x}(\widehat{\overline{e}})$. By Definition 3.2.10 we have $\text{Erase}(x(e)) = \widehat{x}(\text{Erase}(\overline{e}))$ where $x = \widehat{x}$ and $\text{Erase}(\overline{e}) = \widehat{\overline{e}}$ by Definition 3.2.8. Therefore, we have $\overline{e} \cong \widehat{\overline{e}}$.

Given $\gamma(x) = (l, \overline{ty} \to ty)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{\overline{ty}} \to \widehat{ty})$ such that $l = \widehat{l}$ by $\overline{ty} \to ty \cong \widehat{\overline{ty}} \to \widehat{ty}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \overline{ty} \to ty, 1, \text{PermL_Fun}(\text{public}))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{\overline{ty}} \to \widehat{ty},\ 1,\ \text{PermL_Fun}(\text{public}))$ where $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodeFun}(\omega) = (s, 1, \overline{p})$ and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.47 we have $\text{DecodeFun}(\widehat{\omega}) = (\widehat{s},\ \square,\ \widehat{\overline{p}})$ such that

$s \cong \widehat{s}$ and $\overline{p} \cong \widehat{\overline{p}}$.

Given GetFunParamAssign($\overline{p}$, $\overline{e}$) $= s_1$, $\overline{p} \cong \widehat{\overline{p}}$, and $\overline{e} \cong \widehat{\overline{e}}$, by Lemma 3.2.31 we have GetFunParamAssign($\widehat{\overline{p}}$, $\widehat{\overline{e}}$) $= \widehat{s}_1$ where $s_1 \cong_{\psi} \widehat{s}_1$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $s_1 \cong_{\psi} \widehat{s}_1$, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{s}_1)$ by Definition 4.2.1. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1) \Downarrow^{t}_{c_1} (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{s}_1) \Downarrow'_{d_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \Box,\ \text{skip})$ and ψ_1 such that $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \Box,\ \text{skip})$ and $c_1 \cong d_1$. By Definition 4.2.1, we have $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$. By Lemma 4.2.12, we have $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$ and $s \cong \widehat{s}$, BY Lemma 4.2.3 we have $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ s) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \Box,\ \widehat{s})$ by Definition 4.2.1. Given $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ s) \Downarrow^{t}_{c_2} (\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ \text{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \Box,\ \widehat{s}) \Downarrow'_{d_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2, \Box, \text{skip})$ and ψ_2 such that $(\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2, \Box, \text{skip})$ and $c_2 \cong d_2$. By Definition 4.2.1, we have $(\gamma_2, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2)$. By Lemma 4.2.12, we have $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{x}(\widehat{\overline{e}}))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{\overline{ty}} \rightarrow \widehat{ty})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \text{PermL_Fun(public)})$, DecodeFun($\widehat{\omega}$) $= (\widehat{s},\ \Box,\ \widehat{\overline{p}})$, GetFunParamAssign($\widehat{\overline{p}}, \widehat{\overline{e}}$) $= \widehat{s}_1$, $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{s}_1) \Downarrow'_{d_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \Box, \text{skip})$, and $(\widehat{\gamma}_1,\ \widehat{\sigma}_1, \Box, \widehat{s}) \Downarrow'_{d_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \Box,$ skip), we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \Box, \widehat{x}(\widehat{\overline{e}})) \Downarrow'_{fc} (\widehat{\gamma}, \widehat{\sigma}_2, \Box, \text{NULL})$ by Vanilla C rule Function Call.

Given $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $\text{NULL} = \text{NULL}$, by Definition 3.2.17 we have $\text{NULL} \cong_{\psi_2} \text{NULL}$ and by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{NULL}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box, \text{NULL})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x(\overline{e})) \Downarrow^{t}_{fc} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{NULL}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma},\ \Box, \widehat{x}(\widehat{\overline{e}})) \Downarrow'_{fc} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ \text{NULL})$, $\Pi \cong_{\psi_2} \Sigma$, and $fc \cong fc$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x(\overline{e})) \Downarrow^{t}_{fc1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{NULL})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x(\overline{e})) \Downarrow^{t}_{fc1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{NULL})$ by Location-tracking SMC^2 rule Function Call No Return Without Public Side Effects, we have $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $\sigma(l) = (\omega, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})$, DecodeFun($\omega$) $= (s, 1, \overline{p})$, GetFunParamAssign($\overline{p}, \overline{e}$) $= s_1$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s_1) \Downarrow^{t}_{c_1} (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$, and $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ s) \Downarrow^{t}_{c_2} (\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \text{bid},\ \text{acc},\ \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}(\widehat{\overline{e}}))$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ x(\overline{e})) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}(\widehat{\overline{e}}))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $x(\overline{e}) \cong_{\psi} \widehat{x}(\widehat{\overline{e}})$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ x(\overline{e})) \Downarrow^{t}_{fc1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ \mathrm{NULL})$, by Lemma 4.2.2 we have $(l, \mu) \notin x(\overline{e})$. Therefore, by Lemma 3.2.3 we have $x(\overline{e}) \cong \widehat{x}(\widehat{\overline{e}})$. By Definition 3.2.10 we have $\mathrm{Erase}(x(e)) = \widehat{x}(\mathrm{Erase}(\overline{e}))$ where $x = \widehat{x}$ and $\mathrm{Erase}(\overline{e}) = \widehat{\overline{e}}$ by Definition 3.2.8. Therefore, we have $\overline{e} \cong \widehat{\overline{e}}$.

Given $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{\overline{ty}} \rightarrow \widehat{ty})$ such that $l = \widehat{l}$ by $\overline{ty} \rightarrow ty \cong \widehat{\overline{ty}} \rightarrow \widehat{ty}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \overline{ty} \rightarrow ty, 1, \mathrm{PermL_Fun(public)})$, $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \mathrm{PermL_Fun(public)})$ where $\omega \cong_{\psi} \widehat{\omega}$.

Given $\mathrm{DecodeFun}(\omega) = (s, 1, \overline{p})$ and $\omega \cong_{\psi} \widehat{\omega}$, by Lemma 3.2.47 we have $\mathrm{DecodeFun}(\widehat{\omega}) = (\widehat{s},\ \square,\ \widehat{\overline{p}})$ such that $s \cong \widehat{s}$ and $\overline{p} \cong \widehat{\overline{p}}$.

Given $\mathrm{GetFunParamAssign}(\overline{p},\ \overline{e}) = s_1$, $\overline{p} \cong \widehat{\overline{p}}$, and $\overline{e} \cong \widehat{\overline{e}}$, by Lemma 3.2.31 we have $\mathrm{GetFunParamAssign}(\widehat{\overline{p}}, \widehat{\overline{e}}) = \widehat{s}_1$ where $s_1 \cong_{\psi} \widehat{s}_1$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $s_1 \cong_{\psi} \widehat{s}_1$, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ s_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1)$ by Definition 4.2.1. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ s_1) \Downarrow^{t}_{c_1} (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \mathrm{bid},\ \mathrm{acc},\ \mathrm{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1) \Downarrow'_{d_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \mathrm{skip})$ and ψ_1 such that $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \mathrm{bid},\ \mathrm{acc},\ \mathrm{skip}) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \mathrm{skip})$ and $c_1 \cong d_1$. By Definition 4.2.1, we have $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$. By Lemma 4.2.12, we have $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$ and $s \cong \widehat{s}$, by Lemma 4.2.3 we have $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \mathrm{bid},\ \mathrm{acc},\ s) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{s})$ by Definition 4.2.1. Given $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \mathrm{bid},\ \mathrm{acc},\ s) \Downarrow^{t}_{c_2} (\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \mathrm{bid},\ \mathrm{acc},\ \mathrm{skip})$, by the inductive hypothesis, we have $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{s}) \Downarrow'_{d_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square,\ \mathrm{skip})$ and ψ_2 such that $(\gamma_2, \sigma_2,\ \Delta_2,\ \overline{\chi}_2,\ \mathrm{bid},\ \mathrm{acc},\ \mathrm{skip}) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2, \square, \mathrm{skip})$ and $c_2 \cong d_2$. By Definition 4.2.1, we have $(\gamma_2, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2)$. By Lemma 4.2.12, we have $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}(\widehat{\overline{e}}))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{\overline{ty}} \rightarrow \widehat{ty})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{\overline{ty}} \rightarrow \widehat{ty},\ 1,\ \mathrm{PermL_Fun(public)})$, $\mathrm{DecodeFun}(\widehat{\omega}) = (\widehat{s},\ \square,\ \widehat{\overline{p}})$, $\mathrm{GetFunParamAssign}(\widehat{\overline{p}}, \widehat{\overline{e}}) = \widehat{s}_1$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{s}_1) \Downarrow'_{d_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square, \mathrm{skip})$, and $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{s}) \Downarrow'_{d_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2,\ \square, \mathrm{skip})$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}(\widehat{\overline{e}})) \Downarrow'_{fc} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \mathrm{NULL})$ by Vanilla C rule Function Call.

Given $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $\mathrm{NULL} \cong \mathrm{NULL}$, by Lemma 4.2.3 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ \mathrm{NULL}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \mathrm{NULL})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ x(\overline{e})) \Downarrow^{t}_{fc1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ \mathrm{NULL}) \cong_{\psi_2}$

$(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}(\widehat{e}))\Downarrow'_{fc}(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{NULL})$, $\Pi \cong_{\psi_2} \Sigma$, and $fc1 \cong fc$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{malloc}(e)) \Downarrow^t_{mal} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l,0))$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{malloc}(e)) \Downarrow^t_{mal} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l,0))$ by Location-tracking SMC^2 rule Public Malloc, we have $\text{Label}(e,\gamma) = \text{public}$, $(\text{acc} = 0) \wedge (\text{bid} = \mathit{none})$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, $l = \phi()$, and $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, n)\big]\big)\big]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{malloc}(\widehat{e}))$ and ψ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{malloc}(e)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{malloc}(\widehat{e}))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{malloc}(e) \cong_\psi \text{malloc}(\widehat{e})$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{malloc}(e)) \Downarrow^t_{mal} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l,0))$, by Lemma 4.2.2 we have $(l,\mu) \notin \text{malloc}(e)$. Therefore, by Lemma 3.2.3 we have $\text{malloc}(e) \cong \text{malloc}(\widehat{e})$. By Definition 3.2.10 we have $\text{Erase}(\text{malloc}(e)) = \text{malloc}(\text{Erase}(e))$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and $c_1 \cong d_1$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e,\gamma) = \text{public}$, we have $\text{Label}(n,\gamma) = \text{public}$, and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$, $l = \widehat{l}$, and $(l,0) \cong_\psi (\widehat{l},0)$.

Given $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, n)\big)\big]$, $n = \widehat{n}$, $l = \widehat{l}$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, by Lemma 3.2.7 we have $\widehat{\sigma}_2 = \widehat{\sigma}_1\big[\widehat{l} \rightarrow \big(\text{NULL}, \text{void}*, \hat{n}, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, \widehat{n})\big)\big]$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{malloc}(\widehat{e}))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \hat{n})$, $\widehat{l} = \phi()$, and $\widehat{\sigma}_2 = \widehat{\sigma}_1\big[\widehat{l} \rightarrow \big(\text{NULL}, \text{void}*, \hat{n}, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, \widehat{n})\big)\big]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{malloc}(\widehat{e})) \Downarrow'_{mal} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ (\widehat{l},0))$ by Vanilla C rule Malloc.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $(l,0) \cong_\psi (\widehat{l},0)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l,0)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ (\widehat{l},0))$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{malloc}(e)) \Downarrow^t_{mal} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l,0)) \cong$

$(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{malloc}(\widehat{e})) \Downarrow'_{mal} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ (\widehat{l}, 0))$, $\Pi \cong_\psi \Sigma$, and $mal \cong mal$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{pmalloc}(e,\ ty)) \Downarrow^t_{malp} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ (l, 0))$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{pmalloc}(e,\ ty)) \Downarrow^t_{malp} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ (l, 0))$ by Location-tracking SMC^2 rule Private Malloc, we have $(ty = \text{private int}) \vee (ty = \text{private float})$, $\text{Label}(e, \gamma) = \text{public}$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n)$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, $l = \phi()$, and $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL},\ \text{void}*, n, \text{PermL}(\text{Freeable}, ty, \text{private}, n)\big]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{malloc}(\widehat{e}'))$ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{pmalloc}(e, ty)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{malloc}(\widehat{e}'))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{pmalloc}(e, ty) \cong_\psi \text{malloc}(\widehat{e}')$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{pmalloc}(e,\ ty)) \Downarrow^t_{malp} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ (l, 0))$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{pmalloc}(e, ty)$. Therefore, by Lemma 3.2.3 we have $\text{pmalloc}(e,\ ty) \cong \text{malloc}(\widehat{e}')$. By Lemma 3.2.6, we have that $\widehat{e}' = \text{sizeof}(\text{Erase}(ty)) \cdot \text{Erase}(e)$. By Definition 3.2.6, we have $\text{Erase}(ty) = \widehat{ty}$ such that $ty \cong \widehat{ty}$. By Definition 3.2.10, we have $\text{Erase}(e) = \widehat{e}$ such that $e \cong \widehat{e}$. Therefore, we have $\widehat{e}' = \text{sizeof}(\widehat{ty}) \cdot (\widehat{e})$.

Given $\widehat{e}' = \text{sizeof}(\widehat{ty}) \cdot (\widehat{e})$, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{sizeof}(\widehat{ty}) \cdot (\widehat{e}))$.

Given $\text{sizeof}(\widehat{ty})$, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{sizeof}(\widehat{ty})) \Downarrow'_{ty} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{n}_1)$ by Vanilla C rule Size of Type such that $\widehat{n}_1 = \tau(\widehat{ty})$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\widehat{e} \cong e$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and $c_1 \cong d_1$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$, and therefore by Definition 3.2.18 and Definition 3.2.10 we have $n = \widehat{n}$.

Given $\widehat{n}_1$ and $\widehat{n}$, we have $\widehat{n}' = \widehat{n}_1 \cdot \widehat{n}$. Therefore, by Vanilla C rule Multiplication we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{sizeof}(\widehat{ty}) \cdot (\widehat{e}))$ $\Downarrow'_{bm} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}')$.

Given $\widehat{e}' = \text{sizeof}(\widehat{ty}) \cdot (\widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{sizeof}(\widehat{ty})) \Downarrow'_{ty} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{n}_1)$, $\widehat{n}_1 = \tau(\widehat{ty})$, and $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1}$

$(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$, we have $\widehat{e}' = \tau(\widehat{ty}) \cdot \widehat{n} = \widehat{n}'$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$, $l = \widehat{l}$, and $(l, 0) \cong (\widehat{l}, 0)$.

Given $\sigma_2 = \sigma_1\big[l \to \big(\text{NULL},\ \text{void}*,\ n \cdot \tau(ty), \text{PermL}(\text{Freeable}, \text{void}*, \text{private}, n \cdot \tau(ty))\big]$, $n = \widehat{n}$, $\tau(\widehat{ty}) \cdot \widehat{n} = \widehat{n}'$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, $l = \widehat{l}$, $ty \cong \widehat{ty}$, and $\frac{n \cdot \tau(ty)}{\tau(ty)} = \frac{\widehat{n}'}{\tau(\widehat{ty})}$, by Lemma 3.2.8 we have $\widehat{\sigma}_2 = \widehat{\sigma}_1\big[\widehat{l} \to \big(\text{NULL},\ \text{void}*,\ \widehat{n}',$ $\text{PermL}(\text{Freeable}, \text{void}, \text{public}, \widehat{n}')\big)\big]$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{malloc}(\widehat{e}'))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}') \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n}')$, $\widehat{l} = \phi()$, $\text{EncodePtr}(\text{void}*, [1,\ [(\widehat{l}_{default},$ $0)], [1], 1]) = \widehat{\omega}$, and $\widehat{\sigma}_2 = \widehat{\sigma}_1\big[\widehat{l} \to \big(\text{NULL},\ \text{void}*,\ \widehat{n}',\ \text{PermL}(\text{Freeable}, \text{void},\ \text{public},\ \widehat{n}')\big)\big]$, we have $\Sigma \triangleright$ $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{malloc}(\widehat{e})) \Downarrow'_{mal} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ (\widehat{l}, 0))$ by Vanilla C rule Malloc.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0)) \cong_\psi$ $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ (\widehat{l}, 0))$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{pmalloc}(e,\ ty)) \Downarrow^t_{malp} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l, 0))$ $\cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{malloc}(\widehat{e})) \Downarrow'_{mal} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ (\widehat{l}, 0))$, $\Pi \cong_\psi \Sigma$, and $[malp, d_1] \cong [mal, bm, ty, d_1]$ by Definition 3.2.22.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{free}(e)) \Downarrow^t_{fre} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{free}(e)) \Downarrow^t_{fre} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Public Free, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x)$, $\gamma(x) = (l, \text{public}\ bty*)$, $(bty = \text{int}) \vee (bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{void})$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, and $\text{Free}(\sigma_1, l, \gamma) = \sigma_2$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{free}(\widehat{e}))$ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{free}(e)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{free}(\widehat{e}))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{free}(e) \cong_\psi \text{free}(\widehat{e})$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{free}(e)) \Downarrow^t_{fre} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{free}(e)$. Therefore, by Lemma 3.2.3 we have $\text{free}(e) \cong \text{free}(\widehat{e})$. By Definition 3.2.10 we have $\text{Erase}(\text{free}(e)) = \text{free}(\text{Erase}(e))$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,$ $\overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and $c_1 \cong d_1$. Given $x \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{public } bty*)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, and $\text{public } bty* \cong \widehat{bty}*$ by Lemma 3.2.14. By Definition 3.2.13 we have $l \cong_\psi \widehat{l}$.

Given $\text{Free}(\sigma_1, l, \gamma) = \sigma_2$ and $l \cong_\psi \widehat{l}$, by Lemma 3.2.38 we have $\text{Free}(\widehat{\sigma}_1,\ \widehat{l}, \widehat{\gamma}) = \widehat{\sigma}_2$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{free}(\widehat{e}))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*)$, and $\text{Free}(\widehat{\sigma}_1,\ \widehat{l}, \widehat{\gamma}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma}, \square, \text{free}(\widehat{e})) \Downarrow'_{fre} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ by Vanilla C rule Free.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{free}(e)) \Downarrow^t_{fre} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{free}(\widehat{e})) \Downarrow'_{fre}$ $(\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $fre \cong fre$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{pfree}(e)) \Downarrow^t_{frep} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{pfree}(e)) \Downarrow^t_{frep} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private Free, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x)$, $\gamma(x) = (l,\ \text{private } bty*)$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, $(bty = \text{int}) \vee (bty = \text{float})$, and $\text{PFree}(\sigma_1,\ l) = \sigma_2$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{free}(\widehat{e}))$ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{pfree}(e)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{free}(\widehat{e}))$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\text{pfree}(e) \cong_\psi \text{free}(\widehat{e})$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{pfree}(e)) \Downarrow^t_{frep} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin \text{pfree}(e)$. Therefore, by Lemma 3.2.3 we have $\text{pfree}(e) \cong \text{free}(\widehat{e})$. By Definition 3.2.10 we have $\text{Erase}(\text{pfree}(e)) = \text{free}(\text{Erase}(e))$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$ and $c_1 \cong d_1$. Given $x \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $x = \widehat{x}$.

Given $\gamma(x) = (l,\ \text{private } bty*)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, and $\text{private } bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\text{PFree}(\gamma,\ \sigma_1, l) = (\sigma_2, \bar{l}, \bar{j})$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, $l \cong_\psi \widehat{l}$, $\text{GetLocationSwap}(\bar{l}, \bar{j}) = \bar{l}'$, $\psi_2 = \psi[\bar{l}']$, and $\text{SwapMemory}(\widehat{\sigma}_2, \psi_2) = \widehat{\sigma}_3$, by Lemma 3.2.39 we have $\text{Free}(\widehat{\sigma}_1, \widehat{l}, \widehat{\gamma}) = \widehat{\sigma}_2$, such that $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{free}(\widehat{e}))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*})$, and $\text{Free}(\widehat{\sigma}_1,\ \widehat{l}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, \text{free}(\widehat{e})) \Downarrow'_{fre} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ by Vanilla C rule Free.

Given $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{pfree}(e)) \Downarrow^t_{frep} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \text{free}(\widehat{e}))$ $\Downarrow'_{fre} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_{\psi_2} \Sigma$, and $frep \cong fre$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x = e) \Downarrow^t_{ds} (\gamma_1, \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x = e) \Downarrow^t_{ds} (\gamma_1, \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Declaration Assignment, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x) \Downarrow^t_{c_1} (\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$, and $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{c_2} (\gamma_1, \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x = e \cong_\psi \widehat{ty}\ \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x = e) \Downarrow^t_{ds}$ $(\gamma_1, \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin ty\ x = e$. Therefore, by Lemma 3.2.3 we have $ty\ x = e \cong \widehat{ty}\ \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x = e) = \text{Erase}(ty)\ \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(ty) = \widehat{ty}$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $ty \cong \widehat{ty}$ and $x = \widehat{x}$, by Definition 3.2.10 we have $ty\ x \cong \widehat{ty}\ \widehat{x}$. Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x)$ by Definition 4.2.1. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x) \Downarrow^t_{c_1} (\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ x) \Downarrow'_{d_1}$ $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$ and ψ_1 such that $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$. Given $\text{pfree}(e) \notin ty\ x$, we have $\psi_1 = \psi$ by Definition 3.2.11. By Definition 4.2.1 we have $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $x = \widehat{x}$ and $e \cong \widehat{e}$, by Definition 3.2.10 we have $x = e \cong \widehat{x} = \widehat{e}$. Given $(\gamma_1,\ \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, by Lemma 4.2.3 we have $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{x} = \widehat{e})$ such that $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \widehat{x} = \widehat{e}) \cong_\psi (\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ x = e)$. Given $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{c_2} (\gamma_1, \sigma_2,\ \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$, by the inductive hypothesis we have

$(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ x = \widehat{e}) \Downarrow'_{d_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ and ψ_2 such that $(\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc},\ \text{skip}) \cong_{\psi_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Given $\text{pfree}(e) \notin x = e$, we have $\psi_2 = \psi$ by Definition 3.2.11. By Definition 4.2.1, we have $(\gamma_1, \sigma_2) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ x) \Downarrow'_{d_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$, and $(\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ x = \widehat{e}) \Downarrow'_{d_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x} = \widehat{e}) \Downarrow'_{ds} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ by Vanilla C rule Declaration Assignment.

Given $(\gamma_1,\ \sigma_2) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x = e) \Downarrow^t_{ds} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc},\ \text{skip}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x} = \widehat{e})$ $\Downarrow'_{ds} (\widehat{\gamma}_1,\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_{\psi} \Sigma$, and $ds \cong ds$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, ty\ x[e_1]\ = e_2) \Downarrow^t_{das} (\gamma_1, \sigma_2, \Delta_2,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, ty\ x = e) \Downarrow^t_{ds} (\gamma_1, \sigma_2, \Delta_2,\ \overline{\chi}_1,\ \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ ty\ x) \Downarrow^t_d (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x) \Downarrow^t_d (\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Public Declaration, we have $(ty = \text{public}\ bty) \vee (ty = \text{char})$, $\text{acc} = 0$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\omega = \text{EncodeVal}(ty, \text{NULL})$, and $\sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x})$ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x \cong_{\psi} \widehat{ty}\ \widehat{x}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x) \Downarrow^t_d (\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin ty\ x$. Therefore, by Lemma 3.2.3 we have $ty\ x \cong \widehat{ty}\ \widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x) = \text{Erase}(ty)\ \text{Erase}(x)$, $\text{Erase}(ty) = \widehat{ty}$ and $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$. Therefore, we have $ty \cong \widehat{ty}$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $x = \widehat{x}$, $l = \widehat{l}$, $ty \cong \widehat{ty}$, and $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty})]$ such that $(\gamma_1, \sigma) \cong_{\psi} (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given $\omega = \text{EncodeVal}(ty, \text{NULL})$, and $ty \cong \widehat{ty}$, by Lemma 3.2.40 we have $\widehat{\omega} = \text{EncodeVal}(\widehat{bty}, \text{NULL})$ such that $\omega \cong_{\psi} \widehat{\omega}$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega,\ ty,\ 1,\ \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$, $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and $ty \cong \widehat{ty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{bty},\ 1,\ \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))]$ such that $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{bty}\ \widehat{x})$, $\widehat{l} = \phi()$, $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty})]$, $\widehat{\omega} = \text{EncodeVal}(\widehat{bty}, \text{NULL})$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{bty},\ 1,\ \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{bty}\ \widehat{x}) \Downarrow'_d (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square, \text{skip})$ by Vanilla C rule Declaration.

Given $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, by Definition 4.2.1 we have $(\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square, \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x) \Downarrow^t_d (\gamma_1, \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{bty}\ \widehat{x}) \Downarrow'_d (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $d \cong d$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ ty\ x) \Downarrow^t_{d1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ ty\ x) \Downarrow^t_{d1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private Declaration, we have $((ty = bty) \vee (ty = \text{private}\ bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \text{private}\ bty)]$, $\omega = \text{EncodeVal}(ty, \text{NULL})$, and $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private}\ bty, 1, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, 1))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x})$ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ ty\ x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x \cong_\psi \widehat{ty}\ \widehat{x}$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ ty\ x) \Downarrow^t_{d1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin ty\ x$. Therefore, by Lemma 3.2.3 we have $ty\ x \cong \widehat{ty}\ \widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x) = \text{Erase}(ty)\ \text{Erase}(x)$, $\text{Erase}(ty) = \widehat{ty}$ and $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$. Therefore, we have $ty \cong \widehat{ty}$. Given $ty \cong \widehat{ty}$ and $((ty = \text{private}\ bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, by Definition 3.2.6 we have $bty \cong \widehat{bty}$. Therefore, we have $\text{private}\ bty \cong \widehat{bty}$ by Definition 3.2.6.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{private}\ bty)]$, $x = \widehat{x}$, $l = \widehat{l}$, $\text{private}\ bty$, and $(\gamma,\ \sigma) \cong (\widehat{\gamma},\ \widehat{\sigma})$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty})]$ such that $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given $\omega = \text{EncodeVal}(\text{private}\ bty, \text{NULL})$, and $\text{private}\ bty \cong \widehat{bty}$, by Lemma 3.2.40 we have $\widehat{\omega} = \text{EncodeVal}(\widehat{bty}, \text{NULL})$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))]$, $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma})$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{private } bty \cong \widehat{bty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))]$ such that $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{bty}\ \widehat{x})$, $\widehat{l} = \phi()$, $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \widehat{bty})]$, $\widehat{\omega} = \text{EncodeVal}(\widehat{bty}, \text{NULL})$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{bty}\ \widehat{x}) \Downarrow'_d (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$ by Vanilla C rule Declaration.

Given $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$, by Definition 4.2.1 we have $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{d1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{bty}\ \widehat{x}) \Downarrow'_d (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $d1 \cong d$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{d2} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{d2} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$ by Location-tracking SMC^2 rule Private Declaration (Inside a Private - Conditioned If Else Branch), we have $((ty = bty) \vee (ty = \text{private } bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $(\text{acc} > 0) \wedge ((\text{bid} = then) \vee (\text{bid} = else))$, $l = \phi()$, $\overline{\chi}_1 = l :: \overline{\chi}[\text{acc}]$, $\gamma_1 = \gamma[x \rightarrow (l, \text{private } bty)]$, $\omega = \text{EncodeVal}(ty, \text{NULL})$, and $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $ty\ x \cong_\psi \widehat{ty}\ \widehat{x}$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{d2} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin ty\ x$. Therefore, by Lemma 3.2.3 we have $ty\ x \cong \widehat{ty}\ \widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x) = \text{Erase}(ty)\ \text{Erase}(x)$, $\text{Erase}(ty) = \widehat{ty}$ and $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$. Therefore, we have $ty \cong \widehat{ty}$. Given $ty \cong \widehat{ty}$ and $((ty = \text{private } bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, by Definition 3.2.6 we have $bty \cong \widehat{bty}$. Therefore, we have $\text{private } bty \cong \widehat{bty}$ by Definition 3.2.6.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{private } bty)]$, $x = \widehat{x}$, $l = \widehat{l}$, $\text{private } bty$, and $(\gamma, \sigma) \cong (\widehat{\gamma}, \widehat{\sigma})$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \widehat{bty})]$ such that $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma})$.

Given $\omega = \text{EncodeVal}(\text{private } bty, \text{NULL})$, and $\text{private } bty \cong \widehat{bty}$, by Lemma 3.2.40 we have $\widehat{\omega} = \text{EncodeVal}(\widehat{bty}, \text{NULL})$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))]$, $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma})$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{private } bty \cong \widehat{bty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))]$ such that $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{bty}\ \widehat{x})$, $\widehat{l} = \phi()$, $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \widehat{bty})]$, $\widehat{\omega} = \text{EncodeVal}(\widehat{bty}, \text{NULL})$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{bty}\ \widehat{x}) \Downarrow'_d (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$ by Vanilla C rule Declaration.

Given $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$, by Definition 4.2.1 we have $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{d2} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{bty}\ \widehat{x}) \Downarrow'_d (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $d2 \cong d$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by Location-tracking SMC^2 rule Public Pointer Declaration, we have $(ty = \text{public } bty*) \vee ((ty = bty*) \wedge ((bty = \text{char}) \vee (bty = \text{void})))$, $\text{GetIndirection}(*) = i$, $\text{acc} = 0$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \text{public } bty*)]$, $\omega = \text{EncodePtr}(\text{public } bty*, [1, [(l_{default}, 0)], [1], i])$, and $\sigma_1 = \sigma[l \rightarrow (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $ty\ x \cong_\psi \widehat{ty}\ \widehat{x}$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin ty\ x$. Therefore, by Lemma 3.2.3 we have $ty\ x \cong \widehat{ty}\ \widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x) = \text{Erase}(ty)\ \text{Erase}(x)$, $\text{Erase}(ty) = \widehat{ty}$ and $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$. Therefore, we have $ty \cong \widehat{ty}$ such that $* = *$.

Given $\text{GetIndirection}(*) = i$ and $* = *$, by Lemma 3.2.49 we have $\text{GetIndirection}(*) = \widehat{i}$ such that $i = \widehat{i}$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{public } bty*)]$, $x = \widehat{x}$, $l = \widehat{l}$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $\text{public } bty* \cong \widehat{bty}*$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \widehat{bty}*)]$ such that $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma})$.

Given $\omega = \text{EncodePtr}(\text{public } bty*, [1, [(l_{default}, 0)], [1], i])$, public $bty* \cong \widehat{bty}*$, $i = \widehat{i}$, and $[1, [(l_{default}, 0)], [1], i] \cong_\psi [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}]$, by Lemma 3.2.42 we have $\widehat{\omega} = \text{EncodePtr}(\widehat{ty}*, [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}])$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))]$, $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma})$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and public $bty* \cong \widehat{bty}*$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{ty}, 1, \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, 1))]$ such that $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$, $(\widehat{ty} = \widehat{bty}*)$, $\text{GetIndirection}(*) = \widehat{i}$, $\widehat{l} = \phi()$, $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \widehat{ty})]$, $\widehat{\omega} = \text{EncodePtr}(\widehat{ty}*, [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}])$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{ty}, 1, \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, 1))]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}) \Downarrow'_{dp} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$ by Vanilla C rule Pointer Declaration.

Given $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$, by Definition 4.2.1 we have $(\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}) \Downarrow'_{dp} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $dp \cong dp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp1} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp1} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by Location-tracking SMC^2 rule Private Pointer Declaration, we have $((ty = bty*) \vee (ty = \text{private } bty*)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, $\text{GetIndirection}(*) = i$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \text{private } bty*)]$, $\omega = \text{EncodePtr}(\text{private } bty*, [1, [(l_{default}, 0)], [1], i])$, and $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $ty\ x \cong_\psi \widehat{ty}\ \widehat{x}$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp1} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin ty\ x$. Therefore, by Lemma 3.2.3 we have $ty\ x \cong \widehat{ty}\ \widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x) = \text{Erase}(ty)\ \text{Erase}(x)$, $\text{Erase}(ty) = \widehat{ty}$ and $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$. Therefore, we have $ty \cong \widehat{ty}$ such that $* = *$.

Given $((ty = bty*) \vee (ty = \text{private } bty*)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$ and $ty \cong \widehat{ty}$, we have private $bty* \cong \widehat{bty}*$ and $bty \cong \widehat{bty}$ by Definition 3.2.6.

Given $\text{GetIndirection}(*) = i$ and $* = *$, by Lemma 3.2.49 we have $\text{GetIndirection}(*) = \widehat{i}$ such that $i = \widehat{i}$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{private } bty*)]$, $x = \widehat{x}$, $l = \widehat{l}$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $\text{private } bty* \cong \widehat{bty}*$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \widehat{bty}*)]$ such that $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma})$.

Given $\omega = \text{EncodePtr}(\text{private } bty*, [1, [(l_{default}, 0)], [1], i])$, $i = \widehat{i}$, $\text{private } bty* \cong \widehat{bty}*$, and $[1, [(l_{default}, 0)], [1], i] \cong_\psi [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}]$, by Lemma 3.2.42 we have $\widehat{\omega} = \text{EncodePtr}(\widehat{bty}*, [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}])$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))]$, $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma})$, $\text{private } bty* \cong \widehat{bty}*$, $l = \widehat{l}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, 1))]$ such that $(\gamma_1, \sigma_1) \cong (\widehat{\gamma}_1, \widehat{\sigma}_1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$, $(\widehat{ty} = \widehat{bty}*)$, $\text{GetIndirection}(*) = \widehat{i}$, $\widehat{l} = \phi()$, $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \widehat{bty}*)]$, $\widehat{\omega} = \text{EncodePtr}(\widehat{bty}*, [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}])$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, 1))]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}) \Downarrow'_{dp} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$ by Vanilla C rule Pointer Declaration.

Given $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$, by Definition 4.2.1 we have $(\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp1} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x}) \Downarrow'_{dp} (\widehat{\gamma}_1, \widehat{\sigma}_1, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $dp1 \cong dp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp2} (\gamma_1, \sigma_1, \Delta, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp2} (\gamma_1, \sigma_1, \Delta, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$ by Location-tracking SMC^2 rule Private Pointer Declaration, we have $((ty = bty*) \vee (ty = \text{private } bty*)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $(\text{acc} > 0) \wedge ((\text{bid} = then) \vee (\text{bid} = else))$, $\text{GetIndirection}(*) = i$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \text{private } bty*)]$, $\omega = \text{EncodePtr}(\text{private } bty*, [1, [(l_{default}, 0)], [1], i])$, and $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $ty\ x \cong_\psi \widehat{ty}\ \widehat{x}$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_{dp2} (\gamma_1, \sigma_1, \Delta, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$, by

Lemma 4.2.2 we have $(l, \mu) \notin ty\ x$. Therefore, by Lemma 3.2.3 we have $ty\ x \cong \widehat{ty}\ \widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(ty\ x) = \text{Erase}(ty)\ \text{Erase}(x)$, $\text{Erase}(ty) = \widehat{ty}$ and $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$. Therefore, we have $ty \cong \widehat{ty}$ such that $* = *$.

Given $((ty = bty*) \vee (ty = \text{private}\ bty*)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$ and $ty \cong \widehat{ty}$, we have $\text{private}\ bty* \cong \widehat{bty}*$ and $bty \cong \widehat{bty}$ by Definition 3.2.6.

Given $\text{GetIndirection}(*) = i$ and $* = *$, by Lemma 3.2.49 we have $\text{GetIndirection}(*) = \widehat{i}$ such that $i = \widehat{i}$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{private}\ bty*)]$, $x = \widehat{x}$, $l = \widehat{l}$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $\text{private}\ bty* \cong \widehat{bty}*$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty}*)]$ such that $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given $\omega = \text{EncodePtr}(\text{private}\ bty*, [1, [(l_{default}, 0)], [1], i])$, $i = \widehat{i}$, $\text{private}\ bty* \cong \widehat{bty}*$, and $[1, [(l_{default}, 0)], [1], i] \cong_\psi [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}]$, by Lemma 3.2.42 we have $\widehat{\omega} = \text{EncodePtr}(\widehat{bty}*, [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}])$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega,\ \text{private}\ bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{private}\ bty*, \text{private}, 1))]$, $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$, $\text{private}\ bty* \cong \widehat{bty}*$, $l = \widehat{l}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.35 we have $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{ty}, \text{public}, 1))]$ such that $(\gamma_1, \sigma_1) \cong (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x})$, $(\widehat{ty} = \widehat{bty}*)$, $\text{GetIndirection}(*) = \widehat{i}$, $\widehat{l} = \phi()$, $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty}*)]$, $\widehat{\omega} = \text{EncodePtr}(\widehat{bty}*, [1,\ [(\widehat{l}_{default},\ 0)], [1], \widehat{i}])$, and $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{ty},\ \text{public},\ 1))]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}) \Downarrow'_{dp} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$ by Vanilla C rule Pointer Declaration.

Given $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, by Definition 4.2.1 we have $(\gamma_1, \sigma_1,\ \Delta,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x) \Downarrow^t_{dp1} (\gamma_1, \sigma_1,\ \Delta,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}) \Downarrow'_{dp} (\widehat{\gamma}_1,\ \widehat{\sigma}_1,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $dp2 \cong dp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \Downarrow^t_r (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \Downarrow_r^t (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$ by Location-tracking SMC^2 rule Read Public Variable, we have $\gamma(x) = (l, \text{public } bty)$, $\sigma(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1))$, and $\text{DecodeVal}(\text{public } bty, 1, \omega) = v$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{public } bty)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by public $bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, bty, \text{public}, 1))$ where $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodeVal}(\text{public } bty, 1, \omega) = v$, public $bty \cong \widehat{bty}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.41 we have $\text{DecodeVal}(\widehat{bty}, 1, \widehat{\omega}) = \widehat{v}$ and $v \cong_\psi \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty},\ 1, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))$, and $\text{DecodeVal}(\widehat{bty}, 1, \widehat{\omega}) = \widehat{v}$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow_r' (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$ by Vanilla C rule Read Variable.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $v \cong_\psi \widehat{v}$, by Definition 4.2.1 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \Downarrow_r^t (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow_r' (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$, $\Pi \cong_\psi \Sigma$, and $r \cong r$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \Downarrow_{r1}^t (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \Downarrow_{r1}^t (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$ by Location-tracking SMC^2 rule Read Public Variable, we have $\gamma(x) = (l, \text{private } bty)$, $\sigma(l) = (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))$, and $\text{DecodeVal}(\text{private } bty, 1, \omega) = v$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by private $bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega,\ \text{private } bty,\ 1,\ \text{PermL}(\text{Freeable},\ \text{private } bty,\ \text{private},\ 1))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, bty, \text{public}, 1))$ where $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodeVal}(\text{private } bty, 1, \omega) = v$, private $bty \cong \widehat{bty}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.41 we have $\text{DecodeVal}(\widehat{bty}, 1, \widehat{\omega}) = \widehat{v}$ and $v \cong_\psi \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty},\ 1, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, 1))$, and $\text{DecodeVal}(\widehat{bty}, 1, \widehat{\omega}) = \widehat{v}$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_r (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$ by Vanilla C rule Read Variable.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $v \cong_\psi \widehat{v}$, by Definition 4.2.1 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \Downarrow^t_{r1} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_r (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$, $\Pi \cong_\psi \Sigma$, and $r1 \cong r$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_w (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_w (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Public Write Variable, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, $v \neq \text{skip}$, $\text{acc} = 0$, $\gamma(x) = (l, \text{public } bty)$, and $\text{T_UpdateVal}(\sigma_1, l, v,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{public } bty) = (\sigma_2, \Delta_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_w (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong (\gamma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \sigma,\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ and $c_1 \cong d_1$.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$ and by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}$.

Given $\gamma(x) = (l, \text{public } bty)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by public $bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\text{T_UpdateVal}(\sigma_1, l, v,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{public } bty) = (\sigma_2, \Delta_2)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l = \widehat{l}$, $v \cong_\psi \widehat{v}$, and public $bty \cong \widehat{bty}$, by Lemma 4.2.4 we have $\text{UpdateVal}(\widehat{\sigma}_1, \widehat{l}, \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty})$, and $\text{UpdateVal}(\widehat{\sigma}_1,\ \widehat{l},\ \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_w (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ by Vanilla C rule Write.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_w (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_w (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $w \cong w$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{w2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{w2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private Write Variable, we have $\text{Label}(e, \gamma) = \text{private}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private } bty)$, and $\text{T_UpdateVal}(\sigma_1,\ l,\ v,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{private } bty) = (\sigma_2,\ \Delta_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{w2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$ by Definition 4.2.1. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ and $c_1 \cong d_1$. Given $v \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and

$v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private } bty)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by private $bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\text{T_UpdateVal}(\sigma_1, l, v,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{private } bty) = (\sigma_2, \Delta_2)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l = \widehat{l}$, $v \cong_\psi \widehat{v}$, and private $bty \cong \widehat{bty}$, by Lemma 4.2.4 we have $\text{UpdateVal}(\widehat{\sigma}_1, \widehat{l}, \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty})$, and $\text{UpdateVal}(\widehat{\sigma}_1,\ \widehat{l},\ \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_w (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ by Vanilla C rule Write.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{w2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_w (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $w2 \cong w$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{w1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{w1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Write Private Variable Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, $v \neq \text{skip}$, $\gamma(x) = (l,\ \text{private } bty)$, and $\text{T_UpdateVal}(\sigma_1,\ l,\ \text{encrypt}(v),\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{private } bty) = (\sigma_2, \Delta_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{w1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, by the inductive hypothesis we have

$(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ and $c_1 \cong d_1$. Given $v \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l,\ \text{private}\ bty)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that $l = \widehat{l}$ by $\text{private}\ bty \cong \widehat{bty}$ by Lemma 3.2.14.

Given $\text{T_UpdateVal}(\sigma_1,\ l,\ \text{encrypt}(v), \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{private}\ bty) = (\sigma_2, \Delta_2)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l = \widehat{l}$, $v \cong_\psi \widehat{v}$, and $\text{private}\ bty \cong \widehat{bty}$, by Lemma 4.2.4 we have $\text{UpdateVal}(\widehat{\sigma}_1, \widehat{l},\ \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty})$, and $\text{UpdateVal}(\widehat{\sigma}_1,\ \widehat{l},\ \widehat{v}, \widehat{bty}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_w (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ by Vanilla C rule Write.

Given $\Pi \triangleright (\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x = e) \Downarrow^t_{w1} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_w (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $w1 \cong w$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x) \Downarrow^t_{rp} (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ (l_1, \mu_1))$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x) \Downarrow^t_{rp} (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ (l_1, \mu_1))$ by Location-tracking SMC^2 rule Public Pointer Read Single Location, we have $\gamma(x) = (l,\ \text{public}\ bty*)$, $\sigma(l) = (\omega, \text{public}\ bty*,\ 1, \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1))$, and $\text{DecodePtr}(\text{public}\ bty*, 1,\ \omega) = [1, [(l_1, \mu_1)], [1],\ i]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l,\ \text{public}\ bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public}\ bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{public}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1,\ \mu_1)],\ [1],\ i]$, $\text{public } bty* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ \widehat{i}]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*)$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$, and $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ \widehat{i}]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{rp} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}_1, \widehat{\mu}_1))$ by Vanilla C rule Pointer Read Location.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, by Definition 4.2.1 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_1, \mu_1)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}_1, \widehat{\mu}_1))$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \Downarrow^t_{rp} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_1, \mu_1)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{rp} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}_1, \widehat{\mu}_1))$, $\Pi \cong_\psi \Sigma$, and $rp \cong rp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \Downarrow^t_{rp2} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_1, \mu_1))$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \Downarrow^t_{rp2} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_1, \mu_1))$ by Location-tracking SMC^2 rule Private Pointer Read Single Location, we have $\gamma(x) = (l,\ \text{private } bty*)$, $\sigma(l) = (\omega,\ \text{private } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))$, and
$\text{DecodePtr}(\text{private } bty*,\ 1,\ \omega) = [1, [(l_1,\ \mu_1)],\ [1],\ i]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l,\ \text{private } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private } bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{private } bty*, 1\ \text{PermL}(\text{Freeable},\ \text{private } bty*,\ \text{private}, 1))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, $\text{private } bty* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*), \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1, \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$, and $\text{DecodePtr}(\widehat{bty}*,$

$1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{rp} (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{l}_1, \widehat{\mu}_1))$ by Vanilla C rule Pointer Read Location.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, by Definition 4.2.1 we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l_1, \mu_1)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{l}_1, \widehat{\mu}_1))$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp2} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l_1, \mu_1)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{rp} (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{l}_1, \widehat{\mu}_1))$, $\Pi \cong_\psi \Sigma$, and $rp2 \cong rp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp1} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \overline{l}, \overline{j}, i])$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp1} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \overline{l}, \overline{j}, i])$ by Location-tracking SMC^2 rule Private Pointer Read Multiple Locations, we have $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, and $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$ and ψ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$, private $bty* \cong \widehat{bty*}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha, \overline{l}, \overline{j}, i], \text{private } bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha, \overline{l}, \overline{j}, i] \cong_\psi [1, (\widehat{l}_1, \widehat{\mu}_1), [1], i]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*}), \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$, and $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{rp} (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{l}_1, \widehat{\mu}_1))$ by Vanilla C rule Pointer Read Location.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $[\alpha, \overline{l}, \overline{j}, i] \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, by Definition 4.2.1 we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \overline{l}, \overline{j}, i]) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{l}_1, \widehat{\mu}_1))$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp1} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \overline{l}, \overline{j}, i]) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{rp} (\widehat{\gamma}, \widehat{\sigma}, \square, (\widehat{l}_1, \widehat{\mu}_1))$, $\Pi \cong_\psi \Sigma$, and $rp1 \cong rp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x = e) \Downarrow^{t}_{wp} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x = e) \Downarrow^{t}_{wp} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Public Pointer Write Single Location, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ (l_e, \mu_e))$, $\gamma(x) = (l, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\text{acc} = 0$, $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, and $\text{T_UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty*) = (\sigma_2, \Delta_2, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x = e) \cong (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_{\psi} \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x = e) \Downarrow^{t}_{wp} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_{\psi} (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ (l_e, \mu_e))$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (l_e, \widehat{\mu}_e))$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ (l_e, \mu_e)) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$ and $c_1 \cong d_1$. Given $(l_e, \mu_e) \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l_e, \mu_e) \cong_{\psi} (\widehat{l}_e, \widehat{\mu}_e)$.

Given $\gamma(x) = (l, \text{public } bty*)$, $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by public $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_{\psi} \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1,\ \mu_1)],\ [1],\ i]$, public $bty* \cong \widehat{bty}*$, and $\omega \cong_{\psi} \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ \widehat{i}]$ such that $(l_1, \mu_1) \cong_{\psi} (\widehat{l}_1,\ \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $\text{T_UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty*) = (\sigma_2, \Delta_2, 1)$, $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$, $(l, 0) \cong_{\psi} (\widehat{l}, 0)$, public $bty* \cong \widehat{bty}*$, and $[1,\ [(l_e,\ \mu_e)],\ [1],\ i] \cong_{\psi} [1, [(\widehat{l}_e, \widehat{\mu}_e)],\ [1],\ \widehat{i}]$, by Lemma 4.2.6 we have $\text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_e, \widehat{\mu}_e)],\ [1],\ \widehat{i}], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$ such that $(\gamma,\ \sigma_2) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,$ PermL(Freeable, $\widehat{bty}*$, public, 1)), DecodePtr($\widehat{bty}*, 1, \widehat{\omega}$) $= [1, [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ \widehat{i}]$, and UpdatePtr($\widehat{\sigma}_1$, $(\widehat{l}, 0)$, $[1,$ $[(\widehat{l}_e, \widehat{\mu}_e)],\ [1],\ \widehat{i}], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_{wp} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,$ skip) by Vanilla C rule Pointer Assign Location.

Given $(\gamma,\ \sigma_2) \cong_\psi (\gamma,\ \sigma_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},$ bid, acc, skip$) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2, \square,$ skip). Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $x = e) \Downarrow^t_{wp} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},$ bid, acc, skip$) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_{wp}$ $(\widehat{\gamma},\ \widehat{\sigma}_2, \square,$ skip), $\Pi \cong_\psi \Sigma$, and $wp \cong wp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $x = e) \Downarrow^t_{wp2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},$ bid, acc, skip$)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $x = e) \Downarrow^t_{wp2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},$ bid, acc, skip$)$ by Location-tracking SMC^2 rule Private Pointer Write Multiple Locations, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},$ bid, acc, $[\alpha, \overline{l}, \overline{j}, i])$, $\gamma(x) = (l,$ private $bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, and T_UpdatePtr($\sigma_1, (l,\ 0)$, $[\alpha, \overline{l}, \overline{j}, i]$, Δ_1, $\overline{\chi}$, bid, acc, private $bty*) = (\sigma_2, \Delta_2, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $x = e) \Downarrow^t_{wp2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},$ bid, acc, skip$)$, by Lemma 4.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have Erase$(x = e) = \widehat{x} = \widehat{e}$ where $x = \widehat{x}$ and $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,$ $\Delta,\ \overline{\chi},$ bid, acc, $e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},$ bid, acc, $[\alpha, \overline{l}, \overline{j}, i])$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},$ bid, acc, $[\alpha, \overline{l}, \overline{j}, i]) \cong_{\psi_1}$ $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$ and $c_1 \cong d_1$. Given $[\alpha, \overline{l}, \overline{j}, i] \neq$ skip, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $[\alpha, \overline{l}, \overline{j}, i] \cong_\psi (\widehat{l}_e, \widehat{\mu}_e)$.

Given $\gamma(x) = (l,$ private $bty*)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given T_UpdatePtr($\sigma, (l,\ 0)$, $[\alpha, \overline{l}, \overline{j}, i]$, Δ_1, $\overline{\chi}$, bid, acc, private $bty*) = (\sigma_1, \Delta_2, 1)$, $[\alpha, \overline{l}, \overline{j}, i] \cong_\psi (\widehat{l}_e, \widehat{\mu}_e)$,

$\text{private } bty* \cong \widehat{bty}*$, and $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, by Lemma 4.2.6 we have $\text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_e, \widehat{\mu}_e)],\ [1],\ \widehat{i}],\ \widehat{bty}*) = (\widehat{\sigma}_2, 1)$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1], \widehat{i}]$, and $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}, 0),\ [1,\ [(\widehat{l}_e, \widehat{\mu}_e)],\ [1],\ \widehat{i}], \widehat{bt}$ $(\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_{wp} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ by Vanilla C rule Pointer Assign Location.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = [\alpha, \overline{l}, \overline{j}, i]) \Downarrow^t_{wp2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$ $\Downarrow'_{wp} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wp2 \cong wp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{wp1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{wp1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private Pointer Assign Single Location, we have $\gamma(x) = (l,\ \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_e, \mu_e))$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \overline{l},\ \overline{j},\ i]$, $\text{Label}(e, \gamma) = \text{public}$, and $\text{T_UpdatePtr}(\sigma_1,\ (l, 0),\ [1,\ [(l_e, \mu_e)],\ [1],\ i],\ \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{private } bty*) = (\sigma_2, \Delta_2, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{wp1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l,\ \text{private } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private } bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_e, \mu_e))$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_e, \mu_e)) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$ and $c_1 \cong d_1$. Given $(l_e, \mu_e) \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$

and $(l_e, \mu_e) \cong_\psi (\widehat{l}_e, \widehat{\mu}_e)$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$, $\text{private } bty* \cong \widehat{bty}*$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha, \overline{l}, \overline{j}, i], \text{private } bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha, \overline{l}, \overline{j}, i] \cong_\psi [1, (\widehat{l}_1, \widehat{\mu}_1), [1], i]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $\text{T_UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*) = (\sigma_2, \Delta_2, 1)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, $\text{private } bty* \cong \widehat{bty}*$, and $[1, [(l_e, \mu_e)], [1], i] \cong_\psi [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}]$, by Lemma 4.2.6 we have $\text{UpdatePtr}(\widehat{\sigma}, (\widehat{l}, 0), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, (\widehat{l}_e, \widehat{\mu}_e))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, and $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}, 0), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}], \widehat{bty}$ $(\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{wp} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Pointer Assign Location.

Given $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wp1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{wp}$ $(\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wp1 \cong wp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^t_{rdp} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^t_{rdp} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$ by Location-tracking SMC^2 rule Private Pointer Dereference Single Location, we have $\gamma(x) = (l, \text{public } bty*)$, $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, and $\text{DerefPtr}(\sigma, \text{public } bty, (l_1, \mu_1)) = (v, 1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x})$ and ψ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x})$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $*x \cong_\psi *\widehat{x}$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^t_{rdp} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$, by Lemma 4.2.2 we have $(l, \mu) \notin *x$. Therefore, by Lemma 3.2.3 we have $*x \cong *\widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(*x) = *\widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{public } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by public $bty* \cong \widehat{bty*}$ by Lemma 3.2.14. Therefore, by Definition 3.2.6 we have $bty \cong \widehat{bty}$.

Given $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable, public } bty*, \text{public}, 1))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable, } \widehat{bty*}, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, public $bty* \cong \widehat{bty*}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \mu_1)],\ [1],\ 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \mu_1)$.

Given $\text{DerefPtr}(\sigma, \text{public } bty, (l_1, \mu_1)) = (v, 1)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, $bty \cong \widehat{bty}$, and $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$, by Lemma 3.2.58 we have $\text{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{v}, 1)$ such that $v \cong_\psi \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*}), \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1, \text{PermL}(\text{Freeable, } \widehat{bty*}, \text{ public, } 1)), \text{DecodeVal}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ 1]$, and $\text{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{v}, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'_{rdp} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$ by Vanilla C rule Pointer Dereference.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $v \cong_\psi \widehat{v}$, by Definition 4.2.1 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'_{rdp} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$, $\Pi \cong_\psi \Sigma$, and $rdp \cong rdp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp1} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_2, \mu_2))$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp1} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_2, \mu_2))$ by Location-tracking SMC^2 rule Public Pointer Dereference Single Location Higher Level Indirection, we have $\gamma(x) = (l, \text{public } bty*)$, $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable, public } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, $i > 1$, and $\text{DerefPtrHLI}(\sigma, \text{public } bty*, (l_1, \mu_1)) = ([1, [(l_2, \mu_2)], [1], i - 1], 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x \cong_\psi *\widehat{x}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp1} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_2, \mu_2))$, by Lemma 4.2.2 we have $(l, \mu) \notin *x$. Therefore, by Lemma 3.2.3 we have $*x \cong *\widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(*x) = *\widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{public } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by public $bty* \cong \widehat{bty}*$ by Lemma 3.2.14. Therefore, by Definition 3.2.6 we have $bty \cong \widehat{bty}$.

Given $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1,\ 0)],\ [1],\ i]$, public $bty* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ 0)],\ [1],\ \widehat{i}]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $i > 1$ and $i = \widehat{i}$, we have $\widehat{i} > 1$.

Given $\text{DerefPtrHLI}(\sigma, \text{public } bty*, (l_1, \mu_1)) = ([1, [(l_2, \mu_2)], [1], i-1], 1)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$, $bty \cong \widehat{bty}$, and $i = \widehat{i}$, by Lemma 3.2.59 we have $\text{DerefPtrHLI}(\widehat{\sigma},\ \widehat{bty}*,\ (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1], 1)$ such that $[1, [(l_2, \mu_2)], [1], i-1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1]$ and $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*)$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ \widehat{i}]$, $\widehat{i} > 1$, and $\text{DerefPtrHLI}(\widehat{\sigma},\ \widehat{bty}*,\ (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1], 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'_{rdp1} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}_2, \hat{\mu}_2))$ by Vanilla C rule Pointer Dereference Higher Level Indirection.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \hat{\mu}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_2, \mu_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}_2, \hat{\mu}_2))$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp1} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_2, \mu_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'_{rdp1} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}_2, \hat{\mu}_2))$, $\Pi \cong_\psi \Sigma$, and $rdp1 \cong rdp1$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp2} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp2} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$ by Location-tracking SMC^2 rule Private Pointer Dereference, we have $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable},\ \text{private } bty*,\ \text{private},\ \alpha))$, $\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \overline{l},\ \overline{j},\ 1]$, and $\text{Retrieve_vals}(\alpha, \overline{l}, \overline{j}, \text{private } bty, \sigma) = (v, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x})$, by Definition 4.2.1 we have

$(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x \cong_\psi *\widehat{x}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp2} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, by Lemma 4.2.2 we have $(l, \mu) \notin *x$. Therefore, by Lemma 3.2.3 we have $*x \cong *\widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(*x) = *\widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty}*$ by Lemma 3.2.14. Therefore, by Definition 3.2.6 we have $bty \cong \widehat{bty}$.

Given $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \overline{l},\ \overline{j},\ 1]$, $\text{private } bty* \cong \widehat{bty}*$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha, \overline{l}, \overline{j}, 1], \text{private } bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha, \overline{l}, \overline{j}, 1] \cong_\psi [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$.

Given $\text{Retrieve_vals}(\alpha, \overline{l}, \overline{j}, \text{private } bty, \sigma) = (v, 1)$, $\text{DeclassifyPtr}([\alpha,\ \overline{l},\ \overline{j},\ 1],\ \text{private } bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $\text{private } bty \cong \widehat{bty}$, by Lemma 3.2.60 we have $\text{DerefPtr}(\widehat{\sigma},\ \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{v}, 1)$ such that $v \cong_\psi \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*), \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1)), \text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \mu_1)],\ [1],\ 1]$, and $\text{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{v}, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'_{rdp} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$ by Vanilla C rule Pointer Dereference.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $v \cong_\psi \widehat{v}$, by Definition 4.2.1 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp2} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'_{rdp} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$, $\Pi \cong_\psi \Sigma$, and $rdp2 \cong rdp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp3} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [\alpha', \overline{l}', \overline{j}', i-1])$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp3} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [\alpha', \overline{l}', \overline{j}', i-1])$ by Location-tracking SMC^2 rule Private Pointer Dereference Higher Level Indirection, we have $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$, $i > 1$, and $\text{DerefPrivPtr}(\alpha, \overline{l}, \overline{j}, \text{private } bty*, \sigma) = ((\alpha', \overline{l}', \overline{j}'), 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x \cong_\psi *\widehat{x}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp3} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [\alpha', \overline{l}', \overline{j}', i-1])$, by Lemma 4.2.2 we have $(l, \mu) \notin *x$. Therefore, by Lemma 3.2.3 we have $*x \cong *\widehat{x}$. By Definition 3.2.10 we have $\text{Erase}(*x) = *\widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega,\ \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable},\ \text{private } bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \overline{l},\ \overline{j},\ i]$, private $bty* \cong \widehat{bty*}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha, \overline{l}, \overline{j}, i], \text{private } bty*) = (l_1,\ 0)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[\alpha, \overline{l}, \overline{j}, i] \cong_\psi [1, (\widehat{l}_1, 0), [1], i]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$ and $i = \widehat{i}$.

Given $i > 1$ and $i = \widehat{i}$, we have $\widehat{i} > 1$.

Given $\text{DerefPrivPtr}(\alpha, \overline{l}, \overline{j}, \text{private } bty*, \sigma) = ((\alpha', \overline{l}', \overline{j}'), 1)$, $i = \widehat{i}$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $\text{DeclassifyPtr}([\alpha, \overline{l},\ \overline{j},\ i], \text{private } bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, $\text{DeclassifyPtr}([\alpha',\ \overline{l}',\ \overline{j}',\ i-1],\ \text{private } bty*) = (l_2, \mu_2)$, and private $bty* \cong \widehat{bty*}$, by Lemma 3.2.57 we have $\text{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty*}, (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1], 1)$ such that $[\alpha', \overline{l}', \overline{j}', i-1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1]$ and $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*}), \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty*},\ \text{public},\ 1)), \text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1, \mu_1)],\ [1],\ \widehat{i}], \widehat{i} > 1, \widehat{\sigma}(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$, and $\text{DecodeVal}(\widehat{bty*}, 1, \widehat{\omega}_1) = [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'_{rdp1} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}_2, \widehat{\mu}_2))$ by Vanilla C rule Pointer Dereference Higher Level Indirection.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $[\alpha', \overline{l}', \overline{j}', i-1] \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [\alpha', \overline{l}', \overline{j}', i-1]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}_2, \widehat{\mu}_2))$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x) \Downarrow^t_{rdp3} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [\alpha', \overline{l}', \overline{j}', i-1]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x}) \Downarrow'_{rdp1} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ (\widehat{l}_2, \widehat{\mu}_2))$, $\Pi \cong_\psi \Sigma$, and $rdp3 \cong rdp1$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ *x =\ e) \Downarrow^{t}_{wdp} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x =\ e) \Downarrow^{t}_{wdp} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Public Pointer Dereference Write Public Value, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{public } bty*)$, $\sigma_1(l) = (\omega,\ \text{public } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{public } bty*,\ \text{public},\ 1))$, $\text{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1,\ \mu_1)],\ [1],\ 1]$, $\text{T_UpdateOffset}(\sigma_1, (l_1, \mu_1), v,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{public } bty) = (\sigma_2,\ \Delta_2,\ 1)$, $\text{Label}(e, \gamma) = \text{public}$, and $(\text{acc} = 0) \wedge (\text{bid} = none)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ *x = e) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $*x = e \cong_{\psi} *\widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ *x =\ e) \Downarrow^{t}_{wdp} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin *x =\ e$. Therefore, by Lemma 3.2.3 we have $*x =\ e \cong *\widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(*x = e) = *\widehat{x} = \text{Erase}(e)$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_{\psi} (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ and $c_1 \cong d_1$. Given $v \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_{\psi} \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong_{\psi} \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{public } bty*)$, $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by public $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_{\psi} \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, public $bty* \cong \widehat{bty}*$, and $\omega \cong_{\psi} \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ 1]$ such that $(l_1, \mu_1) \cong_{\psi} (\widehat{l}_1,\ \widehat{\mu}_1)$.

Given $\text{T_UpdateOffset}(\sigma_1, (l_1, \mu_1), v, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty) = (\sigma_2, \Delta_2, 1)$, $(\gamma, \sigma_1) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$, $(l_1, \mu_1) \cong_{\psi} (\widehat{l}_1, \widehat{\mu}_1)$, public $bty \cong \widehat{bty}$, and $v \cong_{\psi} \widehat{v}$, by Lemma 4.2.5 we have $\text{UpdateOffset}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1),\ \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$ such that $(\gamma,\ \sigma_2) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$. $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty*})$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,$ PermL(Freeable, $\widehat{bty*}$, public, 1)), DecodePtr($\widehat{bty*}$, 1, $\widehat{\omega}$) = [1, [$(\widehat{l}_1,\ \widehat{\mu}_1)$], [1], 1], and UpdateOffset($\widehat{\sigma}_1$, $(\widehat{l}_1,\ \widehat{\mu}_1)$, $\widehat{v}$, $\widehat{bty}$) $= (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e}) \Downarrow'_{wdp} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ by Vanilla C rule Pointer Dereference Write Value.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^t_{wdp} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e}) \Downarrow'_{wdp} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wdp \cong wdp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^t_{wdp1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^t_{wdp1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Public Pointer Dereference Write Higher Level Indirection, we have $(\text{acc} = 0) \wedge (\text{bid} = none)$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_e, \mu_e))$, $\gamma(x) = (l, \text{public } bty*)$, $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, DecodePtr(public $bty*$, 1, ω) = [1, [$(l_1,\ \mu_1)$], [1], i], $i > 1$, Label(e, γ) = public, and T_UpdatePtr(σ_1, (l_1, μ_1), [1, [$(l_e,\ \mu_e)$], [1], $i - 1$], Δ_1, $\overline{\chi}$, bid, acc, public $bty*$) $= (\sigma_2, \Delta_2, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x = e \cong_\psi *\widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^t_{wdp1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin *x = e$. Therefore, by Lemma 3.2.3 we have $*x = e \cong *\widehat{x} = \widehat{e}$. By Definition 3.2.10 we have Erase($*x = e$) $= *\widehat{x} =$ Erase(e) where $x = \widehat{x}$ and Erase(e) $= \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_e, \mu_e))$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ (l_e, \mu_e)) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$ and $c_1 \cong d_1$. Given $(l_e, \mu_e) \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l_e, \mu_e) \cong_\psi (\widehat{l}_e, \widehat{\mu}_e)$.

Given $\gamma(x) = (l, \text{public } bty*)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by public $bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1, \text{PermL}(\text{Freeable},\ \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1,\ \mu_1)],\ [1],\ i]$, $\text{public } bty* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $i > 1$ and $i = \widehat{i}$, we have $\widehat{i} > 1$.

Given $\text{T_UpdatePtr}(\sigma_1, (l_1, \mu_1), [1, [(l_e, \mu_e)], [1], i-1]\ \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{private } bty*) = (\sigma_2, \Delta_2, 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, $\text{public } bty* \cong \widehat{bty}*$, and $[1,\ [(l_e,\ \mu_e)],\ [1],\ 1] \cong_\psi [1, [(\widehat{l}_e,\ \widehat{\mu}_e)], [1], 1]$, by Lemma 4.2.6 we have $\text{UpdatePtr}(\widehat{\sigma}_1,\ (\widehat{l}_1, \widehat{\mu}_1),\ [1,\ [(\widehat{l}_e,\ \widehat{\mu}_e)], [1],\ i], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1, \text{PermL}(\text{Freeable},\ \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ \widehat{i}]$, $\widehat{i} > 1$, and $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e}) \Downarrow'_{wdp1} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Pointer Dereference Write Higher Level Indirection.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^t_{wdp1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e}) \Downarrow'_{wdp1} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wdp1 \cong wdp1$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^t_{wdp2} (\gamma,\ \sigma_2,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^t_{wdp2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private Pointer Dereference Write Higher Level Indirection, we have $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ (l_e, \mu_e))$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$, and $\text{T_UpdatePrivPtr}(\sigma_1, [\alpha, \overline{l}, \overline{j}, i], [1, [(l_e, \mu_e)], [1], i-1]\ \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ \text{private } bty*) = (\sigma_2, \Delta_2, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $*x = e \cong_\psi *\widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^t_{wdp2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},$

acc, skip), by Lemma 4.2.2 we have $(l, \mu) \notin *x = e$. Therefore, by Lemma 3.2.3 we have $*x = e \cong *\widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(*x = e) = *\widehat{x} = \text{Erase}(e)$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ (l_e, \mu_e))$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ (l_e, \mu_e)) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$ and $c_1 \cong d_1$. Given $(l_e, \mu_e) \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l_e, \mu_e) \cong_\psi (\widehat{l}_e, \widehat{\mu}_e)$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty*},\ \text{public},\ 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$, private $bty* \cong \widehat{bty*}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha, \overline{l},\ \overline{j},\ i],$ private $bty*) = (l_1, 0)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ (\widehat{l}_1, 0),\ [1],\ i]$ where $[\alpha, \overline{l}, \overline{j}, i] \cong_\psi [1, (\widehat{l}_1, 0), [1], \widehat{i}]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$ and $i = \widehat{i}$.

Given $\text{T_UpdatePrivPtr}(\sigma_1,\ [\alpha,\ \overline{l},\ \overline{j},\ i],\ [1,\ [(l_e,\ \mu_e)],\ [1], i-1], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*) = (\sigma_2,\ \Delta_2, 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, private $bty* \cong \widehat{bty*}$, $\text{DeclassifyPtr}([\alpha, \overline{l}, \overline{j},\ i],\ \text{private } bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, and $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1] \cong [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1]$, by Lemma 4.2.8 we have $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1,\ [(\widehat{l}_e, \widehat{\mu}_e)],$ $[1], \widehat{i}-1], \widehat{bty*}) = (\widehat{\sigma}_2, 1)$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e,\ \widehat{\mu}_e))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*})$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1,$ $\text{PermL}(\text{Freeable},\ \widehat{bty*},\ \text{public},\ 1))$, $\text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1], \widehat{i}]$, $\widehat{i} > 1$, and $\text{UpdatePtr}(\widehat{\sigma}_1,$ $(\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1],\ \widehat{bty*}) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square, *\widehat{x} = \widehat{e}) \Downarrow'_{wdp1} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \text{skip})$ by Vanilla C rule Pointer Dereference Write Higher Level Indirection.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ *x = e) \Downarrow^t_{wdp2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$ $\Downarrow'_{wdp1} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wdp2 \cong wdp1$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^{t}_{wdp3} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^{t}_{wdp3} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private Pointer Dereference Write Private Value, we have $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, $v \neq \text{skip}$, $\sigma_1(l) = (\omega,\ \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{Label}(e, \gamma) = \text{private}$, $(bty = \text{int}) \vee (bty = \text{float})$, $\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha, \overline{l}, \overline{j}, 1]$, $\text{acc} = 0$, and $\text{T_UpdatePriv}(\sigma_1,\ \alpha,\ \overline{l},\ \overline{j},\ \text{private } bty,\ v,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc}) = (\sigma_2, \Delta_2, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $*x = e \cong_{\psi} *\widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^{t}_{wdp3} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin *x = e$. Therefore, by Lemma 3.2.3 we have $*x = e \cong *\widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(*x = e) = *\widehat{x} = \text{Erase}(e)$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_{\psi} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ and $c_1 \cong d_1$. Given $v \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_{\psi} \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_{\psi} \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \overline{l},\ \overline{j},\ 1]$, private $bty* \cong \widehat{bty}*$, $\omega \cong_{\psi} \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha,\ \overline{l},\ \overline{j},\ 1], \text{private } bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha, \overline{l}, \overline{j}, 1] \cong_{\psi} [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ such that $(l_1, \mu_1) \cong_{\psi} (\widehat{l}_1, \widehat{\mu}_1)$.

Given T_UpdatePriv$(\sigma_1, \alpha, \bar{l}, \bar{j}, \text{private } bty, v, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_2, \Delta_2, 1)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, private $bty \cong \widehat{bty}$ DeclassifyPtr$([\alpha, \bar{l}, \bar{j}, i], \text{private } bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, private $bty* \cong \widehat{ty}$, and $v \cong_\psi \widehat{v}$, by Lemma 4.2.7 we have UpdateOffset$(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x} = \widehat{e})$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v})$, $\widehat{v} \neq$ skip, $\widehat{\gamma}(x) = (\widehat{l}, \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$, DecodePtr$(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$, and UpdateOffset$(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x} = \widehat{e}) \Downarrow'_{wdp} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Pointer Dereference Write Value.

Given $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^t_{wdp3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x} = \widehat{e}) \Downarrow'_{wdp} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wdp3 \cong wdp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^t_{wdp4} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^t_{wdp4} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by Location-tracking SMC^2 rule Private Pointer Dereference Write Public Value, we have Label$(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$, $v \neq$ skip, acc $= 0$, $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, DecodePtr$(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, and T_UpdatePriv$(\sigma_1, \alpha, \bar{l}, \bar{j}, \text{private } bty, \text{encrypt}(v), \Delta_1, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_2, \Delta_2, 1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x} = \widehat{e})$ and ψ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, *\widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $*x = e \cong_\psi *\widehat{x} = \widehat{e}$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^t_{wdp4} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin *x = e$. Therefore, by Lemma 3.2.3 we have $*x = e \cong *\widehat{x} = \widehat{e}$. By Definition 3.2.10 we have Erase$(*x = e) = *\widehat{x} =$ Erase(e) where $x = \widehat{x}$ and Erase$(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e})$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \cong_\psi (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e)$ by Definition 4.2.1. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v})$ and ψ_1 such that $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{v})$ and $c_1 \cong d_1$. Given $v \neq$ skip, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}$. Given Label$(e_2, \gamma) = \text{public}$, we have Label$(v, \gamma) = \text{public}$ and therefore $v = \widehat{v}$ by Definition 3.2.18 and

Definition 3.2.10.

Given $v \neq \text{skip}$ and $v \cong \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ 1]$, $\text{private } bty* \cong \widehat{bty*}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ 1], \text{private } bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha, \bar{l}, \bar{j}, 1] \cong_\psi [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$.

Given $v = \widehat{v}$, by Definition 3.2.18 we have $\text{encrypt}(v) \cong_\psi \widehat{v}$. Given $\text{T_UpdatePriv}(\sigma_1, \alpha,\ \bar{l},\ \bar{j},\ \text{private } bty,\ \text{encrypt}(v),\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc}) = (\sigma_2, \Delta_2, 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ i],\ \text{private } bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \mu_1)$, $\text{private } bty \cong \widehat{bty}$, and $v \cong_\psi \widehat{v}$, by Lemma 4.2.7 we have $\text{UpdateOffset}(\widehat{\sigma}_1,\ (\widehat{l}_1, \widehat{\mu}_1),\ \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty*})$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty*},\ \text{public},\ 1))$, $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ 1]$, and $\text{UpdateOffset}(\widehat{\sigma}_1,\ (\widehat{l}_1, \widehat{\mu}_1),\ \widehat{v}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e}) \Downarrow'_{wdp} (\widehat{\gamma},\ \widehat{\sigma}_2, \square, \text{skip})$ by Vanilla C rule Pointer Dereference Write Value.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^t_{wdp4} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e}) \Downarrow'_{wdp} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wdp4 \cong wdp$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^t_{wdp5} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^t_{wdp5} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private Pointer Dereference Write Higher Level Indirection Multiple Locations, we have $\gamma(x) = (l, \text{private}$

$bty*$), (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1]$), $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{acc} = 0$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$, $i > 1$, and $\text{T_UpdatePrivPtr}(\sigma_1, [\alpha, \overline{l}, \overline{j}, i], [\alpha_e, \overline{l}_e, \overline{j}_e, i-1], \Delta_1, \overline{\chi}, \text{bid}, \text{acc private } bty*) = (\sigma_2, \Delta_2, 1)$.

Given ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $*\widehat{x} = \widehat{e}$) and ψ such that (γ, σ, Δ, $\overline{\chi}$, bid, acc, $*x = e$) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $*\widehat{x} = \widehat{e}$), by Definition 4.2.1 we have (γ, σ) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$) and $*x = e \cong_\psi *\widehat{x} = \widehat{e}$. Given ($\gamma$, σ, Δ, $\overline{\chi}$, bid, acc, $*x = e$) $\Downarrow^t_{wdp5}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, skip), by Lemma 4.2.2 we have $(l, \mu) \notin *x = e$. Therefore, by Lemma 3.2.3 we have $*x = e \cong *\widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(*x = e) = *\widehat{x} = \text{Erase}(e)$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l, \text{private } bty*)$, ($\gamma$, σ) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$), and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given (γ, σ) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$) and $e \cong \widehat{e}$, by Lemma 4.2.3 we have ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}$) such that ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}$) $\cong_\psi$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, e). Given (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1]$), by the inductive hypothesis we have ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}$) $\Downarrow'_{d_1}$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $(\widehat{l}_e, \widehat{\mu}_e)$) and ψ_1 such that (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1]$) $\cong_{\psi_1}$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $(\widehat{l}_e, \widehat{\mu}_e)$) and $c_1 \cong d_1$. Given $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1] \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have (γ, σ_1) $\cong_\psi$ $(\widehat{\gamma}, \widehat{\sigma}_1)$ and $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1] \cong_\psi [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1]$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, ($\gamma$, σ_1) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$), and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$, private $bty* \cong \widehat{bty*}$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha, \overline{l}, \overline{j}, i], \text{private } bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha, \overline{l}, \overline{j}, i] \cong_\psi [1, (\widehat{l}_1, \widehat{\mu}_1), [1], i]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$ and $i = \widehat{i}$.

Given $i > 1$ and $i = \widehat{i}$, we have $\widehat{i} > 1$.

Given $\text{T_UpdatePrivPtr}(\sigma_1, [\alpha, \overline{l}, \overline{j}, i], [\alpha_e, \overline{l}_e, \overline{j}_e, i-1], \Delta_1, \overline{\chi}, \text{bid}, \text{acc private } bty*) = (\sigma_2, \Delta_2, 1)$, ($\gamma$, σ_1) $\cong_\psi$ $(\widehat{\gamma}, \widehat{\sigma}_1)$, private $bty* \cong \widehat{bty*}$, $\text{DeclassifyPtr}([\alpha, \overline{l}, \overline{j}, i], \text{private } bty*) = (l_1, \mu_1)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, and $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1] \cong_\psi [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1]$, by Lemma 4.2.8 we have $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1], \widehat{bty*}) = (\widehat{\sigma}_2, 1)$ such that (γ, σ_2) $\cong_\psi$ $(\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_e, \widehat{\mu}_e))$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,$ PermL(Freeable, $\widehat{bty}*$, public, 1)), DecodePtr($\widehat{bty}*$, 1, $\widehat{\omega}$) $= [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1], \widehat{i}]$, $\widehat{i} > 1$, and UpdatePtr($\widehat{\sigma}_1$, $(\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e}) \Downarrow'_{wdp1} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square, \text{skip})$ by Vanilla C rule Pointer Dereference Write Higher Level Indirection.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ *x = e) \Downarrow^t_{wdp5} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ *\widehat{x} = \widehat{e})$ $\Downarrow'_{wdp1} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wdp5 \cong wdp1$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++\ x) \Downarrow^t_{pin} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v_1)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++\ x) \Downarrow^t_{pin} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v_1)$ by Location-tracking SMC^2 rule Pre-Increment Public Variable, we have $\text{acc} = 0$, $\gamma(x) = (l,\ \text{public}\ bty)$, $\sigma(l) = (\omega,\ \text{public}\ bty,\ 1,$ PermL(Freeable, public bty, public, 1)), DecodeVal(public bty, 1, ω) $= v$, $v_1 =_{\text{public}} v +_{\text{public}} 1$, and T_UpdateVal($\sigma, l, v_1, \Delta, \overline{\chi}$, bid, acc, public bty) $= (\sigma_1, \Delta_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $++\ x \cong_\psi ++\ \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have Erase($++x$) $=++\ \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{public}\ bty)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by public $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{public}\ bty, 1,$ PermL(Freeable, public bty, public, 1)), $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1,$ PermL(Freeable, bty, public, 1)) where $\omega \cong_\psi \widehat{\omega}$.

Given DecodeVal(public $bty, 1, \omega$) $= v$, public $bty \cong \widehat{bty}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.41 we have DecodeVal($\widehat{bty}$, $1, \widehat{\omega}$) $= \widehat{v}$ and $v \cong_\psi \widehat{v}$.

Given $v_1 =_{\text{public}} v +_{\text{public}} 1$ and $v \cong_\psi \widehat{v}$, by Lemma 3.2.22 we have $\widehat{v}_1 = \widehat{v} + 1$ such that $v_1 \cong_\psi \widehat{v}_1$.

Given T_UpdateVal($\sigma, l, v_1, \Delta, \overline{\chi}$, bid, acc, public bty) $= (\sigma_1, \Delta_1)$, public $bty \cong \widehat{bty}$ $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $l = \widehat{l}$, and

$v_1 \cong_\psi \widehat{v}_1$, by Lemma 4.2.4 we have $\text{UpdateVal}(\widehat{\sigma}, \widehat{l}, \widehat{v}_1, \widehat{bty}) = \widehat{\sigma}_1$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty},\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty},\ \text{public},\ 1))$, $\text{DecodeVal}(\widehat{bty},\ 1,\ \widehat{\omega}) = \widehat{v}$, $\widehat{v}_1 = \widehat{v} + 1$, and $\text{UpdateVal}(\widehat{\sigma},\ \widehat{l},\ \widehat{v}_1, \widehat{bty}) = \widehat{\sigma}_1$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x}) \Downarrow'_{pin} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_1)$ by Vanilla C rule Pre-Increment Variable.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v_1 \cong_\psi \widehat{v}_1$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_1)$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++\ x) \Downarrow^t_{pin} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x}) \Downarrow'_{pin} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_1)$, $\Pi \cong_\psi \Sigma$, and $pin \cong pin$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++\ x)\ \Downarrow^t_{pin1}\ (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v_1)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++\ x)\ \Downarrow^t_{pin1}\ (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v_1)$ by Location-tracking SMC^2 rule Pre-Increment Private Variable, we have $\text{acc} = 0$, $\gamma(x) = (l, \text{private } bty)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))$, $\text{DecodeVal}(\text{private } bty, 1, \omega) = v$, $v_1 =_{\text{private}} v +_{\text{private}} \text{encrypt}(1)$, and $\text{T_UpdateVal}(\sigma, l, v_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty) = (\sigma_1, \Delta_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ++x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $++\ x \cong_\psi ++\ \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(++x) = ++\ \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \text{PermL}(\text{Freeable}, bty, \text{public}, 1))$ where $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodeVal}(\text{private } bty, 1, \omega) = v$, private $bty \cong \widehat{bty}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.41 we have $\text{DecodeVal}(\widehat{bty},\ 1, \widehat{\omega}) = \widehat{v}$ and $v \cong_\psi \widehat{v}$.

Given $\text{encrypt}(1)$, by Definition 3.2.18 we have $\text{encrypt}(1) \cong_\psi 1$. Given $v_1 =_{\text{private}} v +_{\text{private}} \text{encrypt}(1)$ and $v \cong_\psi \widehat{v}$, by Lemma 3.2.22 we have $\widehat{v}_1 = \widehat{v} + 1$ such that $v_1 \cong_\psi \widehat{v}_1$.

Given T_UpdateVal($\sigma, l, v_1, \Delta, \overline{\chi}$, bid, acc, private bty) $= (\sigma_1, \Delta_1)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $l = \widehat{l}$, private $bty \cong \widehat{bty}$, and $v_1 \cong_\psi \widehat{v}_1$, by Lemma 4.2.4 we have UpdateVal($\widehat{\sigma}, \widehat{l}, \widehat{v}_1, \widehat{bty}$) $= \widehat{\sigma}_1$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty},\ 1,$ PermL(Freeable, $\widehat{bty}$, public, 1)), DecodeVal($\widehat{bty}$, 1, $\widehat{\omega}$) $= \widehat{v}$, $\widehat{v}_1 = \widehat{v} + 1$, and UpdateVal($\widehat{\sigma},\ \widehat{l},\ \widehat{v}_1, \widehat{bty}$) $= \widehat{\sigma}_1$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x}) \Downarrow'_{pin} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_1)$ by Vanilla C rule Pre-Increment Variable.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v_1 \cong_\psi \widehat{v}_1$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}$, bid, acc, $v_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_1)$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}$, bid, acc, $++\ x) \Downarrow^t_{pin1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}$, bid, acc, $v_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x}) \Downarrow'_{pin}$ $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_1)$, $\Pi \cong_\psi \Sigma$, and $pin1 \cong pin$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}$, bid, acc, $++\ x) \Downarrow^t_{pin2} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}$, bid, acc, $(l_2, \mu_2))$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}$, bid, acc, $++\ x) \Downarrow^t_{pin2} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}$, bid, acc, $(l_2, \mu_2))$ by Location-tracking SMC^2 rule Pre-Increment Public Pointer Single Location, we have acc $= 0$, $\gamma(x) = (l,$ public $bty*)$, $\sigma(l) = (\omega,$ public $bty*$, 1, PermL(Freeable, public $bty*$, public, 1)), DecodePtr(public $bty*$, 1, ω) $= [1,\ [(l_1,\ \mu_1)],\ [1],\ 1]$, $((l_2,\ \mu_2),\ 1) =$ GetLocation($(l_1, \mu_1), \tau$(public bty), σ), and T_UpdatePtr($\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \Delta, \overline{\chi}$, bid, acc, public $bty*$) $= (\sigma_1, \Delta_1, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}$, bid, acc, $++x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $++\ x \cong_\psi ++\ \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have Erase($++x$) $=++\ \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l,$ public $bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by public $bty* \cong \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega,$ public $bty*$, 1, PermL(Freeable, public $bty*$, public, 1)), $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1,$ PermL(Freeable, $\widehat{bty}*$, public, 1)) such that $\omega \cong_\psi \widehat{\omega}$.

Given DecodePtr(public $bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, public $bty* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have DecodePtr($\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$.

Given $((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty), \sigma)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, public $bty \cong \widehat{bty}$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, by Lemma 3.2.50 we have $\text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty}), \widehat{\sigma}) = ((\widehat{l}_2, \widehat{\mu}_2), 1)$ such that $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$. By Definition 3.2.14 we have $[\alpha, \overline{l}', \overline{j}, 1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1]$.

Given $\text{T_UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty*) = (\sigma_1, \Delta_1, 1)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, public $bty* \cong \widehat{bty}*$, and $[1, [(l_2, \mu_2)], [1], 1] \cong_\psi [1, [(\widehat{l}_2, \mu_2)], [1], 1]$, by Lemma 4.2.6 we have $\text{UpdatePtr}(\widehat{\sigma}, (\widehat{l}, 0), [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], i], \text{public } bty*) = (\widehat{\sigma}_1, 1)$ such that $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$, $((\widehat{l}_2, \widehat{\mu}_2), 1) = \text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty}))$, and $\text{UpdatePtr}(\widehat{\sigma}, (\widehat{l}, 0), [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1], \widehat{bty}*) = (\widehat{\sigma}_1, 1)$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x}) \Downarrow'_{pin2} (\widehat{\gamma}, \widehat{\sigma}_1, \square, (\widehat{l}_2, \widehat{\mu}_2))$ by Vanilla C rule Pre-Increment Pointer.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$, by Definition 4.2.1 we have $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1, \square, (\widehat{l}_2, \widehat{\mu}_2))$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2)) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x}) \Downarrow'_{pin2} (\widehat{\gamma}, \widehat{\sigma}_1, \square, (\widehat{l}_2, \widehat{\mu}_2))$, $\Pi \cong_\psi \Sigma$, and $pin2 \cong pin2$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin3} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin6} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin6} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$ by Location-tracking SMC^2 rule Pre-Increment Private Pointer Single Location, we have $\text{acc} = 0$, $\gamma(x) = (l, \text{private } bty*)$, $\text{GetLocation}((l_1, \mu_1), \sigma(l) = (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, $\tau(\text{private } bty), \sigma) = ((l_2, \mu_2), 1)$, and $\text{T_UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*) = (\sigma_1, \Delta_1, 1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x})$ and ψ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, ++\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $++\ x \cong_\psi ++\ \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(++x) = ++\ \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that $l = \widehat{l}$ by private $bty* \cong \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, private $bty* \cong \widehat{bty*}$, and $\omega \cong_\psi \widehat{\omega}$, by Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty*},\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1,\ \widehat{\mu}_1)$.

Given $((l_2, \mu_2), 1) = \text{GetLocation}((l_1,\ \mu_1), \tau(\text{private } bty), \sigma)$, $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, private $bty \cong \widehat{bty}$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, by Lemma 3.2.50 we have $\text{GetLocation}((\widehat{l}_1,\ \widehat{\mu}_1),\ \tau(\widehat{bty}), \widehat{\sigma}) = ((\widehat{l}_2,\ \widehat{\mu}_2), 1)$ such that $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$. By Definition 3.2.14 we have $[1, [(l_2, \mu_2)], [1], 1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1]$.

Given $\text{T_UpdatePtr}(\sigma, (l, 0), [1,\ [(l_2,\ \mu_2)],\ [1],\ 1], \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{private } bty*) = (\sigma_1, \Delta_1, 1)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, private $bty* \cong \widehat{bty*}$, and $[1,\ [(l_2,\ \mu_2)],\ [1],\ 1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)],\ [1],\ 1]$, by Lemma 4.2.6 we have $\text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_2, \widehat{\mu}_2)],\ [1],\ i], \widehat{bty*}) = (\widehat{\sigma}_1, 1)$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x}), \widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \widehat{bty*})$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty*},\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty*},\ \text{public},\ 1))$, $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ \widehat{\mu}_1)],\ [1],\ 1]$, $((\widehat{l}_2,\ \widehat{\mu}_2), 1) = \text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty}), \widehat{\sigma})$, and $\text{UpdatePtr}(\widehat{\sigma}, (\widehat{l}, 0), [1,\ [(\widehat{l}_2, \widehat{\mu}_2)],\ [1],\ 1], \widehat{bty*}) = (\widehat{\sigma}_1, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x}) \Downarrow'_{pin2} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square, (\widehat{l}_2, \widehat{\mu}_2))$ by Vanilla C rule Pre-Increment Pointer.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ (l_2, \mu_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_2, \widehat{\mu}_2))$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ ++\ x) \Downarrow^t_{pin6} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ (l_2, \mu_2)) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x}) \Downarrow'_{pin2} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square, (\widehat{l}_2, \widehat{\mu}_2))$, $\Pi \cong_\psi \Sigma$, and $pin6 \cong pin2$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin7} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi},\ \text{bid}, \text{acc}, (l_2, \mu_2))$

This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin6} (\gamma,\ \sigma_1, \Delta_1,\ \overline{\chi},\ \text{bid}, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin4} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \overline{l}', \overline{j}, 1])$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ ++\ x) \Downarrow^t_{pin4} (\gamma,\ \sigma_1,\ \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ [\alpha, \overline{l}', \overline{j}, 1])$ by Location-tracking SMC^2

rule Pre-Increment Private Pointer Multiple Locations, we have $acc = 0$, $\gamma(x) = (l,\ \text{private}\ bty*)$, $\sigma(l) = (\omega,\ \text{private}\ bty*,\ \alpha,\ \text{PermL}(\text{Freeable},\ \text{private}\ bty*,\ \text{private},\ \alpha))$, $\text{DecodePtr}(\text{private}\ bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ 1]$, $\text{IncrementList}(\bar{l}, \tau(\text{private}\ bty),\ \sigma) = (\bar{l}', 1)$, and $\text{T_UpdatePtr}(\sigma,\ (l, 0),\ [\alpha, \bar{l}', \bar{j}, 1],\ \Delta,\ \overline{\chi},\ bid,\ acc,\ \text{private}\ bty*) = (\sigma_1, \Delta_1, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ bid,\ acc,\ ++x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $++\ x \cong_\psi ++\ \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(++x) = ++\ \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private}\ bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private}\ bty* \cong \widehat{bty}*$ by Lemma 3.2.14. By Definition 3.2.6, we have $\text{private}\ bty \cong bty$.

Given $\sigma(l) = (\omega, \text{private}\ bty*, \alpha, \text{PermL}(\text{Freeable},\ \text{private}\ bty*, \text{private}, \alpha))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private}\ bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ 1]$, $\text{private}\ bty* \cong \widehat{bty}*$, $\omega \cong_\psi \widehat{\omega}$, and $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ i],\ \text{private}\ bty*) = (l_1, \mu_1)$, by Lemma 3.2.45 we have $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ where $[\alpha, \bar{l}, \bar{j}, 1] \cong_\psi [1, (\widehat{l}_1, \widehat{\mu}_1), [1], 1]$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$.

Given $\text{IncrementList}(\bar{l}, \tau(\text{private}\ bty), \sigma) = (\bar{l}', 1)$, $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ 1],\ \text{private}\ bty*) = (l_1, \mu_1)$ such that $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $ty \cong \widehat{ty}$, and $\text{DeclassifyPtr}([\alpha,\ \bar{l}',\ \bar{j},\ 1],\ \text{private}\ bty*) = (l_2, \mu_2)$, by Lemma 3.2.51 we have $((\widehat{l}_2, \widehat{\mu}_2), 1) = \text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1),\ \tau(\widehat{bty}*))$ such that $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$. By Definition 3.2.14 we have $[\alpha, \bar{l}', \bar{j}, 1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1]$.

Given $\text{T_UpdatePtr}(\sigma, (l, 0), [\alpha, \bar{l}', \bar{j}, 1], \Delta, \overline{\chi}, bid, acc, \text{private}\ bty*) = (\sigma_1, \Delta_1, 1)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, 0) \cong_\psi (\widehat{l}, 0)$, $\text{private}\ bty* \cong \widehat{bty}*$, and $[\alpha, \bar{l}', \bar{j}, 1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1]$, by Lemma 4.2.6 we have $\text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_2, \widehat{\mu}_2)],\ [1],\ 1],\ \widehat{bty}*) = (\widehat{\sigma}_1, 1)$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x})$, $\widehat{\gamma}(x) = (\widehat{l},\ \widehat{bty}*)$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\ \widehat{bty}*,\ 1, \text{PermL}(\text{Freeable},\ \widehat{bty}*,\ \text{public},\ 1))$, $\text{DecodePtr}(\widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ 1]$, $\widehat{i} > 1$, $((\widehat{l}_2, \widehat{\mu}_2), 1) = \text{GetLocation}((\widehat{l}_1,\ \widehat{\mu}_1),\ \tau(\widehat{bty}*))$, and $\text{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, 0),\ [1,\ [(\widehat{l}_2, \widehat{\mu}_2)],\ [1],\ i],\ \widehat{bty}*) = (\widehat{\sigma}_1, 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x}) \Downarrow'_{pin3} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_2, \widehat{\mu}_2))$ by Vanilla C rule Pre-Increment Pointer Higher Level Indirection.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $[\alpha, \bar{l}', \bar{j}, 1] \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_2, \widehat{\mu}_2))$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ ++\ x) \Downarrow^t_{pin4} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ ++\ \widehat{x}) \Downarrow'_{pin3} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ (\widehat{l}_2, \widehat{\mu}_2))$, $\Pi \cong_\psi \Sigma$, and $pin4 \cong pin3$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin5} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}', \bar{j}, i])$

This case is similar to Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin4} (\gamma,\ \sigma_1, \Delta_1,\ \overline{\chi},\ \text{bid}, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1])$.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ ty\ x[e]) \Downarrow^t_{da} (\gamma_1, \sigma_3, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x[e]) \Downarrow^t_{da} (\gamma_1, \sigma_3,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Public 1 Dimension Array Declaration, we have $((ty = \text{public}\ bty) \wedge ((bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{int}))) \vee (ty = \text{char})$, $l = \phi()$, $\text{Label}(e, \gamma) = \text{public}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, $\gamma_1 = \gamma[x \rightarrow (l, \text{public const}\ bty*)]$, $l_1 = \phi()$, $\omega = \text{EncodePtr}(\text{public const}\ bty*, [1, [(l_1, 0)], [1], 1])$, $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{public const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public const}\ bty*, \text{public}, 1))]$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, $n > 0$, $\omega_1 = \text{EncodeVal}(\text{public}\ bty, \text{NULL})$, and $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{public}\ bty, n, \text{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, n))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ ty\ x[e]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}[\widehat{e}])$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x[e] \cong_\psi \widehat{ty}\ \widehat{x}[\widehat{e}]$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ ty\ x[e]) \Downarrow^t_{da} (\gamma_1, \sigma_3, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin ty\ x[e]$. Therefore, by Lemma 3.2.3 we have $ty\ x[e] \cong \widehat{ty}\ \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(ty\ x[e]) = \text{Erase}(ty)\ \text{Erase}(x[e])$, $\text{Erase}(ty) = \widehat{ty}$, $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and $c_1 \cong d_1$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $v = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $ty \cong \widehat{ty}$ and $((ty = \text{public}\ bty) \wedge ((bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{int}))) \vee (ty = \text{char})$, by

Definition 3.2.6 we have $bty \cong \widehat{bty}$. Therefore, we have public const $bty* \cong$ const $\widehat{bty}*$ and public $bty \cong \widehat{bty}$ by Definition 3.2.6.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{public const } bty*)]$, $x = \widehat{x}$, $l = \widehat{l}$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and public const $bty* \cong$ const $\widehat{bty}*$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{bty}*)]$ such that $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $l_1 = \phi()$, by Axiom 3.2.3 we have $\widehat{l}_1 = \phi()$ and $l_1 = \widehat{l}_1$.

Given $[1, [(l_1, 0)], [1], 1]$, by Definition 3.2.14 we have $[1,\ [(l_1,\ 0)],\ [1],\ 1] \cong_\psi [1,\ [(\widehat{l}_1, 0)],\ [1],\ 1]$. Given $\omega =$ EncodePtr(public const $bty*$, $[1, [(l_1, 0)], [1], 1]$) and public const $bty* \cong$ const $\widehat{bty}*$, by Lemma 3.2.42 we have $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{public const } bty*, 1,\ \text{PermL}(\text{Freeable},\ \text{public const } bty*, \text{public}, 1))]$, $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and public const $bty* \cong$ const $\widehat{bty}*$, by Lemma 3.2.35 we have $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega},$ const $\widehat{bty}*$, 1, PermL(Freeable, const $\widehat{bty}*$, public, 1))] such that $(\gamma_1,\ \sigma_2) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$.

Given $n > 0$ and $n = \widehat{n}$, we have $\widehat{n} > 0$.

Given $\omega_1 =$ EncodeVal(public bty, NULL), and public $bty \cong \widehat{bty}$, by Lemma 3.2.40 we have EncodeVal($\widehat{bty}$, NULL) $= \widehat{\omega}_1$ such that $\omega_1 \cong_\psi \widehat{\omega}_1$.

Given $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))]$, $l_1 = \widehat{l}_1$, $\omega_1 \cong_\psi \widehat{\omega}_1$, $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$, $n = \widehat{n}$, and public $bty \cong \widehat{bty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l}_1 \rightarrow (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty},$ public, $\widehat{n}))]$ such that $(\gamma_1, \sigma_3) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ x[\widehat{e}]), (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$, $\widehat{l} = \phi()$, $\widehat{l}_1 = \phi()$, EncodePtr(const $\widehat{bty}*$, $[1,\ [(\widehat{l}_1, 0)],$ $[1], 1]) = \widehat{\omega}$, $\widehat{\gamma}_1 = \widehat{\gamma}[x \rightarrow (\widehat{l},$ const $\widehat{bty}*)]$, $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega},$ const $\widehat{bty}*$, 1, PermL(Freeable, const $\widehat{bty}*$, public, 1))], EncodeVal($\widehat{bty}$, NULL) $= \widehat{\omega}_1$, $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l}_1 \rightarrow (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n},\ \text{PermL}(\text{Freeable},\ \widehat{bty},\ \text{public},\ \widehat{n}))]$, and $\widehat{n} > 0$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ x[\widehat{e}]) \Downarrow'_{da} (\widehat{\gamma}_1,\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ by Vanilla C rule 1D Array Declaration.

Given $(\gamma_1, \sigma_3) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma_1, \sigma_3,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x[e]) \Downarrow^t_{da} (\gamma_1, \sigma_3,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ x[\widehat{e}]) \Downarrow'_{da}$ $(\widehat{\gamma}_1,\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $da \cong da$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x[e]) \Downarrow^{t}_{da1} (\gamma_1, \sigma_3,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x[e]) \Downarrow^{t}_{da1} (\gamma_1, \sigma_3,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private 1 Dimension Array Declaration, we have $\text{Label}(e, \gamma) = \text{public}$, $((ty = \text{private}\ bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, $n > 0$, $l = \phi()$, $l_1 = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \text{private const}\ bty*)]$, $\omega = \text{EncodePtr}(\text{private const}\ bty*, [1, [(l_1, 0)], [1], 1])$, $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1))]$, $\omega_1 = \text{EncodeVal}(\text{private}\ bty, \text{NULL})$, and $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{private}\ bty, n, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x[e]) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}[\widehat{e}])$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x[e] \cong_{\psi} \widehat{ty}\ \widehat{x}[\widehat{e}]$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x[e]) \Downarrow^{t}_{da1} (\gamma_1, \sigma_3,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin ty\ x[e]$. Therefore, by Lemma 3.2.3 we have $ty\ x[e] \cong \widehat{ty}\ \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(ty\ x[e]) = \text{Erase}(ty)\ \text{Erase}(x[e])$, $\text{Erase}(ty) = \widehat{ty}$, $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $ty \cong \widehat{ty}$ and $((ty = \text{private}\ bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, by Definition 3.2.6 we have $bty \cong \widehat{bty}$. Therefore, we have $\text{private const}\ bty* \cong \text{const}\ \widehat{bty}*$ and $\text{private}\ bty \cong \widehat{bty}$ by Definition 3.2.6.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and $c_1 \cong d_1$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_{\psi} \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $n > 0$ and $n = \widehat{n}$, we have $\widehat{n} > 0$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $l_1 = \phi()$, by Axiom 3.2.3 we have $\widehat{l_1} = \phi()$ and $l_1 = \widehat{l_1}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{private const } bty*)]$, $x = \widehat{x}$, $l = \widehat{l}$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and private const $bty* \cong$ const $\widehat{bty}*$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \widehat{bty}*)]$ such that $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$.

Given $[1, [(l_1, 0)], [1], 1]$, by Definition 3.2.14 we have $[1, [(l_1, 0)], [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$. Given $\omega = \text{EncodePtr}(\text{private const } bty*, [1, [(l_1, 0)], [1], 1])$ and private const $bty* \cong$ const $\widehat{bty}*$, by Lemma 3.2.42 we have $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))]$, $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and private const $bty* \cong$ const $\widehat{bty}*$, by Lemma 3.2.35 we have $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))]$ such that $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_2)$.

Given $\omega_1 = \text{EncodeVal}(\text{private } bty, \text{NULL})$, and private $bty \cong \widehat{bty}$, by Lemma 3.2.40 we have $\text{EncodeVal}(\widehat{bty}, \text{NULL}) = \widehat{\omega}_1$ such that $\omega_1 \cong_\psi \widehat{\omega}_1$.

Given $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))]$, $l_1 = \widehat{l}_1$, $\omega_1 \cong_\psi \widehat{\omega}_1$, $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_2)$, $n = \widehat{n}$, and private $bty \cong \widehat{bty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l}_1 \rightarrow (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))]$ such that $(\gamma_1, \sigma_3) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ x[\widehat{e}])$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n})$, $\widehat{l} = \phi()$, $\widehat{l}_1 = \phi()$, $\text{EncodePtr}(\text{const } \widehat{bty}*, [1, [(\widehat{l}_1, 0)], [1], 1]) = \widehat{\omega}$, $\widehat{\gamma}_1 = \widehat{\gamma}[x \rightarrow (\widehat{l}, \text{const } \widehat{bty}*)]$, $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))]$, $\text{EncodeVal}(\widehat{bty}, \text{NULL}) = \widehat{\omega}_1$, $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l}_1 \rightarrow (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))]$, and $\widehat{n} > 0$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ x[\widehat{e}]) \Downarrow'_{da} (\widehat{\gamma}_1, \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Declaration.

Given $(\gamma_1, \sigma_3) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma_1, \sigma_3, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow^t_{da1} (\gamma_1, \sigma_3, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ x[\widehat{e}]) \Downarrow'_{da} (\widehat{\gamma}_1, \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $da1 \cong da$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow^t_{da2} (\gamma_1, \sigma_3, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow^t_{da2} (\gamma_1, \sigma_3, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$ by Location-tracking SMC^2 rule Private 1 Dimension Array Declaration (Inside a Private - Conditioned If Else Branch), we have $\text{Label}(e, \gamma) = \text{public}$, $((ty = \text{private } bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1}$

$(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, $(\text{acc} > 0) \wedge ((\text{bid} = \mathit{then}) \vee (\mathit{else}))$, $n > 0$, $l = \phi()$, $l_1 = \phi()$, $\overline{\chi}_1 = l :: l_1 :: \overline{\chi}[\text{acc}]$, $\gamma_1 = \gamma[x \rightarrow (l, \text{private const } \mathit{bty}*)]$, $\omega = \text{EncodePtr}(\text{private const } \mathit{bty}*, [1, [(l_1, 0)], [1], 1])$, $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{private const } \mathit{bty}*, 1, \text{PermL}(\text{Freeable}, \text{private const } \mathit{bty}*, \text{private}, 1))]$, $\omega_1 = \text{EncodeVal}(\text{private } \mathit{bty}, \text{NULL})$, and $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{private } \mathit{bty}, n, \text{PermL}(\text{Freeable}, \text{private } \mathit{bty}, \text{private}, n))]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x[e]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{ty}\ \widehat{x}[\widehat{e}])$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $ty\ x[e] \cong_\psi \widehat{ty}\ \widehat{x}[\widehat{e}]$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ ty\ x[e]) \Downarrow^t_{da1} (\gamma_1, \sigma_3,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin ty\ x[e]$. Therefore, by Lemma 3.2.3 we have $ty\ x[e] \cong \widehat{ty}\ \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(ty\ x[e]) = \text{Erase}(ty)\ \text{Erase}(x[e])$, $\text{Erase}(ty) = \widehat{ty}$, $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $ty \cong \widehat{ty}$ and $e \cong \widehat{e}$.

Given $ty \cong \widehat{ty}$ and $((ty = \text{private } \mathit{bty}) \vee (ty = \mathit{bty})) \wedge ((\mathit{bty} = \text{int}) \vee (\mathit{bty} = \text{float}))$, by Definition 3.2.6 we have $\mathit{bty} \cong \widehat{\mathit{bty}}$. Therefore, we have $\text{private const } \mathit{bty}* \cong \text{const } \widehat{\mathit{bty}}*$ and $\text{private } \mathit{bty} \cong \widehat{\mathit{bty}}$ by Definition 3.2.6.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ n) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{n})$ and $c_1 \cong d_1$. Given $n \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_\psi \widehat{n}$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}(n, \gamma) = \text{public}$ and therefore $n = \widehat{n}$ by Definition 3.2.18 and Definition 3.2.10.

Given $n > 0$ and $n = \widehat{n}$, we have $\widehat{n} > 0$.

Given $l = \phi()$, by Axiom 3.2.3 we have $\widehat{l} = \phi()$ and $l = \widehat{l}$.

Given $l_1 = \phi()$, by Axiom 3.2.3 we have $\widehat{l_1} = \phi()$ and $l_1 = \widehat{l_1}$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{private const } \mathit{bty}*)]$, $x = \widehat{x}$, $l = \widehat{l}$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $\text{private const } \mathit{bty}* \cong \text{const } \widehat{\mathit{bty}}*$, by Lemma 3.2.34 we have $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{\mathit{bty}}*)]$ such that $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given $[1, [(l_1, 0)], [1], 1]$, by Definition 3.2.14 we have $[1, [(l_1,\ 0)],\ [1],\ 1] \cong_\psi [1,\ [(\widehat{l}_1, 0)],\ [1],\ 1]$. Given $\omega = \text{EncodePtr}(\text{private const } \mathit{bty}*, [1,\ [(l_1, 0)],\ [1],\ 1])$ and $\text{private const } \mathit{bty}* \cong \text{const } \widehat{\mathit{bty}}*$, by Lemma 3.2.42 we have $\omega \cong_\psi \widehat{\omega}$.

Given $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))]$, $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_1)$, $l = \widehat{l}$, $\omega \cong_\psi \widehat{\omega}$, and private const $bty* \cong$ const $\widehat{bty}*$, by Lemma 3.2.35 we have $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))]$ such that $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_2)$.

Given $\omega_1 = \text{EncodeVal}(\text{private } bty, \text{NULL})$, and private $bty \cong \widehat{bty}$, by Lemma 3.2.40 we have $\text{EncodeVal}(\widehat{bty}, \text{NULL}) = \widehat{\omega}_1$ such that $\omega_1 \cong_\psi \widehat{\omega}_1$.

Given $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))]$, $l_1 = \widehat{l}_1$, $\omega_1 \cong_\psi \widehat{\omega}_1$, $(\gamma_1, \sigma_2) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_2)$, $n = \widehat{n}$, and private $bty \cong \widehat{bty}$, by Lemma 3.2.35 we have $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l}_1 \rightarrow (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))]$ such that $(\gamma_1, \sigma_3) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ x[\widehat{e}])$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{n})$, $\widehat{l} = \phi()$, $\widehat{l}_1 = \phi()$, $\text{EncodePtr}(\text{const } \widehat{bty}*, [1, [(\widehat{l}_1, 0)], [1], 1]) = \widehat{\omega}$, $\widehat{\gamma}_1 = \widehat{\gamma}[x \rightarrow (\widehat{l}, \text{const } \widehat{bty}*)]$, $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))]$, $\text{EncodeVal}(\widehat{bty}, \text{NULL}) = \widehat{\omega}_1$, $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l}_1 \rightarrow (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))]$, and $\widehat{n} > 0$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ x[\widehat{e}]) \Downarrow'_{da} (\widehat{\gamma}_1, \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Declaration.

Given $(\gamma_1, \sigma_3) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma_1, \sigma_3, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}_1, \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow^t_{da2} (\gamma_1, \sigma_3, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{ty}\ x[\widehat{e}]) \Downarrow'_{da} (\widehat{\gamma}_1, \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $da2 \cong da$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^t_{ra} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_i)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^t_{ra} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_i)$ by Location-tracking SMC^2 rule Public 1D Array Read Public Index, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, and $0 \leq i \leq n-1$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}])$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^t_{ra} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_i)$, by Lemma 4.2.2 we have $(l, \mu) \notin x[e]$. Therefore, by Lemma 3.2.3 we have $x[e] \cong \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega,\ \text{public const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{public const } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{public const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1,\ \text{public } bty, n, \text{PermL}(\text{Freeable},\ \text{public } bty,\ \text{public},\ n))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n},\ \text{PermL}(\text{Freeable},\ bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{public } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{public } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty,\ \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$. By Definition 3.2.10 we have $\forall m \in \{0, ..., n-1\}$, $v_m \cong_\psi \widehat{v}_m$. Therefore, given $i = \widehat{i}$, we have $v_i \cong_\psi \widehat{v}_{\widehat{i}}$.

Given $0 \leq i \leq n-1$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $0 \leq \hat{i} \leq \widehat{n} - 1$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, $\widehat{\gamma}(x) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{const } \widehat{bty}*,\ \text{public},\ 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ 0)],\ [1],\ 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n},\ \text{PermL}(\text{Freeable},\ \widehat{bty},\ \text{public}, \widehat{n}))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $0 \leq \widehat{i} \leq \widehat{n}-1$, and $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{ra} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v}_{\widehat{i}})$ by Vanilla C rule Array Read.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v_i \cong_\psi \widehat{v}_{\widehat{i}}$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_i) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v}_{\widehat{i}})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{ra} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_i) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{ra} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v}_{\widehat{i}})$, $\Pi \cong_\psi \Sigma$, and $ra \cong ra$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{ra3} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_i)$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{ra3} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_i)$ by Location-tracking SMC^2 rule Private 1D Array Read Public Index, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i)$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, and $0 \leq i \leq n-1$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}])$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{ra3} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_i)$, by Lemma 4.2.2 we have $(l, \mu) \notin x[e]$. Therefore, by Lemma 3.2.3 we have $x[e] \cong \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such

that $(l_1, 0) \cong_\psi (\widehat{l_1}, 0)$.

Given $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l_1}$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l_1}) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$. By Definition 3.2.10 we have $\forall m \in \{0, ..., n-1\}$, $v_m \cong_\psi \widehat{v}_m$. Therefore, given $i = \widehat{i}$, we have $v_i \cong_\psi \widehat{v}_{\widehat{i}}$.

Given $0 \leq i \leq n-1$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $0 \leq \hat{i} \leq \widehat{n} - 1$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, $\widehat{\gamma}(x) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{const } \widehat{bty}*,\ \text{public},\ 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l_1},\ 0)],\ [1],\ 1]$, $\widehat{\sigma}_1(\widehat{l_1}) = (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n},\ \text{PermL}(\text{Freeable},\ \widehat{bty},\ \text{public},\ \widehat{n}))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $0 \leq \widehat{i} \leq \widehat{n}-1$, and $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{ra} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v}_{\widehat{i}})$ by Vanilla C rule Array Read.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v_i \cong_\psi \widehat{v}_{\widehat{i}}$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v_i) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_{\widehat{i}})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e]) \Downarrow^t_{ra3} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v_i) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{ra} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_{\widehat{i}})$, $\Pi \cong_\psi \Sigma$, and $ra3 \cong ra$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e]) \Downarrow^t_{ra1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e]) \Downarrow^t_{ra1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$ by Location-tracking SMC^2 rule Private 1D Array Read Private Index we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, $\gamma(x) = (l, \text{private const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ [(l_1,\ 0)],\ [1],\ 1]$, $\sigma_1(l_1) = (\omega_1, \text{private } bty,\ n,\ \text{PermL}(\text{Freeable},\ \text{private } bty,\ \text{private},\ n))$, $\text{Label}(e,\ \gamma) = \text{private}$, $\text{DecodeVal}(\text{private } bty,\ n,\ \omega_1) = [v_0,\ ...,\ v_{n-1}]$, and $v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge v_m$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}])$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e]) \Downarrow^t_{ra1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, by

Lemma 4.2.2 we have $(l, \mu) \notin x[e]$. Therefore, by Lemma 3.2.3 we have $x[e] \cong \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$ by Definition 4.2.1. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable},\ bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$. By Axiom 3.2.1, we have $0 \leq \widehat{i} \leq \widehat{n} - 1$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge v_m$, by Axiom 3.2.1 and Lemma 3.2.9, we have $v \cong_\psi \widehat{v}_{\widehat{i}}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, $\widehat{\gamma}(x) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public},\ 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ 0)],\ [1],\ 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n},\ \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public},\ \widehat{n}))$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $0 \leq \widehat{i} \leq \widehat{n} - 1$, and $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{ra} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_{\widehat{i}})$ by Vanilla C rule Array Read.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}_{\widehat{i}}$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_{\widehat{i}})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{ra1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{ra} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v}_{\widehat{i}})$, $\Pi \cong_\psi \Sigma$, and $ra1 \cong ra$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{ra2} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v)$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{ra2} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v)$ by Location-tracking SMC^2 rule Public 1D Array Read Private Index, we have $\gamma(x) = (l,\ \text{public const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i)$, $\sigma_1(l) = (\omega, \text{public const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$, $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, and $v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge \text{encrypt}(v_m)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}])$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{ra2} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v)$, by Lemma 4.2.2 we have $(l, \mu) \notin x[e]$. Therefore, by Lemma 3.2.3 we have $x[e] \cong \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l,\ \text{public const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$.

Given $\sigma_1(l) = (\omega,\ \text{public const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1], 1]$, $\text{public const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$,

Lemma 3.2.44 we have $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1], 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{public } bty \cong \widehat{bty}$, and $n = \widehat{n}$. By Axiom 3.2.1, we have $0 \leq \widehat{i} \leq \widehat{n} - 1$.

Given $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{public } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $\gamma(x) = (l,\ \text{public const } bty*)$, by Definition 3.2.17 we have $[v_0, ..., v_{n-1}] = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$. Given $v = \bigvee_{m=0}^{n-1} (i = \text{encrypt}(m)) \wedge \text{encrypt}(v_m)$, by Axiom 3.2.1 and Lemma 3.2.10 we have $v \cong_\psi \widehat{v}_{\widehat{i}}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, $\widehat{\gamma}(x) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1,\ [(\widehat{l}_1,\ 0)],\ [1],\ 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty},\ \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$, $0 \leq \widehat{i} \leq \widehat{n}-1$, and $\text{DecodeVal}(\widehat{bty},\ \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{ra} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_{\widehat{i}})$ by Vanilla C rule Array Read.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}_{\widehat{i}}$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v}_{\widehat{i}})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{ra2} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{ra} (\widehat{\gamma},\ \widehat{\sigma}_1, \square, \widehat{v}_{\widehat{i}})$, $\Pi \cong_\psi \Sigma$, and $ra2 \cong ra$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Public 1D Array Write Public Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i)$, $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_2(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$, $\sigma_2(l_1) = (\omega_1, \text{public } bty,\ n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}](\frac{v}{v_i})$, $0 \leq i \leq n - 1$, and $\text{T_UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty) = (\sigma_3, \Delta_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e_1]\ = e_2 \cong_\psi \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2)$ $\Downarrow^t_{wa}$ $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x[e_1]\ = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1]\ = e_2 \cong \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. By Definition 3.2.10 we have $\text{Erase}(x[e_1]\ = e_2 = \text{Erase}(x[e_1]) = \text{Erase}(e_2)$, $\text{Erase}(x[e_1]) = \widehat{x}[\text{Erase}(e_1)]$ where $x = \widehat{x}$, $\text{Erase}(e_1) = \widehat{e}_1$, and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \cong_\psi (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)$. Given $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and ψ_2 such that $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and $c_2 \cong d_2$. Given $v \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \text{public const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{public const } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{public const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by

Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, public $bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, public $bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{v}{v_i}\right)$, $v \cong_\psi \widehat{v}$, $i = \widehat{i}$, and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, by Lemma 3.2.63 we have $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\hat{i}}}\right)$ such that $[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}]$.

Given $0 \leq i \leq n-1$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $0 \leq \hat{i} \leq \widehat{n} - 1$.

Given $\text{T_UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n_e-1}], \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty) = (\sigma_3, \Delta_3)$, $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $l_1 = \widehat{l}_1$, public $bty \cong \widehat{bty}$, and $[v'_0, ..., v'_{n_e-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}_e-1}]$, by Lemma 4.2.4 we have $\text{UpdateVal}(\widehat{\sigma}_2, \widehat{l}_1, [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3$ such that $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \hat{i})$, $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, $0 \leq \hat{i} \leq \widehat{n} - 1$, $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\hat{i}}}\right)$, and $\text{UpdateVal}(\widehat{\sigma}_2, \widehat{l}_1, [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \Downarrow'_{wa} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Write.

Given $(\gamma, \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3, \square, \text{skip})$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wa} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \Downarrow'_{wa} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wa \cong wa$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wa4} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wa4} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by Location-tracking SMC^2 rule Private 1D Array Write Private Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{public}$, $\text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private } bty,$

n, PermL(Freeable, private bty, private, n)), DecodeVal(private bty, n, ω_1) $= [v_0, ..., v_{n-1}]$, $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}](\frac{v}{v_i})$ $0 \leq i \leq n-1$, and T_UpdateVal($\sigma_2, l_1, [v'_0, ..., v'_{n-1}]$, Δ_2, $\overline{\chi}$, bid, acc, private bty) $= (\sigma_3, \Delta_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $x[e_1]\ = e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e_1]\ = e_2 \cong_\psi \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $x[e_1]\ = e_2)$ $\Downarrow^t_{wa4}$ $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},$ bid, acc, skip), by Lemma 4.2.2 we have $(l, \mu) \notin x[e_1]\ = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1]\ = e_2 \cong \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. By Definition 3.2.10 we have Erase($x[e_1]\ = e_2$ = Erase($x[e_1]$) = Erase(e_2), Erase($x[e_1]$) $= \widehat{x}$[Erase(e_1)] where $x = \widehat{x}$, Erase(e_1) $= \widehat{e}_1$, and Erase(e_2) $= \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}_1) \cong_\psi (\gamma,\ \sigma,\ \Delta,$ $\overline{\chi}$, bid, acc, $e_1)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},$ bid, acc, $i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \Box,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},$ bid, acc, $i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq$ skip, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given Label(e_1, γ) = public, we have Label(i, γ) = public and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{e}_2) \cong_\psi (\gamma,\ \sigma_1,$ $\Delta_1, \overline{\chi}$, bid, acc, $e_2)$. Given $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}$, bid, acc, $e_2)$ $\Downarrow^t_{c_2}$ $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}$, bid, acc, $v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ \widehat{v})$ and ψ_2 such that $(\gamma,\ \sigma_2,$ acc, $v) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\ \widehat{v})$ and $c_2 \cong d_2$. Given $v \neq$ skip, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq$ skip and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq$ skip.

Given $\gamma(x) = (l,$ private const $bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l},$ const $\widehat{bty}*)$ such that $l = \widehat{l}$ by private const $bty* \cong$ const $\widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,$ private const $bty*,$ 1, PermL(Freeable, private const $bty*$, private, 1)), $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega},$ const $\widehat{bty}*$, 1, PermL(Freeable, const $\widehat{bty}*$, public, 1)) such that $\omega \cong_\psi \widehat{\omega}$.

Given DecodePtr(private const $bty*$, 1, ω) $= [1,\ (l_1, 0),\ [1],\ 1]$, private const $bty* \cong$ const $\widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have DecodePtr($\widehat{bty}*, 1, \widehat{\omega}$) $= [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{v}{v_i}\right)$, $v \cong_\psi \widehat{v}$, $i = \widehat{i}$, and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, by Lemma 3.2.63 we have $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\hat{i}}}\right)$ such that $[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}]$.

Given $0 \leq i \leq n-1$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $0 \leq \hat{i} \leq \widehat{n} - 1$.

Given $\text{T_UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n_e-1}], \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{ private } bty) = (\sigma_3, \Delta_3)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $l_1 = \widehat{l}_1$, $\text{private } bty \cong \widehat{bty}$, and $[v'_0, ..., v'_{n_e-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}_e-1}]$, by Lemma 4.2.4 we have $\text{UpdateVal}(\widehat{\sigma}_2,\ \widehat{l}_1,\ [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \hat{i})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, $0 \leq \hat{i} \leq \widehat{n} - 1$, $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\hat{i}}}\right)$, and $\text{UpdateVal}(\widehat{\sigma}_2,\ \widehat{l}_1,\ [\widehat{v}'_0,\ ...,\ \widehat{v}'_{\widehat{n}-1}], \widehat{bty}) = \widehat{\sigma}_3$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{wa} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ by Vanilla C rule 1D Array Write.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa4} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{wa} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wa4 \cong wa$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa1} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa1} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private 1D Array Write Public Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$,

$v \neq \text{skip}$, $\gamma(x) = (l,\ \text{private const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}](\frac{\text{encrypt}(v)}{v_i})$, $0 \leq i \leq n-1$, and $\text{T_UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \Delta_2, \overline{\chi}, \text{bid}, \text{private } bty) = (\sigma_3, \Delta_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma,\ \text{acc},\ \Delta,\ \overline{\chi},\ \text{bid},\ x[e_1]\ = e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e_1]\ = e_2 \cong_\psi \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa1} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x[e_1]\ = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1]\ = e_2 \cong \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. By Definition 3.2.10 we have $\text{Erase}(x[e_1]\ = e_2 = \text{Erase}(x[e_1]) = \text{Erase}(e_2)$, $\text{Erase}(x[e_1]) = \widehat{x}[\text{Erase}(e_1)]$ where $x = \widehat{x}$, $\text{Erase}(e_1) = \widehat{e}_1$, and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \cong_\psi (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)$. Given $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and ψ_2 such that $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and $c_2 \cong d_2$. Given $v \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(v, \gamma) = \text{public}$ and therefore $v = \widehat{v}$ by Definition 3.2.17.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given DecodePtr(private const $bty*$, 1, ω) = [1, (l_1, 0), [1], 1], private const $bty*$ $\cong$ const $\widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have DecodePtr($\widehat{bty}*, 1, \widehat{\omega}$) = [1, ($\widehat{l}_1$, 0), [1], 1] where [1, (l_1, 0), [1], 1] $\cong_\psi$ [1, ($\widehat{l}_1$, 0), [1], 1] such that (l_1, 0) $\cong_\psi$ ($\widehat{l}_1$, 0).

Given $\sigma_2(l_1)$ = (ω_1, private bty, n, PermL(Freeable, private bty, private, n)), (γ, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$), and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1)$ = ($\widehat{\omega}_1$, $\widehat{bty}$, $\widehat{n}$, PermL(Freeable, bty, public, $\widehat{n}$)) where $\omega_1 \cong_\psi \widehat{\omega}_1$, private bty $\cong$ $\widehat{bty}$, and $n = \widehat{n}$.

Given DecodeVal(private bty, n, ω_1) = [v_0, ..., v_{n-1}], private bty $\cong$ $\widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have DecodeVal(bty, $\widehat{n}$, $\widehat{\omega}_1$) = [$\widehat{v}_0$, ..., $\widehat{v}_{n-1}$] and [v_0, ..., v_{n-1}] $\cong_\psi$ [$\widehat{v}_0$, ..., $\widehat{v}_{n-1}$].

Given [v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]$\left(\frac{\text{encrypt}(v)}{v_i}\right)$ and $v = \widehat{v}$, by Definition 3.2.18 and Definition 3.2.10 we have encrypt(v) $\cong_\psi$ $\widehat{v}$. Given $i = \widehat{i}$ and [v_0, ..., v_{n-1}] $\cong_\psi$ [$\widehat{v}_0$, ..., $\widehat{v}_{n-1}$], by Lemma 3.2.63 we have [$\widehat{v}'_0$, ..., $\widehat{v}'_{\widehat{n}-1}$] = [$\widehat{v}_0$, ..., $\widehat{v}_{\widehat{n}-1}$]$\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$ such that [v'_0, ..., v'_{n-1}] $\cong_\psi$ [$\widehat{v}'_0$, ..., $\widehat{v}'_{\widehat{n}-1}$].

Given $0 \leq i \leq n-1$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $0 \leq \hat{i} \leq \widehat{n} - 1$.

Given T_UpdateVal(σ_2, l_1, [v'_0, ..., v'_{n_e-1}], Δ_2, $\overline{\chi}$, bid, acc, private bty) = (σ_3, Δ_3), (γ, σ_2) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$), $l_1 = \widehat{l}_1$, private bty $\cong$ $\widehat{bty}$, and [v'_0, ..., v'_{n_e-1}] $\cong_\psi$ [$\widehat{v}'_0$, ..., $\widehat{v}'_{\widehat{n}_e-1}$], by Lemma 4.2.4 we have UpdateVal($\widehat{\sigma}_2$, $\widehat{l}_1$, [$\widehat{v}'_0$, ..., $\widehat{v}'_{\widehat{n}-1}$], $\widehat{bty}$) = $\widehat{\sigma}_3$ such that (γ, σ_3) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_3$).

Given ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{x}[\widehat{e}_1] = \widehat{e}_2$), ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{e}_1$) $\Downarrow'_{d_1}$ ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\hat{i}$), ($\widehat{\gamma}$, $\widehat{\sigma}_1$, $\square$, $\widehat{e}_2$) $\Downarrow'_{d_2}$ ($\widehat{\gamma}$, $\widehat{\sigma}_2$, $\square$, $\widehat{v}$), $\widehat{v} \neq$ skip, $\widehat{\gamma}(\widehat{x})$ = ($\widehat{l}$, const $\widehat{bty}*$), $\widehat{\sigma}_2(\widehat{l})$ = ($\widehat{\omega}$, const $\widehat{bty}*$, 1, PermL(Freeable, const $\widehat{bty}*$, public, 1)), DecodePtr(const $\widehat{bty}*$, 1, $\widehat{\omega}$) = [1, [($\widehat{l}_1$, 0)], [1], 1], $\widehat{\sigma}_2(\widehat{l}_1)$ = ($\widehat{\omega}_1$, $\widehat{bty}$, $\widehat{n}$, PermL(Freeable, $\widehat{bty}$, public, $\widehat{n}$)), DecodeVal($\widehat{bty}$, $\widehat{n}$, $\widehat{\omega}_1$) = [$\widehat{v}_0$, ..., $\widehat{v}_{\widehat{n}-1}$], $0 \leq \hat{i} \leq \widehat{n} - 1$, [$\widehat{v}'_0$, ..., $\widehat{v}'_{\widehat{n}-1}$] = [$\widehat{v}_0$, ..., $\widehat{v}_{\widehat{n}-1}$]$\left(\frac{\widehat{v}}{\widehat{v}_{\hat{i}}}\right)$, and UpdateVal($\widehat{\sigma}_2$, $\widehat{l}_1$, [$\widehat{v}'_0$, ..., $\widehat{v}'_{\widehat{n}-1}$], $\widehat{bty}$) = $\widehat{\sigma}_3$, we have $\Sigma \triangleright$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{x}[\widehat{e}_1] = \widehat{e}_2$) $\Downarrow'_{wa}$ ($\widehat{\gamma}$, $\widehat{\sigma}_3$, $\square$, skip) by Vanilla C rule 1D Array Write.

Given (γ, σ_3) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_3$), by Definition 4.2.1 we have (γ, σ_3, Δ_3, $\overline{\chi}$, bid, acc, skip) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}_3$, $\square$, skip). Therefore, we have (γ, σ, Δ, $\overline{\chi}$, bid, acc, $x[e_1] = e_2$) $\Downarrow^t_{wa1}$ (γ, σ_3, Δ_3, $\overline{\chi}$, bid, acc, skip) $\cong_\psi$ ($\widehat{\gamma}$, $\widehat{\sigma}$, $\square$, $\widehat{x}[\widehat{e}_1] = \widehat{e}_2$) $\Downarrow'_{wa}$ ($\widehat{\gamma}$, $\widehat{\sigma}_3$, $\square$, skip), $\Pi \cong_\psi \Sigma$, and $wa1 \cong wa$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa2} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa2} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private 1D Array Write Public Value Private Index, we have $\text{Label}(e_1, \gamma) = \text{private}$, $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i)$, $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1,\ 0)],\ [1],\ 1]$, $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $(bty = \text{int}) \vee (bty = \text{float})$, $v' = \text{encrypt}(v)$, $\forall v_m \in [v_0,\ ...,\ v_{n-1}].\ v'_m = ((i = \text{encrypt}(m)) \wedge v') \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, and $\text{T_UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{private } bty) = (\sigma_3, \Delta_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma, \text{acc}, x[e_1]\ = e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e_1]\ = e_2 \cong_\psi \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa2} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x[e_1]\ = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1]\ = e_2 \cong \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. By Definition 3.2.10 we have $\text{Erase}(x[e_1]\ = e_2 = \text{Erase}(x[e_1]) = \text{Erase}(e_2)$, $\text{Erase}(x[e_1]) = \widehat{x}[\text{Erase}(e_1)]$ where $x = \widehat{x}$, $\text{Erase}(e_1) = \widehat{e}_1$, and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \cong_\psi (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \cong_\psi (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2)$. Given $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and ψ_2 such that $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and $c_2 \cong d_2$. Given $v \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$. Given $\text{Label}(e_2, \gamma) = \text{public}$, we have $\text{Label}(v, \gamma) = \text{public}$ and therefore $v = \widehat{v}$ by Definition 3.2.17.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$. By Axiom 3.2.1, we have $0 \leq \widehat{i} \leq \widehat{n} - 1$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$. By Axiom 3.2.1, we have $0 \leq \widehat{i} \leq \widehat{n} - 1$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $v' = \text{encrypt}(v)$ and $v = \widehat{v}$, by Definition 3.2.17 we have $v' \cong_\psi \widehat{v}$.

Given $\forall v_m \in [v_0,\ ...,\ v_{n-1}].\ v'_m = ((i = \text{encrypt}(m)) \wedge v') \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, by Axiom 3.2.1 and Lemma 3.2.12, we have $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$ such that $[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}]$.

Given $\text{T_UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n_e-1}],\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{private } bty) = (\sigma_3,\ \Delta_3)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $l_1 = \widehat{l}_1$, $\text{private } bty \cong \widehat{bty}$ and $[v'_0, ..., v'_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Lemma 4.2.4 we have $\text{UpdateVal}(\widehat{\sigma}_2,\ \widehat{l}_1,\ [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}],\ \widehat{bty}) = \widehat{\sigma}_3$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, $0 \leq \widehat{i} \leq \widehat{n} - 1$, $[\widehat{v}'_0,\ ...,\ \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$, and $\text{UpdateVal}(\widehat{\sigma}_2,\ \widehat{l}_1,\ [\widehat{v}'_0, ...,\ \widehat{v}'_{\widehat{n}-1}],\ \widehat{bty}) = \widehat{\sigma}_3$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{wa} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Write.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa2} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]$

$= \widehat{e}_2) \Downarrow'_{wa} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wa2 \cong wa$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa3} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa3} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private 1D Array Write Private Value Private Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, $v \neq \text{skip}$, $\gamma(x) = (l,\ \text{private const}\ bty*)$, $\sigma_2(l) = (\omega,\ \text{private const}\ bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const}\ bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private}\ bty, n, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n))$, $\text{DecodeVal}(\text{private}\ bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $(bty = \text{int}) \vee (bty = \text{float})$, $\forall v_m \in [v_0, ..., v_{n-1}].\ v'_m = ((i = \text{encrypt}(m)) \wedge v) \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, and $\text{T_UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{private}\ bty) = (\sigma_3, \Delta_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e_1]\ = e_2 \cong_\psi \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa3} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x[e_1]\ = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1]\ = e_2 \cong \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. By Definition 3.2.10 we have $\text{Erase}(x[e_1]\ = e_2 = \text{Erase}(x[e_1]) = \text{Erase}(e_2)$, $\text{Erase}(x[e_1]) = \widehat{x}[\text{Erase}(e_1)]$ where $x = \widehat{x}$, $\text{Erase}(e_1) = \widehat{e}_1$, and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \cong_\psi (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)$. Given $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and ψ_2 such that $(\gamma,\ \sigma_2, \text{acc},\ v) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and $c_2 \cong d_2$. Given $v \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l,\ \text{private const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$. By Axiom 3.2.1, we have $0 \leq \widehat{i} \leq \widehat{n} - 1$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $\forall v_m \in [v_0, ..., v_{n-1}].\ v'_m = ((i = \text{encrypt}(m)) \wedge v) \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, $[v_0,\ ...,\ v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, and $i \cong_\psi \widehat{i}$, by Axiom 3.2.1 and Lemma 3.2.11 we have $[v'_0, ..., v'_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$.

Given $\text{T_UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n_e-1}], \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ \text{private } bty) = (\sigma_3, \Delta_3)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $l_1 = \widehat{l}_1$, $\text{private } bty \cong \widehat{bty}$, and $[v'_0, ..., v'_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Lemma 4.2.4 we have $\text{UpdateVal}(\widehat{\sigma}_2,\ \widehat{l}_1,\ [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}],\ \widehat{bty}) = \widehat{\sigma}_3$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)\ \Downarrow'_{d_1}\ (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)\ \Downarrow'_{d_2}\ (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*,\ 1,\ \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*,\ \text{public},\ 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1,\ [(\widehat{l}_1, 0)],\ [1],\ 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $\text{DecodeVal}(\widehat{bty}, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]$, $0 \leq \widehat{i} \leq \widehat{n} - 1$, $[\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}-1}]\left(\frac{\widehat{v}}{\widehat{v}_{\widehat{i}}}\right)$, and $\text{UpdateVal}(\widehat{\sigma}_2,\ \widehat{l}_1,\ [\widehat{v}'_0, ..., \widehat{v}'_{\widehat{n}-1}],\ \widehat{bty}) = \widehat{\sigma}_3$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{wa} (\widehat{\gamma}, \widehat{\sigma}_3, \square, \text{skip})$ by Vanilla C rule 1D Array Write.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wa3} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2) \Downarrow'_{wa}$

$(\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wa3 \cong wa$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e]) \Downarrow^t_{rao} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e]) \Downarrow^t_{rao} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$ by Location-tracking SMC^2 rule Public 1D Array Read Out of Bounds Public Index, we have $\text{Label}(e, \gamma) = \text{public}$, $\gamma(x) = (l,\ \text{public const } bty*)$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i)$, $\sigma_1(l) = (\omega,\ \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(i < 0) \vee (i \geq n)$, and $\text{ReadOOB}(i,\ n,\ l_1,\ \text{public } bty,\ \sigma_1) = (v,\ 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}])$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e]) \Downarrow^t_{rao} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, by Lemma 4.2.2 we have $(l, \mu) \notin x[e]$. Therefore, by Lemma 3.2.3 we have $x[e] \cong \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l,\ \text{public const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $\sigma_1(l) = (\omega,\ \text{public const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{public const } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{public const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{public } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $(i < 0) \vee (i \geq n)$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$.

Given $\text{ReadOOB}(i,\ n,\ l_1,\ \text{public } bty,\ \sigma_1) = (v,\ 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $i = \widehat{i}$, $n = \widehat{n}$, $l = \widehat{l}$, and $\text{public } bty \cong \widehat{bty}$, by Lemma 3.2.62 we have $\text{ReadOOB}(\widehat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty},\ \widehat{\sigma}_1) = (\widehat{v},\ 1)$ such that $v \cong_\psi \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1, [(\widehat{l}_1, 0)],\ [1],\ 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}$ public, $\widehat{n}))$, $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$, and $\text{ReadOOB}(\widehat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty},\ \widehat{\sigma}_1) = (\widehat{v},\ 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{rao} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ by Vanilla C rule 1D Array Read Out of Bounds.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{rao} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{rao} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\Pi \cong_\psi \Sigma$, and $rao \cong rao$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{rao1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v)$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{rao1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v)$ by Location-tracking SMC^2 rule Private 1D Array Read Out of Bounds Public Index, we have $\text{Label}(e, \gamma) = \text{public}$, $\gamma(x) = (l,\ \text{private const } bty*)$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, $\sigma_1(l) = (\omega,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1,\ 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1,\ \text{private } bty,\ n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(i < 0) \vee (i \geq n)$, and $\text{ReadOOB}(i,\ n,\ l_1, \text{private } bty,\ \sigma_1) = (v,\ 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}])$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}])$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e]) \Downarrow^t_{rao1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ v)$, by Lemma 4.2.2 we have $(l, \mu) \notin x[e]$. Therefore, by Lemma 3.2.3 we have $x[e] \cong \widehat{x}[\widehat{e}]$. By Definition 3.2.10 we have $\text{Erase}(x[e]) = \widehat{x}[\text{Erase}(e)]$ where $x = \widehat{x}$ and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $\gamma(x) = (l,\ \text{private const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$

by private const $bty*$ $\cong$ const $\widehat{bty}*$ by Lemma 3.2.14.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},$ bid, acc, $i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},$ bid, acc, $i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq$ skip, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\mathrm{Label}(e_1, \gamma) =$ public, we have $\mathrm{Label}(i, \gamma) =$ public and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $\sigma_1(l) = (\omega,$ private const $bty*,\ 1,$ PermL(Freeable, private const $bty*$, private, 1)), $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty}*, \mathrm{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given DecodePtr(private const $bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, private const $bty* \cong$ const $\widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\mathrm{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1,$ private $bty, n,$ PermL(Freeable, private bty, private, n)), $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \mathrm{PermL}(\mathrm{Freeable}, bty, \mathrm{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, private $bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $(i < 0) \vee (i \geq n)$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $(\hat{i} < 0) \vee (\hat{i} \geq \widehat{n})$.

Given ReadOOB($i,\ n,\ l_1,$ private $bty,\ \sigma_1) = (v,\ 1)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, $i = \widehat{i}$, $n = \widehat{n}$, $l = \widehat{l}$, and private $bty \cong \widehat{bty}$, by Lemma 3.2.62 we have $\mathrm{ReadOOB}(\hat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty},\ \widehat{\sigma}_1) = (\widehat{v}, 1)$ such that $v \cong_\psi \widehat{v}$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \hat{i})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\ \mathrm{const}\ \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\ \mathrm{const}\ \widehat{bty}*, 1,$ $\mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty}*, \mathrm{public}, 1))$, $\mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty}*,\ 1,\ \widehat{\omega}) = [1, [(\widehat{l}_1, 0)],\ [1],\ 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1,\ \widehat{bty}, \widehat{n},$ $\mathrm{PermL}(\mathrm{Freeable},\ \widehat{bty},\ \mathrm{public},\ \widehat{n}))$, $(\hat{i} < 0) \vee (\hat{i} \geq \widehat{n})$, and $\mathrm{ReadOOB}(\hat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty},\ \widehat{\sigma}_1) = (\widehat{v},\ 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{rao} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$ by Vanilla C rule 1D Array Read Out of Bounds.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $v \cong_\psi \widehat{v}$, by Definition 4.2.1 we have $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},$ bid, acc, $v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{v})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},$ bid, acc, $x[e]) \Downarrow^t_{rao1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},$ bid, acc, $v) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{x}) \Downarrow'_{rao}$ $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{v})$, $\Pi \cong_\psi \Sigma$, and $rao1 \cong rao$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wao} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e_1]\ = e_2)\ \Downarrow^t_{wao} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Public 1D Array Write Out of Bounds Public Index Public Value, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i)$, $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2) \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_2(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(i < 0) \vee (i \geq n)$, and $\text{T_WriteOOB}(v, i, n, l_1, \text{public } bty, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_3, \Delta_3, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e_1]\ = e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e_1]\ = e_2 \cong_\psi \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wao} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x[e_1]\ = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1]\ = e_2 \cong \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. By Definition 3.2.10 we have $\text{Erase}(x[e_1]\ = e_2 = \text{Erase}(x[e_1]) = \text{Erase}(e_2)$, $\text{Erase}(x[e_1]) = \widehat{x}[\text{Erase}(e_1)]$ where $x = \widehat{x}$, $\text{Erase}(e_1) = \widehat{e}_1$, and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \cong (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1)$. Given $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ e_1) \Downarrow^t_{c_1} (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \cong_\psi (\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2)$. Given $(\gamma,\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc},\ e_2)\ \Downarrow^t_{c_2} (\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and ψ_2 such that $(\gamma,\ \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc},\ v) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and $c_2 \cong d_2$. Given $v \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \text{public const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{public const } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{public const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable},\ \text{public } bty, \text{public}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{public } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $(i < 0) \vee (i \geq n)$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$.

Given $\text{T_WriteOOB}(v,\ i,\ n,\ l_1,\ \text{public } bty,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc}) = (\sigma_3, \Delta_3, 1)$, $v \cong_\psi \widehat{v}$, $i = \widehat{i}$, $n = \widehat{n}$, $l_1 = \widehat{l}_1$, $\text{public } bty \cong \widehat{bty}$, and $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, by Lemma 4.2.9 we have $\text{WriteOOB}(\widehat{v},\ \widehat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty},\ \widehat{\sigma}_2) = (\widehat{\sigma}_3,\ 1)$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(x) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$, and $\text{WriteOOB}(\widehat{v},\ \widehat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty},\ \widehat{\sigma}_2) = (\widehat{\sigma}_3,\ 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \Downarrow'_{wao} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ by Vanilla C rule 1D Array Write Out of Bounds.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1] = e_2) \Downarrow^t_{wao} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \Downarrow'_{wao} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wao \cong wao$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1] = e_2) \Downarrow^t_{wao2} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2)\ \Downarrow^t_{wao2}\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private 1D Array Write Out of Bounds Public Index Public Value, we have $\text{Label}(e_1, \gamma) = \text{public}$, $\text{Label}(e_2, \gamma) = \text{private}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow^t_{c_1}\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow^t_{c_2}\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_2(l) = (\omega,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{private const } bty*,\ \text{private},\ 1))$, $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ [(l_1,\ 0)],\ [1],\ 1]$, $\sigma_2(l_1) = (\omega_1,\ \text{private } bty,\ n,\ \text{PermL}(\text{Freeable},\ \text{private } bty,\ \text{private},\ n))$, $(i\ <\ 0) \vee (i\ \geq\ n)$, and $\text{T_WriteOOB}(v, i, n, l_1, \text{private } bty, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_3, \Delta_3, 1)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x[e_1]\ = e_2 \cong_\psi \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2)\ \Downarrow^t_{wao2}\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x[e_1]\ = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1]\ = e_2 \cong \widehat{x}[\widehat{e}_1]\ = \widehat{e}_2$. By Definition 3.2.10 we have $\text{Erase}(x[e_1]\ = e_2 = \text{Erase}(x[e_1]) = \text{Erase}(e_2)$, $\text{Erase}(x[e_1]) = \widehat{x}[\text{Erase}(e_1)]$ where $x = \widehat{x}$, $\text{Erase}(e_1) = \widehat{e}_1$, and $\text{Erase}(e_2) = \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow^t_{c_1}\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)\ \Downarrow'_{d_1}\ (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given $\text{Label}(e_1, \gamma) = \text{public}$, we have $\text{Label}(i, \gamma) = \text{public}$ and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)$ such that $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2) \cong_\psi (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)$. Given $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow^t_{c_2}\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)\ \Downarrow'_{d_2}\ (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and ψ_2 such that $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$ and $c_2 \cong d_2$. Given $v \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$.

Given $v \neq \text{skip}$ and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty*})$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty*}$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$,

and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $(i < 0) \vee (i \geq n)$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$.

Given $\text{T_WriteOOB}(v,\ i,\ n,\ l_1,\ \text{private } bty,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc}) = (\sigma_3, \Delta_3, 1)$, $v \cong_\psi \widehat{v}$, $i = \widehat{i}$, $n = \widehat{n}$, $l_1 = \widehat{l}_1$, $\text{private } bty \cong \widehat{bty}$, and $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, by Lemma 4.2.9 we have $\text{WriteOOB}(\widehat{v},\ \widehat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty},\ \widehat{\sigma}_2) = (\widehat{\sigma}_3,\ 1)$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ =\ \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1)\ \Downarrow'_{d_1}\ (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{e}_2)\ \Downarrow'_{d_2}\ (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, $\widehat{v} \neq \text{skip}$, $\widehat{\gamma}(x) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$, and $\text{WriteOOB}(\widehat{v},\ \widehat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty},\ \widehat{\sigma}_2) = (\widehat{\sigma}_3,\ 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1]\ =\ \widehat{e}_2) \Downarrow'_{wao} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ by Vanilla C rule 1D Array Write Out of Bounds.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2)\ \Downarrow^t_{wao2}\ (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \widehat{x}[\widehat{e}_1]\ =\ \widehat{e}_2) \Downarrow'_{wao} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wao2 \cong wao$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wao1} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x[e_1]\ = e_2) \Downarrow^t_{wao1} (\gamma,\ \sigma_3,\ \Delta_3,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Private 1D Array Write Public Value Out of Bounds Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_1)\ \Downarrow^t_{c_1}\ (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ i)$, $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e_2)\ \Downarrow^t_{c_2}\ (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ v)$, $v \neq \text{skip}$, $\gamma(x) = (l,\ \text{private const } bty*)$, $\sigma_2(l) = (\omega,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable},$

private const $bty*$, private, 1)), DecodePtr(private const $bty*$, 1, ω) $=$ [1, [(l_1, 0)], [1], 1], $\sigma_2(l_1) = (\omega_1$, private bty, n, PermL(Freeable, private bty, private, n)), $(i < 0) \vee (i \geq n)$, and T_WriteOOB(encrypt(v), i, n, l_1, private $bty, \sigma_2, \Delta_2, \overline{\chi}$, bid, acc) $= (\sigma_3, \Delta_3, 1)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$ and ψ such that $(\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $x[e_1] = e_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x[e_1] = e_2 \cong_\psi \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. Given $(\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $x[e_1] = e_2)$ $\Downarrow^t_{wao1}$ $(\gamma, \sigma_3, \Delta_3, \overline{\chi}$, bid, acc, skip), by Lemma 4.2.2 we have $(l, \mu) \notin x[e_1] = e_2$. Therefore, by Lemma 3.2.3 we have $x[e_1] = e_2 \cong \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. By Definition 3.2.10 we have Erase($x[e_1] = e_2$ = Erase($x[e_1]$) = Erase(e_2), Erase($x[e_1]$) $= \widehat{x}$[Erase(e_1)] where $x = \widehat{x}$, Erase(e_1) $= \widehat{e}_1$, and Erase(e_2) $= \widehat{e}_2$. Therefore, we have $e_1 \cong \widehat{e}_1$ and $e_2 \cong \widehat{e}_2$.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $e_1 \cong \widehat{e}_1$, by Lemma 4.2.3 we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1)$ such that $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \cong_\psi (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $e_1)$. Given $(\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}$, bid, acc, $i)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and ψ_1 such that $(\gamma, \sigma_1, \Delta_1, \overline{\chi}$, bid, acc, $i) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{i})$ and $c_1 \cong d_1$. Given $i \neq$ skip, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $i \cong_\psi \widehat{i}$. Given Label(e_1, γ) = public, we have Label(i, γ) = public and therefore $i = \widehat{i}$ by Definition 3.2.17.

Given $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $e_2 \cong \widehat{e}_2$, by Lemma 4.2.3 we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2)$ such that $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \cong_\psi (\gamma, \sigma_1, \Delta_1, \overline{\chi}$, bid, acc, $e_2)$. Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}$, bid, acc, $e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}$, bid, acc, $v)$, by the inductive hypothesis we have $(\widehat{\gamma}, \widehat{\sigma}_1, \square, \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$ and ψ_2 such that $(\gamma, \sigma_2, \Delta_2, \overline{\chi}$, bid, acc, $v) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2, \square, \widehat{v})$ and $c_2 \cong d_2$. Given $v \neq$ skip, by Lemma 4.2.1 we have $\psi_2 = \psi$. By Definition 4.2.1 we have $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $v \cong_\psi \widehat{v}$. Given Label(e_2, γ) = public, we have Label(v, γ) = public and therefore $v = \widehat{v}$ by Definition 3.2.17.

Given $v \neq$ skip and $v \cong_\psi \widehat{v}$, by Definition 3.2.10 we have $\widehat{v} \neq$ skip.

Given $\gamma(x) = (l$, private const $bty*)$, $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}$, const $\widehat{bty}*)$ such that $l = \widehat{l}$ by private const $bty* \cong$ const $\widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_2(l) = (\omega$, private const $bty*$, 1, PermL(Freeable, private const $bty*$, private, 1)), $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}$, const $\widehat{bty}*$, 1, PermL(Freeable, const $\widehat{bty}*$, public, 1)) such that $\omega \cong_\psi \widehat{\omega}$.

Given DecodePtr(private const $bty*$, 1, ω) $=$ [1, (l_1, 0), [1], 1], private const $bty* \cong$ const $\widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$,

Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $(i < 0) \vee (i \geq n)$, $i = \widehat{i}$, and $n = \widehat{n}$, we have $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$.

Given $\text{T_WriteOOB}(\text{encrypt}(v), i, n, l_1, \text{private } bty, \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc}) = (\sigma_3, \Delta_3, 1)$ and $v = \widehat{v}$, by Definition 3.2.10 we have $\text{encrypt}(v) \cong_\psi \widehat{v}$. Given $i = \widehat{i}$, $n = \widehat{n}$, $l_1 = \widehat{l}_1$, $\text{private } bty \cong \widehat{bty}$, and $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, by Lemma 4.2.9 we have $\text{WriteOOB}(\widehat{v},\ \widehat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty},\ \widehat{\sigma}_2) = (\widehat{\sigma}_3,\ 1)$ such that $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1] = \widehat{e}_2)$, $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}_1) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ \widehat{i})$, $\widehat{\gamma}(x) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{n}))$, $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{n})$, $(\widehat{\gamma},\ \widehat{\sigma}_1,\ \square, \widehat{e}_2) \Downarrow'_{d_2} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \widehat{v})$, and $\text{WriteOOB}(\widehat{v},\ \widehat{i},\ \widehat{n},\ \widehat{l}_1,\ \widehat{bty},\ \widehat{\sigma}_2) = (\widehat{\sigma}_3,\ 1)$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \Downarrow'_{wao} (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$ by Vanilla C rule 1D Array Write Out of Bounds.

Given $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x[e_1] = e_2) \Downarrow^t_{wao1} (\gamma,\ \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \Downarrow'_{wao}$ $(\widehat{\gamma},\ \widehat{\sigma}_3,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wao1 \cong wao$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x) \Downarrow^t_{ra5} (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x) \Downarrow^t_{ra5} (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ [v_0, ..., v_{n-1}])$ by Location-tracking SMC^2 rule Private 1D Array Read Entire Array, we have $\gamma(x) = (l, \text{private const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)],\ [1],\ 1]$, $\sigma(l_1) = (\omega_1,\ \text{private } bty,\ n,\ \text{PermL}(\text{Freeable},\ \text{private } bty,\ \text{private},\ n))$, and $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$ such that $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$

and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable},\ \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{private } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega}_1$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*,\ 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*,\ \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n},\ \text{PermL}(\text{Freeable},\ bty,\ \text{public}, \widehat{n}))$, and $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega}_1) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{ra4} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ [\widehat{v}_0,\ ...,\widehat{v}_{n-1}])$ by Vanilla C rule 1D Array Read Entire Array.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, by Definition 4.2.1 we have $(\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ [v_0, ..., v_{n-1}]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ [\widehat{v}_0,\ ...,\widehat{v}_{n-1}])$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x) \Downarrow^t_{ra5} (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [v_0, ..., v_{n-1}]) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x}) \Downarrow'_{ra4} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ [\widehat{v}_0,\ ...,\widehat{v}_{n-1}])$, $\Pi \cong_\psi \Sigma$, and $ra5 \cong ra4$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ x) \Downarrow^t_{ra4} (\gamma,\ \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc},\ [v_0, ..., v_{n-1}])$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{ra4} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [v_0, ..., v_{n-1}])$ by Location-tracking SMC^2 rule Public 1D

Array Read Entire Array, we have $\gamma(x) = (l, \text{public const } bty*)$, $\sigma(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, and $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$ and ψ such that $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$, by Definition 4.2.1 we have $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $x \cong_\psi \widehat{x}$. By Definition 3.2.18 and Definition 3.2.10 we have $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, (l_1, 0), [1], 1]$, $\text{public const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l_1}, 0), [1], 1]$ where $[1, (l_1, 0), [1], 1] \cong_\psi [1, (\widehat{l_1}, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l_1}, 0)$.

Given $\sigma(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l_1 = \widehat{l_1}$, by Lemma 3.2.15 we have $\widehat{\sigma}(\widehat{l_1}) = (\widehat{\omega_1}, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega_1}$, $\text{public } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{public } bty \cong \widehat{bty}$, and $\omega_1 \cong_\psi \widehat{\omega_1}$, by Lemma 3.2.41 we have $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega_1}) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x})$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, $\text{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l_1}, 0)], [1], 1]$, $\widehat{\sigma}(\widehat{l_1}) = (\widehat{\omega_1}, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$, and $\text{DecodeVal}(bty, \widehat{n}, \widehat{\omega_1}) = [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{ra4} (\widehat{\gamma}, \widehat{\sigma}, \square, [\widehat{v}_0, ..., \widehat{v}_{n-1}])$ by Vanilla C rule 1D Array Read Entire Array.

Given $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$ and $[v_0, ..., v_{n-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{n-1}]$, by Definition 4.2.1 we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [v_0, ..., v_{n-1}]) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, [\widehat{v}_0, ..., \widehat{v}_{n-1}])$. Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{ra4} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [v_0, ..., v_{n-1}]) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x}) \Downarrow'_{ra4} (\widehat{\gamma}, \widehat{\sigma}, \square, [\widehat{v}_0, ..., \widehat{v}_{n-1}])$, $\Pi \cong_\psi \Sigma$, and $ra4 \cong ra4$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^{t}_{wa5} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^{t}_{wa5} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$ by Location-tracking SMC^2 rule Public 1D Array Write Entire Array, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [v_0, ..., v_{n_e-1}])$, $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v_m \neq \text{skip}$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_1(l) = (\omega,\ \text{public const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{public const } bty*,\ \text{public},\ 1))$, $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ [(l_1,\ 0)],\ [1],\ 1]$, $\sigma_1(l_1) = (\omega_1,\ \text{public } bty,\ n,\ \text{PermL}(\text{Freeable},\ \text{public } bty,\ \text{public},\ n))$, $n_e = n$, and $\text{T_UpdateVal}(\sigma_1, l_1, [v_0, ..., v_{n_e-1}], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty) = (\sigma_2, \Delta_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_{\psi} \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^{t}_{wa5} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^{t}_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [v_0, ..., v_{n_e-1}])$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [v_0, ..., v_{n_e-1}]) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$ and $c_1 \cong d_1$. Given $[v_0, ..., v_{n_e-1}] \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $[v_0, ..., v_{n_e-1}] \cong_{\psi} [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$.

Given $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v_m \neq \text{skip}$ and $[v_0, ..., v_{n_e-1}] \cong_{\psi} [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Definition 3.2.10 we have $\forall \widehat{v}_m \in [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}].\ \widehat{v}_m \neq \text{skip}$.

Given $\gamma(x) = (l, \text{public const } bty*)$, $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{public const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega,\ \text{public const } bty*,\ 1,\ \text{PermL}(\text{Freeable},\ \text{public const } bty*, \text{public}, 1))$, $(\gamma,\ \sigma_1) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_{\psi} \widehat{\omega}$.

Given DecodePtr(public const $bty*$, 1, ω) = [1, $(l_1, 0)$, [1], 1], public const $bty* \cong$ const $\widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have DecodePtr(const $\widehat{bty}*, 1, \widehat{\omega}$) = $[1, (\widehat{l_1}, 0), [1], 1]$ where [1, $(l_1, 0)$, [1], 1] $\cong_\psi [1, (\widehat{l_1}, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l_1}, 0)$.

Given $\sigma_1(l_1)$ = (ω_1, public bty, n, PermL(Freeable, public bty, public, n)), $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $l_1 = \widehat{l_1}$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l_1}) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, public $bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $n_e = n$, $n = \widehat{n}$, and $n_e = \widehat{n}_e$, we have $\widehat{n}_e = \widehat{n}$.

Given T_UpdateVal($\sigma_1, l_1, [v_0, ..., v_{n_e-1}]$, Δ_1, $\overline{\chi}$, bid, acc, public bty) = (σ_2, Δ_2), $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l_1 = \widehat{l_1}$, public $bty \cong \widehat{bty}$, and $[v_0, ..., v_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Lemma 4.2.4 we have UpdateVal($\widehat{\sigma}_1, \widehat{l_1}, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}], \widehat{bty}$) $= \widehat{\sigma}_2$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e})$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$, $\forall \widehat{v}_m \in [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]. \widehat{v}_m \neq$ skip, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}$, const $\widehat{bty}*$, 1, PermL(Freeable, const $\widehat{bty}*$, public, 1)), DecodePtr(const $\widehat{bty}*, 1, \widehat{\omega}$) = [1, $[(\widehat{l_1}, 0)]$, [1], 1], $\widehat{\sigma}_1(\widehat{l_1}) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}$, PermL(Freeable, bty, public, $\widehat{n}$)), $\widehat{n}_e = \widehat{n}$, and UpdateVal($\widehat{\sigma}_1, \widehat{l_1}$, $[\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}], \widehat{bty}$) $= \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{wa5} (\widehat{\gamma}, \widehat{\sigma}_2, \square$, skip) by Vanilla C rule 1D Array Write Entire Array.

Given $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma, \sigma_2, \Delta_2, \overline{\chi}$, bid, acc, skip) $\cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2, \square$, skip). Therefore, we have $(\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $x = e) \Downarrow^t_{wa5} (\gamma, \sigma_2, \Delta_2, \overline{\chi}$, bid, acc, skip) $\cong_\psi (\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{x} = \widehat{e}) \Downarrow'_{wa5}$ $(\widehat{\gamma}, \widehat{\sigma}_2, \square$, skip), $\Pi \cong_\psi \Sigma$, and $wa5 \cong wa5$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $x = e_1) \Downarrow^t_{wa6} (\gamma, \sigma_2, \Delta_2, \overline{\chi}$, bid, acc, skip)

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $x = e) \Downarrow^t_{wa6} (\gamma, \sigma_2, \Delta_2, \overline{\chi}$, bid, acc, skip) by Location-tracking SMC^2 rule Private 1D Array Write Entire Private Array, we have Label(e, γ) = private, $(\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $e) \Downarrow^t_{c_1} (\gamma, \sigma_1$, $\Delta_1, \overline{\chi}$, bid, acc, $[v_0, ..., v_{n_e-1}])$, $\forall v_m \in [v_0, ..., v_{n_e-1}]. v_m \neq$ skip, $\gamma(x) = (l$, private const $bty*)$, $\sigma_1(l) = (\omega$, private const $bty*$, 1, PermL(Freeable, private const $bty*$, private, 1)), DecodePtr(private const $bty*$, 1, ω) = [1, $[(l_1, 0)]$, [1], 1], $\sigma_1(l_1) = (\omega_1$, private bty, n, PermL(Freeable, private bty, private, n)), $n_e = n$, and T_UpdateVal($\sigma_1, l_1, [v_0, ..., v_{n_e-1}], \Delta_1, \overline{\chi}$, bid, acc, private bty) = (σ_2, Δ_2).

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{wa6} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip})$, by Lemma 4.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have $\text{Erase}(x = e) = \text{Erase}(x) = \text{Erase}(e)$, $\text{Erase}(x) = \widehat{x}$ where $x = \widehat{x}$, and $\text{Erase}(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [v_0, ..., v_{n_e-1}])$, by the inductive hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$ and ψ_1 such that $(\gamma,\ \sigma_1, \text{acc},\ [v_0, ..., v_{n_e-1}]) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$ and $c_1 \cong d_1$. Given $[v_0, ..., v_{n_e-1}] \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $[v_0, ..., v_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$.

Given $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v_m \neq \text{skip}$ and $[v_0, ..., v_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Definition 3.2.18 and Definition 3.2.10 we have $\forall \widehat{v}_m \in [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}].\ \widehat{v}_m \neq \text{skip}$.

Given $\gamma(x) = (l, \text{private const } bty*)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $n_e = n$, $n = \widehat{n}$, and $n_e = \widehat{n}_e$, we have $\widehat{n}_e = \widehat{n}$.

Given T_UpdateVal($\sigma_1, l_1, [v_0, ..., v_{n_e-1}]$, Δ_1, $\overline{\chi}$, bid, acc, private bty) $= (\sigma_2, \Delta_2)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l_1 = \widehat{l}_1$, private $bty \cong \widehat{bty}$, and $[v_0, ..., v_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Lemma 4.2.4 we have UpdateVal($\widehat{\sigma}_1, \widehat{l}_1$, $[\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}], \widehat{bty}) = \widehat{\sigma}_2$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma}, \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$, $\forall \widehat{v}_m \in [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}].\ \widehat{v}_m \neq$ skip, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$, DecodePtr(const $\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$, $\widehat{n}_e = \widehat{n}$, and UpdateVal($\widehat{\sigma}_1, \widehat{l}_1$, $[\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}], \widehat{bty}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_{wa5} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,$ skip) by Vanilla C rule 1D Array Write Entire Array.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi}$, bid, acc, skip$) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,$ skip). Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi}$, bid, acc, $x = e) \Downarrow^t_{wa6} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi}$, bid, acc, skip$) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_{wa5} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,$ skip), $\Pi \cong_\psi \Sigma$, and $wa6 \cong wa5$ by Definition 4.2.2.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi}$, bid, acc, $x = e) \Downarrow^t_{wa7} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi}$, bid, acc, skip$)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi}$, bid, acc, $x = e) \Downarrow^t_{wa7} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi}$, bid, acc, skip$)$ by Location-tracking SMC^2 rule Private 1D Array Write Entire Public Array, we have Label$(e, \gamma) =$ public, $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi}$, bid, acc, $e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi}$, bid, acc, $[v_0, ..., v_{n_e-1}])$, $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v_m \neq$ skip, $\gamma(x) = (l,$ private const $bty*)$, $(bty =$ int$) \vee (bty =$ float$)$, $\sigma_1(l) = (\omega,$ private const $bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, DecodePtr(private const $bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v'_m = \text{encrypt}(v_m)$, $\sigma_1(l_1) = (\omega_1,$ private $bty, n,$ PermL(Freeable, private bty, private, n)), $n_e = n$, and T_UpdateVal($\sigma_1, l_1, [v'_0, ..., v'_{n_e-1}]$, Δ_1, $\overline{\chi}$, bid, acc, private $bty) = (\sigma_2, \Delta_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$ and ψ such that $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi}$, bid, acc, $x = e) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, by Definition 4.2.1 we have $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi}$, bid, acc, $x = e) \Downarrow^t_{wa7} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi}$, bid, acc, skip$)$, by Lemma 4.2.2 we have $(l, \mu) \notin x = e$. Therefore, by Lemma 3.2.3 we have $x = e \cong \widehat{x} = \widehat{e}$. By Definition 3.2.10 we have Erase$(x = e) =$ Erase$(x) =$ Erase(e), Erase$(x) = \widehat{x}$ where $x = \widehat{x}$, and Erase$(e) = \widehat{e}$. Therefore, we have $e \cong \widehat{e}$.

Given $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $e \cong \widehat{e}$, by Lemma 4.2.3 we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e})$ such that $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \cong_\psi (\gamma,\ \sigma,\ \Delta,\ \overline{\chi}$, bid, acc, $e)$. Given $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi}$, bid, acc, $e) \Downarrow^t_{c_1} (\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi}$, bid, acc, $[v_0, ..., v_{n_e-1}])$, by the inductive

hypothesis we have $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{e}) \Downarrow'_{d_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$ and ψ_1 such that $(\gamma,\ \sigma_1,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ [v_0,\ ...,\ v_{n_e-1}]) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\ [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}])$ and $c_1 \cong d_1$. Given $[v_0,\ ...,\ v_{n_e-1}] \neq \text{skip}$, by Lemma 4.2.1 we have $\psi_1 = \psi$. By Definition 4.2.1 we have $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $[v_0, ..., v_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$. By Lemma 3.2.18, we have $n_e = \widehat{n}_e$. Given $\text{Label}(e, \gamma) = \text{public}$, we have $\text{Label}([v_0,\ ...,\ v_{n_e-1}], \gamma) = \text{public}$ and therefore $[v_0, ..., v_{n_e-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$ by Definition 3.2.18 and Definition 3.2.10.

Given $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v_m \neq \text{skip}$ and $[v_0, ..., v_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Definition 3.2.18 and Definition 3.2.10 we have $\forall \widehat{v}_m \in [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}].\ \widehat{v}_m \neq \text{skip}$.

Given $\gamma(x) = (l,\ \text{private const } bty*)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $x = \widehat{x}$, we have $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that $l = \widehat{l}$ by $\text{private const } bty* \cong \text{const } \widehat{bty}*$ by Lemma 3.2.14.

Given $\sigma_1(l) = (\omega,\ \text{private const } bty*,\ 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l = \widehat{l}$, by Lemma 3.2.16 we have $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable, const } \widehat{bty}*, \text{public}, 1))$ such that $\omega \cong_\psi \widehat{\omega}$.

Given $\text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ (l_1, 0),\ [1],\ 1]$, $\text{private const } bty* \cong \text{const } \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, Lemma 3.2.44 we have $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, (\widehat{l}_1, 0), [1], 1]$ where $[1,\ (l_1, 0),\ [1],\ 1] \cong_\psi [1, (\widehat{l}_1, 0), [1], 1]$ such that $(l_1, 0) \cong_\psi (\widehat{l}_1, 0)$.

Given $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v'_m = \text{encrypt}(v_m)$ and $[v_0,\ ...,\ v_{n_e-1}] = [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Definition 3.2.10 and Definition 3.2.18 we have $[v'_0, ..., v'_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$.

Given $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l_1 = \widehat{l}_1$, by Lemma 3.2.15 we have $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}, \text{PermL}(\text{Freeable}, bty, \text{public}, \widehat{n}))$ where $\omega_1 \cong_\psi \widehat{\omega}_1$, $\text{private } bty \cong \widehat{bty}$, and $n = \widehat{n}$.

Given $n_e = n$, $n = \widehat{n}$, and $n_e = \widehat{n}_e$, we have $\widehat{n}_e = \widehat{n}$.

Given $\text{UpdateVal}(\sigma_1, l_1, [v'_0, ..., v'_{n_e-1}],\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{private } bty) = (\sigma_2, \Delta_2)$, $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l_1 = \widehat{l}_1$, $\text{private } bty \cong \widehat{bty}$, and $[v'_0, ..., v'_{n_e-1}] \cong_\psi [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}]$, by Lemma 4.2.4 we have $\text{UpdateVal}(\widehat{\sigma}_1, \widehat{l}_1, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e-1}], \widehat{bty}) = \widehat{\sigma}_2$ such that $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given $(\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e})$, $(\widehat{\gamma}, \widehat{\sigma}, \square, \widehat{e})\ \Downarrow'_{d_1}\ (\widehat{\gamma}, \widehat{\sigma}_1, \square, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e - 1}])$, $\forall \widehat{v}_m \in [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e - 1}].\ \widehat{v}_m \neq \text{skip}$, $\widehat{\gamma}(\widehat{x}) = (\widehat{l}$, const $\widehat{bty}*)$, $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}$, const $\widehat{bty}*$, 1, PermL(Freeable, const $\widehat{bty}*$, public, 1)), DecodePtr(const $\widehat{bty}*$, 1, $\widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$, $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{n}$, PermL(Freeable, bty, public, $\widehat{n}))$, $\widehat{n}_e = \widehat{n}$, and UpdateVal$(\widehat{\sigma}_1, \widehat{l}_1, [\widehat{v}_0, ..., \widehat{v}_{\widehat{n}_e - 1}], \widehat{bty}) = \widehat{\sigma}_2$, we have $\Sigma \triangleright (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_{wa5} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$ by Vanilla C rule 1D Array Write Entire Array.

Given $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, by Definition 4.2.1 we have $(\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$. Therefore, we have $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ x = e) \Downarrow^t_{wa7} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{skip}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma},\ \square,\ \widehat{x} = \widehat{e}) \Downarrow'_{wa5} (\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\ \text{skip})$, $\Pi \cong_\psi \Sigma$, and $wa7 \cong wa5$ by Definition 4.2.2.

$\square$

4.3 Noninterference

Location-tracking SMC^2 satisfies a strong form of noninterferences guaranteeing that two execution traces are indistinguishable up to differences in private values. This stronger version entails data-obliviousness. Instead of using execution traces, we will work directly with evaluation trees in the Location-tracking SMC^2 semantics – equivalence of evaluation trees up to private values implies equivalence of execution traces based on the Location-tracking SMC^2 semantics. This guarantee is provided at the semantics level, we do not consider here compiler optimizations.

For noninterference, it is convenient to introduce a notion of equivalence requiring that the two memories agree on publicly observable values. Because we assume that private data in memories are encrypted, and so their encrypted value is publicly observable, it is sufficient to consider syntactic equality of memories. Notice that if $\sigma_1 = \sigma_2$ we can still have $\sigma_1\ell \neq \sigma_2\ell$, i.e., two executions starting from the same configuration can actually differ with respect to private data. What we show is that this difference can occur only in atomic operations working on private data, which we assume is not publicly observable.

We want to consider two evaluation trees as *low-equivalent* if they are identical up to private relational operations. To formalize this, we need first to identify codes up to private relational operations – these are atomic operations that are implemented by means of some cryptographic primitive and we assume that their difference is not publicly observable. We define *low-equivalence* over Location-tracking SMC^2 evaluation codes in Definition 3.3.1 and evaluation trees in Definition 4.3.1. Based on the notion of low-equivalence between evaluation trees, we can now state our main noninterference result.

Theorem 4.3.1 (Noninterference over evaluation trees)**.** *For every environment γ, γ', γ''; memory σ, σ', $\sigma'' \in$ Mem; location map Δ, Δ', Δ''; local variable tracker $\overline{\chi}$, $\overline{\chi}'$, $\overline{\chi}''$, branch identifier* bid, bid', bid''*; accumulator* acc, acc', $\mathrm{acc}'' \in \mathbb{N}$*; statement s, values v', v''; step evaluation codes $[d'_1, ..., d'_n]$, $[d''_1, ..., d''_n]$; if $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s) \Downarrow^t_{[d'_1,...,d'_n]} (\gamma', \sigma', \Delta', \overline{\chi}', \mathrm{bid}', \mathrm{acc}', v')$ and $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s) \Downarrow^t_{[d''_1,...,d''_n]} (\gamma'', \sigma'', \Delta'', \overline{\chi}'', \mathrm{bid}'', \mathrm{acc}'', v'')$, then $\gamma' = \gamma''$, $\Delta' = \Delta''$, $\overline{\chi}' = \overline{\chi}''$, $\mathrm{bid}' = \mathrm{bid}''$, $\sigma' = \sigma''$, $\mathrm{acc}' = \mathrm{acc}''$, $v' = v''$, $[d'_1, ..., d'_n] \simeq_L [d''_1, ..., d''_n]$, and $\Pi \simeq_L \Sigma$.*

Proof. Proof Sketch: By induction over all Location-tracking SMC^2 semantic rules. Notice that low-equivalence of evaluation trees already implies the equivalence of the resulting configurations. We repeated them to make the meaning of the theorem clearer. Moreover, notice that two evaluation trees can differ only in atomic operations implemented through cryptographic primitives. Thus the two corresponding traces are equivalent and data-obliviousness follows.

We make the assumption that both evaluation traces are over the same program (this is given by having the same s

in the starting states) and all public data will remain the same, including data read as input during the evaluation of the program. A portion of the complexity of this proof is within ensuring that memory accesses within our semantics remain data oblivious. Several rules follow fairly simply and leverage similar ideas, which we will discuss first, and then we will provide further intuition behind the more complex cases. The full proof is available in Section 4.3.2, with this theorem identical to Theorem 4.3.2.

For all rules leveraging helper algorithms, we must reason about the helper algorithms, and that they behave deterministically by definition and have data-oblivious memory accesses. Given this and that these helper algorithms do no modify the private data, we maintain the properties of noninterference of this theorem. First we reason that our helper algorithms to translate values into their byte representation will do so deterministically, and therefore maintain indistinguishability between the value and byte representation. We can then reason that our helper algorithms that take these byte values and store them into memory will also do so deterministically, so that when we later access the data in memory we will obtain the same indistinguishable values we had stored.

It is also important to take note here our functions to help us retrieve data from memory, particularly in cases such as when reading out of bounds of an array. When proving these cases to maintain noninterference, we leverage our definition of how memory blocks are assigned in a monotonically increasing fashion, and how the algorithms for choosing which memory block to read into after the current one are deterministic. This, as well as our original assumptions of having identical public input, allows us to reason that if we access out of bounds (including accessing data at a non-aligned position, such as a chunk of bytes in the middle of a memory block), we will be pulling from the same set of bytes each time, and therefore we will end up with the same interpretation of the data as we continue to evaluate the remainder of the program. It is important to note again here that by definition, our semantics will always interpret bytes of data as the type it is expected to be, not the type it actually is (i.e., reading bytes of data that marked private in memory by overshooting a public array will not decrypt the bytes of data, but instead give you back a garbage public value). To reiterate this point, even when reading out of bounds, we will not reveal anything about private data, as the results of these helper algorithms will be indistinguishable.

For private pointers, it is important to note that the obtaining multiple locations is deterministic based upon the program that is being evaluated. A pointer can initially gain multiple locations through the evaluation of a private if else. Once there exists a pointer that has obtained multiple locations in such a way, it can be assigned to another pointer to give that pointer multiple locations. The other case for a pointer to gain multiple location is through the use of `pfree` on a pointer with multiple locations (i.e., the case where a pointer has locations l_1, l_2, l_3 and we free l_1) - when this occurs, if another pointer had referred to only l_1, it will now gain locations in order to mask whether we had to move the true location or not. When reasoning about pointers with multiple locations, we maintain that given the tags for which location is the true location are indistinguishable, then it is not possible to distinguish between them by their usage as defined in the rules or helper algorithms using them. Additionally, to reason about `pfree`, we leverage

that the definitions of the helper algorithms are deterministic, and that (wlog), we will be freeing the same location. We will then leverage our Axiom about the multiparty protocol MPC_{free}. After the evaluation of MPC_{free}, it will deterministically update memory and all other pointers as we mentioned in the brief example above.

For the Private If Else rule, the most important element we must leverage is how values are resolved, showing that given our resolution style, we are not able to distinguish between the ending values. In order to do this, we also must reason about the entirety of the rule, including all of if else helper algorithms. First, we note that the evaluation of the **`then`** branches follows by induction, as does the evaluation of the **`else`** branch once we have reasoned through the restoration phase. We must then reason about our rules that update memory and our update algorithms, and how given a program, we will deterministically find all modifications to memory and add them to our tracking structure Δ properly. Then we can reason that the behavior of $\mathrm{T_restore}$ will deterministically perform the same updates, because Δ will contain the same information in every evaluation. Now, we are able to move on to reasoning about resolution, and show that given all of this and the definitions of the resolution helper algorithms and rule, we are not able to distinguish between the ending values.

Within the array rules, the main concern is in reading from and writing at a private index. We currently handle this complexity within our rules by accessing all locations within the array in rules Location-tracking Array Read Private Index and Location-tracking Array Write Private Index. In Location-tracking Array Read Private Index, we clearly read data from every index of the array, privately computing the true value from all values in the array. Similarly, in Location-tracking Array Write Private Index, we read data from every index of the array, then proceed to privately update every value of in array. All other array rules use public indices, and in turn only access that publicly known location. Within the pointer rules, our main concern is that we access all locations that are referred to by a private pointer when we have multiple locations. For this, we will reason about the contents of the rules and the helper algorithms used by the pointer rules, which can be shown to deterministically do so.

□

4.3.1 Supporting Metatheory

For the Proof of Noninterference over the Location-tracking semantics several definitions, axioms, and lemmas remain unchanged:

- Definitions 3.3.1, 3.3.2, 3.3.4, 3.3.5,
- Axioms 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5,
- Lemmas 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5, 3.3.6, 3.3.7, 3.3.8, 3.3.11, 3.3.9, 3.3.10, 3.3.17, 3.3.18, 3.3.19, 3.3.20, 3.3.21, 3.3.22, 3.3.23, 3.3.24

Definition 4.3.1. Two Location-tracking SMC^2 evaluation trees Π and Σ are *low-equivalent*, in symbols $\Pi \simeq_L \Sigma$, if and only if Π and Σ have the same structure as trees, and for each node in Π proving $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s) \Downarrow_{c_\Pi} (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid},\ \text{acc}_1,\ v)$, the corresponding node in Σ proves $(\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \text{bid},\ \text{acc},\ s) \Downarrow_{c_\Sigma} (\gamma_1,\ \sigma_1,\ \Delta_1,\ \overline{\chi}_1,\ \text{bid}, \text{acc}_1,\ v)$ and $c_\Pi \cong_L c_\Sigma$.

Lemma 4.3.1. *Given memory σ, σ', location map Δ, Δ', and accumulator* acc, acc'*, if* $\text{T_restore}(\sigma, \Delta, \text{acc}) = \sigma_2$, $\text{T_restore}(\sigma', \Delta', \text{acc}') = \sigma'_2$, $\sigma = \sigma'$, $\Delta = \Delta'$, *and* $\text{acc} = \text{acc}'$, *then* $\sigma_2 = \sigma'_2$.

Proof. By definition of Algorithm T_restore, T_restore is deterministic. □

Lemma 4.3.2. *Given memory σ, σ', location map Δ, Δ', local variable tracker $\overline{\chi}, \overline{\chi}'$, branch identifier* bid, bid'*, accumulator* acc, acc'*, and variable name $res_{\text{acc}}, res'_{\text{acc}}$, if* $\text{T_resolve}(\sigma, \Delta, \overline{\chi}, \text{bid},\ \text{acc}, res_{\text{acc}}) = \sigma_1$, $\text{T_resolve}(\sigma', \Delta', \overline{\chi}', \text{bid}', \text{acc}', res'_{\text{acc}}) = \sigma_1$, $\sigma = \sigma'$, $\text{acc} = \text{acc}'$, *and* $res_{\text{acc}} = res'_{\text{acc}}$, *then* $\sigma_1 = \sigma'_1$.

Proof. By definition of Algorithm T_resolve, T_resolve is deterministic. □

Lemma 4.3.3. *Given memory $\sigma_1,\ \sigma'_1$, memory block identifier $l,\ l'$, value $v,\ v'$, location map Δ, Δ', local variable tracker $\overline{\chi}, \overline{\chi}'$, branch identifier* bid, bid'*, accumulator* acc, acc'*, and $ty,\ ty'$, if* $\text{T_UpdateVal}(\sigma_1,\ l,\ v,\ \Delta_1,\ \overline{\chi},\ \text{bid},\ \text{acc}, ty) = (\sigma_2,\ \Delta_2)$, $\text{T_UpdateVal}(\sigma'_1,\ l', v', \Delta'_1, \overline{\chi}',\ \text{bid}', \text{acc}', ty') = (\sigma'_2,\ \Delta'_2)$, $\sigma_1 = \sigma'_1$, $l = l'$, $v = v'$, $\Delta = \Delta'$, $\overline{\chi} = \overline{\chi}'$, $\text{bid} = \text{bid}'$, $\text{acc} = \text{acc}'$*and $ty = ty'$, then $\sigma_2 = \sigma'_2$ and $\Delta_2 = \Delta'_2$.*

Proof. By definition of Algorithm T_UpdateVal, T_UpdateVal is deterministic. □

Lemma 4.3.4. *Given memory $\sigma_1,\ \sigma'_1$, location $(l,\ \mu)$, $(l',\ \mu')$, value $v,\ v'$, location map Δ_1, Δ'_1, local variable tracker $\overline{\chi},\ \overline{\chi}'$, branch identifier* $\text{bid},\ \text{bid}'$*, accumulator* $\text{acc},\ \text{acc}'$*, and type $ty,\ ty'$, if* $\text{T_UpdateOffset}(\sigma_1,\ (l,\ \mu),\ v,\ \Delta_1,\ \overline{\chi}, \text{bid}, \text{acc},\ ty) = (\sigma_2,\ \Delta_2,\ j)$, $\text{T_UpdateOffset}(\sigma'_1,\ (l',\ \mu'),\ v', \Delta'_1,\ \overline{\chi}',\ \text{bid}',\ \text{acc}', ty') = (\sigma'_2, \Delta'_2, j')$, $\sigma_1 = \sigma'_1$, $(l, \mu) = (l', \mu')$, $v = v'$, $ty = ty'$, $\Delta_1 = \Delta'_1$, $\overline{\chi} = \overline{\chi}'$, $\text{bid} = \text{bid}'$, *and* $\text{acc} = \text{acc}'$, *then $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, and $j = j'$.*

Proof. By definition of Algorithm T_UpdateOffset, T_UpdateOffset is deterministic. □

Lemma 4.3.5. *Given memory σ_1, σ'_1, memory block identifier list $\overline{l}, \overline{l}'$, tag list $\overline{j}, \overline{j}'$, type ty, ty', location map Δ_1, Δ'_1, local variable tracker $\overline{\chi}, \overline{\chi}'$, branch identifier* bid, bid'*, accumulator* acc, acc'*, and value v_1, v_2, v'_1, v'_2, if* $\text{T_UpdatePriv}(\sigma_1, v_1, \overline{l}, \overline{j}, ty, v_2, \Delta_1,\ \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_2, \Delta_2, j)$, $\text{T_UpdatePriv}(\sigma_1, v'_1, \overline{l}', \overline{j}', ty', v'_2, \Delta'_1, \overline{\chi}',\ \text{bid}', \text{acc}') = (\sigma'_2, \Delta'_2, j')$, $\sigma_1 = \sigma'_1$, $v_1 = v'_1$, $\overline{l} = \overline{l}'$, $\overline{j} = \overline{j}'$, $ty = ty'$, $v_2 = v'_2$, $\Delta_1 = \Delta'_1$, $\overline{\chi} = \overline{\chi}'$, $\text{bid} = \text{bid}'$, *and* $\text{acc} = \text{acc}'$, *then $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, and $j = j'$.*

Proof. By definition of Algorithm T_UpdatePriv, T_UpdatePriv is deterministic. □

Lemma 4.3.6. *Given memory* σ_1, σ_1'*, location* $(l, \mu), (l', \mu')$*, pointer data structure* $[\alpha, \bar{l}, \bar{j}, i]$*,* $[\alpha', \bar{l}', \bar{j}', i']$*, location map* Δ_1, Δ_1'*, local variable tracker* $\overline{\chi}, \overline{\chi}'$*, branch identifier* $\mathrm{bid}, \mathrm{bid}'$*, accumulator* $\mathrm{acc}, \mathrm{acc}'$*, and type* ty, ty'*, if* $\mathrm{T_UpdatePrivPtr}(\sigma_1,\ (l,\ \mu),\ [\alpha,\ \bar{l},\ \bar{j},\ i],\ \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ ty) = (\sigma_2,\ \Delta_2,\ j)$*,* $\mathrm{T_UpdatePrivPtr}(\sigma_1',\ (l', \mu'),\ [\alpha', \bar{l}', \bar{j}', i'],\ \Delta_1', \overline{\chi}', \mathrm{bid}',\ \mathrm{acc}', ty') = (\sigma_2', \Delta_2', j')$*,* $\sigma_1 = \sigma_1'$*,* $(l, \mu) = (l', \mu')$*,* $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$*,* $\Delta_1 = \Delta_1'$*,* $\overline{\chi} = \overline{\chi}'$*,* $\mathrm{bid} = \mathrm{bid}'$*,* $\mathrm{acc} = \mathrm{acc}'$*, and* $ty = ty'$*, then* $\sigma_2 = \sigma_2'$*,* $\Delta_2 = \Delta_2'$*, and* $j = j'$*.*

Proof. By definition of Algorithm T_UpdatePtr, T_UpdatePtr is deterministic. □

Lemma 4.3.7. *Given memory* σ_1, σ_1'*, location* $(l, \mu), (l', \mu')$*, pointer data structure* $[\alpha, \bar{l}, \bar{j}, i]$*,* $[\alpha_1, \bar{l}_1, \bar{j}_1, i_1,\ [\alpha', \bar{l}', \bar{j}', i'],\ [\alpha_1', \bar{l}_1', \bar{j}_1', i_1']$*, location map* Δ_1, Δ_1'*, local variable tracker* $\overline{\chi}, \overline{\chi}'$*, branch identifier* $\mathrm{bid},\ \mathrm{bid}'$*, accumulator* $\mathrm{acc}, \mathrm{acc}'$*, and type* ty, ty'*, if* $\mathrm{T_UpdatePrivPtr}(\sigma_1,\ [\alpha, \bar{l}, \bar{j}, i],\ [\alpha_1, \bar{l}_1, \bar{j}_1, i_1], \Delta_1,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc}, ty) = (\sigma_2,\ \Delta_2,\ j)$*,* $\mathrm{T_UpdatePrivPtr}(\sigma_1, [\alpha', \bar{l}', \bar{j}', i'], [\alpha_1', \bar{l}_1', \bar{j}_1', i_1'], \Delta_1', \overline{\chi}', \mathrm{bid}', \mathrm{acc}', ty') = (\sigma_2', \Delta_2', j')$*,* $\sigma_1 = \sigma_1'$*,* $(l, \mu) = (l', \mu')$*,* $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$*,* $[\alpha_1, \bar{l}_1, \bar{j}_1, i_1] = [\alpha_1', \bar{l}_1', \bar{j}_1', i_1']$ $ty = ty'$*,* $\Delta_1 = \Delta_1'$*,* $\overline{\chi} = \overline{\chi}'$*,* $\mathrm{bid} = \mathrm{bid}'$*, and* $\mathrm{acc} = \mathrm{acc}'$*, then* $\sigma_2 = \sigma_2'$*,* $\Delta_2 = \Delta_2'$*, and* $j = j'$*.*

Proof. By definition of Algorithm T_UpdatePrivPtr, T_UpdatePrivPtr is deterministic. □

Lemma 4.3.8. *Given value* v, v'*, number* n_1, n_1', n_2, n_2'*, memory block identifier* l, l'*, type* ty, ty'*, location map* Δ, Δ'*, local variable tracker* $\overline{\chi}, \overline{\chi}'$*, branch identifier* $\mathrm{bid}, \mathrm{bid}'$*, accumulator* $\mathrm{acc}, \mathrm{acc}'$*, and memory* σ, σ'*,*
if $\mathrm{T_WriteOOB}(v, n_1, n_2, l, ty, \sigma, \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc}) = (\sigma_1, \Delta_1, j)$*,* $\mathrm{T_WriteOOB}(v', n_1', n_2', l',\ ty', \sigma', \Delta', \overline{\chi}', \mathrm{bid}', \mathrm{acc}') = (\sigma_1', \Delta_1', j')$*,* $v = v'$*,* $n_1 = n_1'$*,* $n_2 = n_2'$*,* $l = l'$*,* $ty = ty'$*,* $\sigma = \sigma'$*,* $\Delta = \Delta'$*,* $\overline{\chi} = \overline{\chi}'$*,* $\mathrm{bid} = \mathrm{bid}'$*, and* $\mathrm{acc} = \mathrm{acc}'$*, then* $\sigma_1 = \sigma_1'$*,* $\Delta_1 = \Delta_1'$*, and* $j = j'$*.*

Proof. By definition of Algorithm T_WriteOOB, T_WriteOOB is deterministic. □

4.3.2 Proof of Noninterference

Theorem 4.3.2 (Noninterference over evaluation trees)**.** *For every environment* γ*,* γ'*,* γ''*; memory* σ*,* σ'*,* $\sigma'' \in \mathrm{Mem}$*; location map* Δ*,* Δ'*,* Δ''*; local variable tracker* $\overline{\chi}, \overline{\chi}', \overline{\chi}''$*, branch identifier* $\mathrm{bid}, \mathrm{bid}', \mathrm{bid}''$*; accumulator* acc*,* acc'*,* $\mathrm{acc}'' \in \mathbb{N}$*; statement* s*, values* v'*,* v''*; step evaluation codes* $[c_1', ..., c_n'], [c_1'', ..., c_n'']$*;*
if $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ s) \Downarrow^t_{[c_1', ..., c_n']} (\gamma',\ \sigma',\ \Delta',\ \overline{\chi}',\ \mathrm{bid}',\ \mathrm{acc}',\ v')$ *and* $\Sigma \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ s) \Downarrow^t_{[c_1'', ..., c_n'']} (\gamma'',\ \sigma'',\ \Delta'',\ \overline{\chi}'',\ \mathrm{bid}'',\ \mathrm{acc}'',\ v'')$*,*
then $\gamma' = \gamma''$*,* $\Delta' = \Delta''$*,* $\overline{\chi}' = \overline{\chi}''$*,* $\mathrm{bid}' = \mathrm{bid}''$*,* $\sigma' = \sigma''$*,* $\mathrm{acc}' = \mathrm{acc}''$*,* $v' = v''$*,* $[c_1', ..., c_n'] \simeq_L [c_1'', ..., c_n'']$*, and* $\Pi \simeq_L \Sigma$*.*

Proof.

Case $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1 < e_2) \Downarrow^t_{ltt1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3)$

Given $\Pi \triangleright (\gamma,\ \sigma,\ \Delta,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ e_1 < e_2) \Downarrow^t_{ltt1} (\gamma,\ \sigma_2,\ \Delta_2,\ \overline{\chi},\ \mathrm{bid},\ \mathrm{acc},\ n_3)$ by rule Location-tracking Private Less

Than True, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_1)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_2)$, $n_1 <_{\text{private}} n_2$, and $\text{encrypt}(1) = n_3$.

By definition 3.3.1, given $c = \textit{ltt1}$, we have $c \simeq_L c'$ if $c' = \textit{ltt1} \vee \textit{ltf1}$. Therefore, we have the following two subcases:

Subcase $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{\textit{ltt1}} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n'_3)$

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{\textit{ltt1}} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n'_3)$ by rule Location-tracking Private Less Than True, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n'_1)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e'_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n'_2)$, $n'_1 <_{\text{private}} n'_2$, and $\text{encrypt}(1) = n'_3$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_1)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n'_1)$, by the inductive hypothesis we have that $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $n_1 = n'_1$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_2)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n'_2)$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have that $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $n_2 = n'_2$, and $c_2 \simeq_L c'_2$.

Given $\text{encrypt}(1) = n_3$ and $\text{encrypt}(1) = n'_3$, we have $n_3 = n'_3 = \text{encrypt}(1)$.

Therefore we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, and $n_3 = n'_3$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Subcase $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{\textit{ltf1}} (\gamma, \sigma_2, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{\textit{ltf1}} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$ by rule Location-tracking Private Less Than False, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_e (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n'_1)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_e (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n'_2)$, $n'_1 >=_{\text{private}} n'_2$, and $\text{encrypt}(0) = n'_3$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \Delta, \overline{\chi}, \text{bid}, \sigma_1, \text{acc}, n_1)$ and $(\gamma, \Delta, \overline{\chi}, \text{bid}, \sigma, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n'_1)$, by the inductive hypothesis we have that $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $n_1 = n'_1$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_2)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n'_2)$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have that $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $n_2 = n'_2$, and $c_2 \simeq_L c'_2$.

Given $\text{encrypt}(1) = n_3$ and $\text{encrypt}(0) = n'_3$, by Axiom 3.3.1 we have $n_3 = n'_3$.

Therefore we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, and $n_3 = n_3'$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \ e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltf1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltf2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \ e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltf3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 == e_2) \Downarrow^t_{eqt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 == e_2) \Downarrow^t_{eqt2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 == e_2) \Downarrow^t_{eqt3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 == e_2) \Downarrow^t_{eqf1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 == e_2) \Downarrow^t_{eqf2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 == e_2) \Downarrow^t_{eqf3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1! = e_2) \Downarrow^t_{net1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1! = e_2) \Downarrow^t_{net2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1! = e_2) \Downarrow^t_{net3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1! = e_2) \Downarrow^t_{nef1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1! = e_2) \Downarrow^t_{nef2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1! = e_2) \Downarrow^t_{nef3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, 1)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, 1)$ by rule Location-tracking Public Less Than True, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_1)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_2)$, and $n_1 <_{\text{public}} n_2$.

By definition 3.3.1, given $c = ltt$, we have $c \simeq_L c'$ if and only if $c' = ltt$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 < e_2) \Downarrow^t_{ltt} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, 1)$ by rule Location-tracking Public Less Than True, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n'_1)$, $(\gamma,$

σ_1', Δ_1', $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c_2'}$ (γ, σ_2', Δ_2', $\overline{\chi}$, bid, acc, n_2'), and $n_1' <_{\text{public}} n_2'$

Given (γ, σ, Δ, $\overline{\chi}$, bid, acc, e_1) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, n_1) and (γ, σ, Δ, $\overline{\chi}$, bid, acc, e_1) $\Downarrow^t_{c_1'}$ (γ, σ_1', Δ_1', $\overline{\chi}$, bid, acc, n_1'), by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $n_1 = n_1'$, and $c_1 \simeq_L c_1'$.

Given (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c_2}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n_2), (γ, σ_1', Δ_1', $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c_2'}$ (γ, σ_2', Δ_2', $\overline{\chi}$, bid, acc, n_2'), and $\sigma_1 = \sigma_1'$, by the inductive hypothesis we have $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $n_2 = n_2'$, and $c_2 \simeq_L c_2'$.

Therefore we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, and $1 = 1$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 < e_2$) $\Downarrow^t_{ltf}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, 0)
This case is similar to Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 < e_2$) $\Downarrow^t_{ltt}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, 1).

Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 == e_2$) $\Downarrow^t_{eqt}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, 1)
This case is similar to Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 < e_2$) $\Downarrow^t_{ltt}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, 1).

Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 == e_2$) $\Downarrow^t_{eqf}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, 0)
This case is similar to Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 < e_2$) $\Downarrow^t_{ltt}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, 1).

Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 != e_2$) $\Downarrow^t_{net}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, 1)
This case is similar to Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 < e_2$) $\Downarrow^t_{ltt}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, 1).

Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 != e_2$) $\Downarrow^t_{nef}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, 0)
This case is similar to Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 < e_2$) $\Downarrow^t_{ltt}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, 1).

Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 + e_2$) $\Downarrow^t_{bp}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n_3)
Given $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 + e_2$) $\Downarrow^t_{bp}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n_3) by rule Location-tracking Public Addition, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, ($\gamma$, σ, Δ, $\overline{\chi}$, bid, acc, e_1) $\Downarrow^t_{c_1}$ (γ, σ_1, acc, n_1), (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c_2}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n_2), and $n_1 +_{\text{public}} n_2 = n_3$.

By definition 3.3.1, given $c = bp$, we have $c \simeq_L c'$ if and only if $c' = bp$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n'_3)$ by rule Location-tracking Public Addition, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n_1)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n'_2)$, and $n'_1 +_{\text{public}} n'_2 = n'_3$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_1)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n'_1)$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $n_1 = n'_1$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_2)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n'_2)$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $n_2 = n'_2$, and $c_2 \simeq_L c'_2$.

Given $n_1 = n'_1$, $n_2 = n'_2$, $n_1 +_{\text{public}} n_2 = n_3$, and $n'_1 +_{\text{public}} n'_2 = n'_3$, we have $n_3 = n'_3$.

Therefore we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, and $n_3 = n'_3$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 - e_2) \Downarrow^t_{bs} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 \cdot e_2) \Downarrow^t_{bm} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$ by rule Location-tracking Private Addition, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_1)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_2)$, and $n_1 +_{\text{private}} n_2 = n_3$.

By definition 3.3.1, given $c = bp1$, we have $c \simeq_L c'$ if and only if $c' = bp1$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp1} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n'_3)$ by rule Location-tracking Private Addition, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n'_1)$, $(\gamma, \sigma'_1,$

Δ_1', $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c_2'}$ (γ, σ_2', Δ_2', $\overline{\chi}$, bid, acc, n_2'), and $n_1' +_{\text{private}} n_2' = n_3'$.

Given (γ, σ, Δ, $\overline{\chi}$, bid, acc, e_1) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, n_1) and (γ, σ, Δ, $\overline{\chi}$, bid, acc, e_1) $\Downarrow^t_{c_1'}$ (γ, σ_1', Δ_1', $\overline{\chi}$, bid, acc, n_1'), by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $n_1 = n_1'$, and $c_1 \simeq_L c_1'$.

Given (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c_2}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n_2), (γ, σ_1', Δ_1', $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c_2'}$ (γ, σ_2', Δ_2', $\overline{\chi}$, bid, acc, n_2'), and $\sigma_1 = \sigma_1'$, by the inductive hypothesis we have $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $n_2 = n_2'$, and $c_2 \simeq_L c_2'$.

Given $n_1 = n_1'$, $n_2 = n_2'$, $n_1 +_{\text{private}} n_2 = n_3$, and $n_1' +_{\text{private}} n_2' = n_3'$, by Axiom 3.3.2 we have $n_3 = n_3'$.

Therefore we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, and $n_3 = n_3'$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 - e_2$) $\Downarrow^t_{bs1}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n_3)
This case is similar to Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 + e_2$) $\Downarrow^t_{bp1}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n_3).

Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 \cdot e_2$) $\Downarrow^t_{bm1}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n_3)
This case is similar to Case $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 + e_2$) $\Downarrow^t_{bp1}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n_3).

Case $\Pi \triangleright$ (γ, σ, acc, Δ, $\overline{\chi}$, bid, $e_1 + e_2$) $\Downarrow^t_{bp2}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n_3)
Given $\Pi \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 + e_2$) $\Downarrow^t_{bp2}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n_3) by rule Location-tracking Public-Private Addition, we have $\text{Label}(e_1, \gamma) = \text{public}$, $\text{Label}(e_2, \gamma) = \text{private}$, ($\gamma$, σ, Δ, $\overline{\chi}$, bid, acc, e_1) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, n_1), (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c_2}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, n_2), and $\text{encrypt}(n_1) +_{\text{private}} n_2 = n_3$.

By definition 3.3.1, given $c = bp2$, we have $c \simeq_L c'$ if and only if $c' = bp2$.

Given $\Sigma \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, $e_1 + e_2$) $\Downarrow^t_{bp2}$ (γ, σ_2', Δ_2', $\overline{\chi}$, bid, acc, n_3') by rule Location-tracking Public-Private Addition, we have $\text{Label}(e_1, \gamma) = \text{public}$, $\text{Label}(e_2, \gamma) = \text{private}$, ($\gamma$, σ, Δ, $\overline{\chi}$, bid, acc, e_1) $\Downarrow^t_{c_1'}$ (γ, σ_1', Δ_1', $\overline{\chi}$, bid, acc, n_1'), (γ, σ_1', Δ_1', $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c_2'}$ (γ, σ_2', Δ_2', $\overline{\chi}$, bid, acc, n_2'), and $\text{encrypt}(n_1') +_{\text{private}} n_2' = n_3'$.

Given (γ, σ, Δ, $\overline{\chi}$, bid, acc, e_1) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, n_1) and (γ, σ, Δ, $\overline{\chi}$, bid, acc, e_1) $\Downarrow^t_{c_1'}$ (γ, σ_1', Δ_1', $\overline{\chi}$, bid, acc, n_1'), by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $n_1 = n_1'$, and $c_1 \simeq_L c_1'$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_2)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n'_2)$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $n_2 = n'_2$, and $c_2 \simeq_L c'_2$.

Given $n_1 = n'_1$, $\text{encrypt}(n_1)$, and $\text{encrypt}(n'_1)$, by Axiom 3.3.1 we have that $\text{encrypt}(n_1) = \text{encrypt}(n'_1)$.

Given $\text{encrypt}(n_1) = \text{encrypt}(n'_1)$, $n_2 = n'_2$, $\text{encrypt}(n_1) +_{\text{private}} n_2 = n_3$, and $\text{encrypt}(n'_1) +_{\text{private}} n'_2 = n'_3$, by Axiom 3.3.2 we have $n_3 = n'_3$.

Therefore we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, and $n_3 = n'_3$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp\mathcal{3}} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp\mathcal{2}} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 - e_2) \Downarrow^t_{bs\mathcal{2}} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp\mathcal{2}} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 - e_2) \Downarrow^t_{bs\mathcal{3}} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp\mathcal{2}} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 \cdot e_2) \Downarrow^t_{bm\mathcal{2}} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp\mathcal{2}} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 \cdot e_2) \Downarrow^t_{bm\mathcal{3}} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1 + e_2) \Downarrow^t_{bp\mathcal{2}} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n_3)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow^t_{iep} (\gamma, \sigma_6, \Delta_6, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow^t_{iep} (\gamma, \sigma_6, \Delta_6, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private If Else, we have $\text{Label}(e, \gamma) = \text{private}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private int } res_{\text{acc}+1} = n) \Downarrow^t_{c_2} (\gamma_1, \sigma_2, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$, $\Delta_2 = \Delta_1.push([\,])$, $\overline{\chi}_2 = \overline{\chi}_1.push([\,])$, $(\gamma_1, \sigma_2,$

Δ_2, $\overline{\chi}_2$, *then*, acc + 1, s_1) $\Downarrow^t_{c_3}$ (γ_2, σ_3, Δ_3, $\overline{\chi}_3$, *then*, acc + 1, skip), T_restore(σ_3, Δ_3, acc + 1) = σ_4, (γ_1, σ_4, Δ_3, $\overline{\chi}_2$, *else*, acc + 1, s_2) $\Downarrow^t_{c_4}$ (γ_3, σ_5, Δ_4, $\overline{\chi}_4$, *else*, acc + 1, skip), T_resolve(σ_5, Δ_4, $\overline{\chi}$, bid, acc, $res_{\text{acc+1}}$) = (σ_6, Δ_5), and $\Delta_6 = \Delta_5.pop()$.

By definition 3.3.1, given $c = iep$, we have $c \simeq_L c'$ if and only if $c' = iep$.

Given $\Sigma \triangleright$ (γ, σ, Δ, $\overline{\chi}$, bid, acc, if (e) s_1 else s_2) $\Downarrow^t_{iep}$ (γ, σ'_6, Δ'_6, $\overline{\chi}$, bid, acc, skip) by rule Location-tracking Private If Else, we have Label(e, γ) = private, (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c'_1}$ (γ, σ'_1, Δ'_1, $\overline{\chi}$, bid, acc, n'), (γ, σ'_1, Δ'_1, $\overline{\chi}$, bid, acc, private int $res_{\text{acc+1}} = n'$) $\Downarrow^t_{c'_2}$ (γ'_1, σ'_2, Δ'_1, $\overline{\chi}'_1$, bid, acc, skip), $\Delta'_2 = \Delta'_1.push([\,])$, $\overline{\chi}'_2 = \overline{\chi}'_1.push([\,])$, ($\gamma'_1$, σ'_2, Δ'_2, $\overline{\chi}'_2$, *then*, acc + 1, s_1) $\Downarrow^t_{c'_3}$ (γ'_2, σ'_3, Δ'_3, $\overline{\chi}'_3$, *then*, acc + 1, skip), T_restore(σ'_3, Δ'_3, acc + 1) = σ'_4, (γ'_1, σ'_4, Δ'_3, $\overline{\chi}'_2$, *else*, acc + 1, s_2) $\Downarrow^t_{c'_4}$ (γ'_3, σ'_5, Δ'_4, $\overline{\chi}'_4$, *else*, acc + 1, skip), T_resolve(σ'_5, Δ'_4, $\overline{\chi}$, bid, acc, $res_{\text{acc+1}}$) = (σ'_6, Δ'_5), and $\Delta'_6 = \Delta'_5.pop()$.

Given (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, n), and (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c'_1}$ (γ, σ'_1, Δ'_1, $\overline{\chi}$, bid, acc, n'), by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, private int $res_{\text{acc+1}} = n$) $\Downarrow^t_{c_2}$ (γ_1, σ_2, Δ_2, $\overline{\chi}_1$, bid, acc, skip), (γ, σ'_1, Δ'_1, $\overline{\chi}$, bid, acc, private int $res_{\text{acc+1}} = n'$) $\Downarrow^t_{c'_2}$ (γ'_1, σ'_2, Δ'_1, $\overline{\chi}_1$, bid, acc, skip), $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, and $n = n'$, we have {private int $res_{\text{acc+1}} = n$} = {private int $res_{\text{acc+1}} = n'$}. By the inductive hypothesis, we have $\gamma_1 = \gamma'_1$, $\sigma_2 = \sigma'_2$, $\overline{\chi}_1 = \overline{\chi}'_1$,and $c_2 \simeq_L c'_2$.

Given $\Delta_2 = \Delta_1.push([\,])$, $\Delta'_2 = \Delta'_1.push([\,])$, and $\Delta_1 = \Delta'_1$, we have $\Delta_2 = \Delta'_2$.

Given $\overline{\chi}'_2 = \overline{\chi}'_1.push([\,])$, $\overline{\chi}_2 = \overline{\chi}_1.push([\,])$, and $\overline{\chi}_1 = \overline{\chi}'_1$, we have $\overline{\chi}_2 = \overline{\chi}'_2$.

Given (γ_1, σ_2, Δ_2, $\overline{\chi}_2$, *then*, acc + 1, s_1) $\Downarrow^t_{c_3}$ (γ_2, σ_3, Δ_3, $\overline{\chi}_3$, *then*, acc + 1, skip), (γ'_1, σ'_2, Δ'_2, $\overline{\chi}'_2$, *then*, acc + 1, s_1) $\Downarrow^t_{c'_3}$ (γ'_2, σ'_3, Δ'_3, $\overline{\chi}'_3$, *then*, acc + 1, skip), $\gamma_1 = \gamma'_1$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, and $\overline{\chi}_2 = \overline{\chi}'_2$, by the inductive hypothesis we have $\gamma_2 = \gamma'_2$, $\sigma_3 = \sigma'_3$, $\Delta_3 = \Delta'_3$, $\overline{\chi}_3 = \overline{\chi}'_3$, and $c_3 \simeq_L c'_3$.

Given T_restore(σ_3, Δ_3, acc + 1) = σ_4, T_restore(σ'_3, Δ'_3, acc + 1) = σ'_4, $\sigma_3 = \sigma'_3$, and $\Delta_3 = \Delta'_3$, by Lemma 4.3.1 we have $\sigma_4 = \sigma'_4$.

Given (γ_1, σ_4, Δ_3, $\overline{\chi}_2$, *else*, acc + 1, s_2) $\Downarrow^t_{c_4}$ (γ_3, σ_5, Δ_4, $\overline{\chi}_4$, *else*, acc + 1, skip), (γ'_1, σ'_4, Δ'_3, $\overline{\chi}'_2$, *else*, acc + 1, s_2)

$\Downarrow^t_{c'_4}$ $(\gamma'_3, \sigma'_5, \Delta'_4, \overline{\chi}'_4, \mathit{else}, \mathrm{acc}+1, \mathrm{skip})$, $\gamma_1 = \gamma'_1$, $\sigma_4 = \sigma'_4$, $\Delta_3 = \Delta'_3$, $\overline{\chi}_2 = \overline{\chi}'_2$, by the inductive hypothesis we have $\gamma_3 = \gamma'_3$, $\sigma_5 = \sigma'_5$, $\Delta_4 = \Delta'_4$, $\overline{\chi}_4 = \overline{\chi}'_4$, and $c_4 \simeq_L c'_4$.

Given $\mathrm{T_resolve}(\sigma_5, \Delta_4, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathit{res}_{\mathrm{acc}+1}) = (\sigma_6, \Delta_5)$, $\mathrm{T_resolve}(\sigma'_5, \Delta'_4, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathit{res}_{\mathrm{acc}+1}) = (\sigma'_6, \Delta'_5)$, $\sigma_5 = \sigma'_5$, and $\Delta_4 = \Delta'_4$, by Lemma 4.3.2 we have $\sigma_6 = \sigma'_6$ and $\Delta_5 = \Delta'_5$.

Given $\Delta_6 = \Delta_5.\mathit{pop}()$, $\Delta'_6 = \Delta'_5.\mathit{pop}()$, and $\Delta_5 = \Delta'_5$, we have $\Delta_6 = \Delta'_6$.

Therefore, we have $\gamma = \gamma$, $\sigma_6 = \sigma'_6$, $\Delta_6 = \Delta'_6$, $\overline{\chi} = \overline{\chi}$, $\mathrm{bid} = \mathrm{bid}$, $\mathrm{acc} = \mathrm{acc}$, $\mathrm{skip} = \mathrm{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \Downarrow^t_{iet} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \Downarrow^t_{iet} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$ by rule Location-tracking If Else True, we have $\mathrm{Label}(e, \gamma) = \mathrm{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n)$, $n \neq 0$, and $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s_1) \Downarrow^t_{c_2} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$.

By definition 3.3.1, given $c = iet$, we have $c \simeq_L c'$ if and only if $c' = iet$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \Downarrow^t_{iet} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$ by rule Location-tracking If Else True, we have $\mathrm{Label}(e, \gamma) = \mathrm{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n')$, $n' \neq 0$, and $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s_1) \Downarrow^t_{c'_2} (\gamma'_1, \sigma'_2, \Delta'_2, \overline{\chi}'_1, \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s_1) \Downarrow^t_{c_2} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s_1) \Downarrow^t_{c'_2} (\gamma'_1, \sigma'_2, \Delta'_2, \overline{\chi}'_1, \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$, $\sigma_1 = \sigma'_1$, and $\Delta_1 = \Delta'_1$, by the inductive hypothesis we have $\gamma_1 = \gamma'_1$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi}_1 = \overline{\chi}'_1$, and $c_2 \simeq_L c'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi} = \overline{\chi}$, $\mathrm{bid} = \mathrm{bid}$, $\mathrm{acc} = \mathrm{acc}$, and $\mathrm{skip} = \mathrm{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow^t_{ief} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \Downarrow^t_{iet} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \&x) \Downarrow^t_{loc} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \&x) \Downarrow^t_{loc} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$ by rule Location-tracking Address Of, we have $\gamma(x) = (l,\ ty)$.

By definition 3.3.1, given $c = loc$, we have $c \simeq_L c'$ if and only if $c' = loc$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \&x) \Downarrow^t_{loc} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l', 0))$ by rule Location-tracking Address Of, we have $\gamma(x) = (l',\ ty')$.

Given $\gamma = \gamma$, we have that $l = l'$ and $ty = ty'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, and $(l, 0) = (l', 0)$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{sizeof}(ty)) \Downarrow^t_{ty} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, n)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{sizeof}(ty)) \Downarrow^t_{ty} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, n)$ by rule Location-tracking Size of Type, we have $n = \tau(ty)$.

By definition 3.3.1, given $c = ty$, we have $c \simeq_L c'$ if and only if $c' = ty$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{sizeof}(ty)) \Downarrow^t_{ty} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, n')$ by rule Location-tracking Size of Type, we have $n' = \tau(ty)$.

Given $n = \tau(ty)$ and $n' = \tau(ty)$, by definition of τ we have $n = n'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, and $n = n'$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{while } (e)\ s) \Downarrow^t_{wle} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \text{acc}, \text{while } (e)\ s) \Downarrow^t_{wle} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking While End, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n)$, $\text{Label}(e, \gamma) = \text{public}$, and $n = 0$.

By definition 3.3.1, given $c = wle$, we have $c \simeq_L c'$ if and only if $c' = wle$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{while } (e)\ s) \Downarrow^t_{wle} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking While End, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n')$, $\text{Label}(e, \gamma) = \text{public}$, and $n' = 0$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{while } (e)\ s) \Downarrow^t_{wlc} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{while } (e)\ s) \Downarrow^t_{wlc} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking While Continue, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n)$, $n \neq 0$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow^t_{c_2} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$, and $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{while } (e)\ s) \Downarrow^t_{c_3} (\gamma_2, \sigma_3, \Delta_3, \overline{\chi}_2, \text{bid}, \text{acc}, \text{skip})$.

By definition 3.3.1, given $c = wlc$, we have $c \simeq_L c'$ if and only if $c' = wlc$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{while } (e)\ s) \Downarrow^t_{wlc} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking While Continue, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n')$, $n' \neq 0$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow^t_{c'_2} (\gamma'_1, \sigma'_2, \Delta'_2, \overline{\chi}'_1, \text{bid}, \text{acc}, \text{skip})$, and $(\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, \text{while } (e)\ s) \Downarrow^t_{c'_3} (\gamma'_2, \sigma'_3, \Delta'_3, \overline{\chi}'_2, \text{bid}, \text{acc}, \text{skip})$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow^t_{c_2} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow^t_{c'_2} (\gamma'_1, \sigma'_2, \Delta'_2, \overline{\chi}'_1, \text{bid}, \text{acc}, \text{skip})$, $\sigma_1 = \sigma'_1$, and $\Delta_1 = \Delta'_1$, by the inductive hypothesis we have $\gamma_1 = \gamma'_1$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi}_1 = \overline{\chi}'_1$, and $c_2 \simeq_L c'_2$.

Given $(\gamma_1, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{while } (e)\ s) \Downarrow^t_{c_3} (\gamma_2, \sigma_3, \Delta_3, \overline{\chi}_2, \text{bid}, \text{acc}, \text{skip})$, $(\gamma'_1, \sigma'_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{while } (e)\ s) \Downarrow^t_{c'_3} (\gamma'_2, \sigma'_3, \Delta'_3, \overline{\chi}'_2, \text{bid}, \text{acc}, \text{skip})$, $\gamma_1 = \gamma'_1$, $\sigma_2 = \sigma'_2$, and $\Delta_2 = \Delta'_2$, by the inductive hypothesis we have $\gamma_2 = \gamma'_2$, $\sigma_3 = \sigma'_3$, $\Delta_3 = \Delta'_3$, $\overline{\chi}_2 = \overline{\chi}'_2$, and $c_3 \simeq_L c'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\Delta_3 = \Delta'_3$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s_1; s_2) \Downarrow^t_{ss} (\gamma_2, \sigma_2, \Delta_2, \overline{\chi}_2, \text{bid}, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s_1; s_2) \Downarrow^t_{ss} (\gamma_2, \sigma_2, \Delta_2, \overline{\chi}_2, \text{bid}, \text{acc}, v)$ by rule Location-tracking Statement Sequencing, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s_1) \Downarrow^t_{c_1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$ and $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, s_2) \Downarrow^t_{c_2} (\gamma_2, \sigma_2, \Delta_2, \overline{\chi}'_2, \text{bid}, \text{acc}, v)$.

By definition 3.3.1, given $c = ss$, we have $c \simeq_L c'$ if and only if $c' = ss$.

Given $\Sigma \triangleright (\gamma, \sigma, \text{acc}, \Delta, \overline{\chi}, \text{bid}, s_1; s_2) \Downarrow^t_{ss} (\gamma'_2, \sigma'_2, \Delta'_2, \overline{\chi}'_2, \text{bid}, \text{acc}, v')$ by rule Location-tracking Statement Sequencing, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s_1) \Downarrow^t_{c'_1} (\gamma'_1, \sigma'_1, \Delta'_1, \overline{\chi}'_1, \text{bid}, \text{acc}, \text{skip})$ and $(\gamma'_1, \sigma'_1, \Delta'_1, \overline{\chi}'_1, \text{bid}, \text{acc}, s_2) \Downarrow^t_{c'_2} (\gamma'_2, \sigma'_2, \Delta'_2, \overline{\chi}'_2, \text{bid}, \text{acc}, v)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s_1) \Downarrow^t_{c_1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s_1) \Downarrow^t_{c'_1} (\gamma'_1, \sigma'_1, \Delta'_1, \overline{\chi}'_1, \text{bid}, \text{acc}, \text{skip})$, by the inductive hypothesis we have $\gamma_1 = \gamma'_1$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\overline{\chi}_1 = \overline{\chi}'_1$, and $c_1 \simeq_L c'_1$.

Given $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, s_2) \Downarrow^t_{c_2} (\gamma_2, \sigma_2, \Delta_2, \overline{\chi}_2, \text{bid}, \text{acc}, v)$, $(\gamma'_1, \sigma'_1, \Delta'_1, \overline{\chi}'_1, \text{bid}, \text{acc}, s_2) \Downarrow^t_{c'_2} (\gamma'_2, \sigma'_2, \Delta'_2, \overline{\chi}'_2, \text{bid}, \text{acc}, v)$, $\gamma_1 = \gamma'_1$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, and $\overline{\chi}_1 = \overline{\chi}'_1$, by the inductive hypothesis we have $\gamma_2 = \gamma'_2$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi}_2 = \overline{\chi}'_2$, $v = v'$, and $c_2 \simeq_L c'_2$.

Therefore, we have $\gamma_2 = \gamma'_2$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi}_2 = \overline{\chi}'_2$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $v = v'$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (e)) \Downarrow^t_{ep} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (e)) \Downarrow^t_{ep} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$ by rule Location-tracking Parentheses, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$.

By definition 3.3.1, given $c = ep$, we have $c \simeq_L c'$ if and only if $c' = ep$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (e)) \Downarrow^t_{ep} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, v')$ by rule Location-tracking Parentheses, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, v')$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, v')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $v = v'$, and $c_1 \simeq_L c'_1$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $v = v'$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \{s\}) \Downarrow^t_{sb} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \{s\}) \Downarrow^t_{sb} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Statement Block, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow^t_{c_1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$.

By definition 3.3.1, given $c = sb$, we have $c \simeq_L c'$ if and only if $c' = sb$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \{s\}) \Downarrow^t_{sb} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Statement Block, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow^t_{c'_1} (\gamma'_1, \sigma'_1, \Delta'_1, \overline{\chi}'_1, \text{bid}, \text{acc}, \text{skip})$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow^t_{c_1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, s) \Downarrow^t_{c'_1} (\gamma'_1, \sigma'_1, \Delta'_1, \overline{\chi}'_1, \text{bid}, \text{acc}, \text{skip})$, by the inductive hypothesis we have $\gamma_1 = \gamma'_1$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\overline{\chi}_1 = \overline{\chi}'_1$, and $c_1 \simeq_L c'_1$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cl} (\gamma, \sigma_3, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cl} (\gamma, \sigma_3, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$ by rule Location-tracking Cast Public Location, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$, $(ty = \text{public}\ bty*) \vee (ty = \text{char}*)$, $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega, \text{void}, n, \text{PermL(Freeable, void, public}, n)\big)\big]$, and $\sigma_3 = \sigma_2\big[l \rightarrow \big(\omega, ty, \frac{n}{\tau(ty)}, \text{PermL(Freeable}, ty, \text{public}, \frac{n}{\tau(ty)})\big)\big]$.

By definition 3.3.1, given $c = cl$, we have $c \simeq_L c'$ if and only if $c' = cl$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cl} (\gamma, \sigma'_3, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, (l', 0))$ by rule Location-tracking Cast Public Location, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, (l', 0))$, $(ty = \text{public}\ bty'*) \vee (ty = \text{char}*)$, $\sigma'_1 = \sigma'_2\big[l' \rightarrow \big(\omega', \text{void}, n', \text{PermL(Freeable, void, public}, n')\big)\big]$, and $\sigma'_3 = \sigma'_2\big[l' \rightarrow \big(\omega', ty, \frac{n'}{\tau(ty)}, \text{PermL(Freeable}, ty, \text{public}, \frac{n'}{\tau(ty)})\big)\big]$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, (l', 0))$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta - 1 = \Delta'_1$, $l = l'$, and $c_1 \simeq_L c'_1$.

Given $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega, \text{void}, n, \text{PermL(Freeable, void, public}, n)\big)\big]$, $\sigma'_1 = \sigma'_2\big[l' \rightarrow \big(\omega', \text{void}, n', \text{PermL(Freeable, void, public}, n')\big)\big]$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\sigma_2 = \sigma'_2$, $\omega = \omega'$, and $n = n'$.

Given $\sigma_3 = \sigma_2\big[l \rightarrow \big(\omega, ty, \frac{n}{\tau(ty)}, \text{PermL(Freeable}, ty, \text{public}, \frac{n}{\tau(ty)})\big)\big]$, $\sigma'_3 = \sigma'_2\big[l' \rightarrow \big(\omega', ty, \frac{n'}{\tau(ty)}, \text{PermL(Freeable}, ty, \text{public}, \frac{n'}{\tau(ty)})\big)\big]$, $\sigma_2 = \sigma'_2$, $l = l'$, $\omega = \omega'$, and $n = n'$, we have $\sigma_3 = \sigma'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\Delta_1 = \Delta'_1$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $(l, 0) = (l', 0)$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cl1} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cl} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cv} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_1)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^t_{cv} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_1)$ by rule Location-tracking Cast Public

Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n)$, $(ty = \text{public int}) \vee (ty = \text{public float})$, and $n_1 = \text{Cast}(\text{public}, ty, n)$.

By definition 3.3.1, given $c = cv$, we have $c \simeq_L c'$ if and only if $c' = cv$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^{t}_{cv} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n'_1)$ by rule Location-tracking Cast Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n')$, $(ty = \text{public int}) \vee (ty = \text{public float})$, and $n'_1 = \text{Cast}(\text{public}, ty, n')$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $n_1 = \text{Cast}(\text{public}, ty, n)$, $n'_1 = \text{Cast}(\text{public}, ty, n')$, and $n = n'$, by definition of Cast, we have $n_1 = n'_1$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $n_1 = n'_1$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^{t}_{cv1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_1)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (ty)\ e) \Downarrow^{t}_{cv} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n_1)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow^{t}_{inp} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow^{t}_{inp} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking SMC Input Public Value, we have $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$, $\text{acc} = 0$, $(\gamma\ \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^{t}_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n)$, $\gamma(x) = (l, \text{public}\ bty)$, $\text{InputValue}(x, n) = n_1$, and $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, x = n_1) \Downarrow^{t}_{c_3} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

By definition 3.3.1, given $c = inp$, we have $c \simeq_L c'$ if and only if $c' = inp$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow^{t}_{inp} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking SMC Input Public Value, we have $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, x')$, $\text{acc} = 0$, $(\gamma\ \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^{t}_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n')$, $\gamma(x') = (l', \text{public}\ bty')$, $\text{InputValue}(x',$

$n') = n'_1$, and $(\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, x' = n'_1) \Downarrow^t_{c'_3} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$ and $(\gamma, \sigma, \text{acc}, \Delta, \overline{\chi}, \text{bid}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, x')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $x = x'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma\, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n)$, $(\gamma\, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n')$, $\sigma_1 = \sigma'_1$, and $\Delta_1 = \Delta'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $n = n'$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x) = (l, \text{public } bty)$, $\gamma(x') = (l', \text{public } bty')$, and $x = x'$, we have $l = l'$ and $bty = bty'$.

Given $\text{InputValue}(x, n) = n_1$, $\text{InputValue}(x', n') = n'_1$, $x = x'$, and $n = n'$, by Axiom 3.3.3 we have $n_1 = n'_1$.

Given $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, x = n_1) \Downarrow^t_{c_3} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$, $(\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, x' = n'_1) \Downarrow^t_{c'_3} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $x = x'$, and $n = n'$, by the inductive hypothesis we have $\sigma_3 = \sigma'_3$, $\Delta_3 = \Delta'_3$, and $c_3 \simeq_L c'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\Delta_3 = \Delta'_3$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow^t_{inp3} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2)) \Downarrow^t_{inp} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp1} (\gamma, \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp1} (\gamma, \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking SMC Input Public 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n)$, $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, e_3) \Downarrow^t_{c_3} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, n_1)$, $\gamma(x) = (l, \text{public const } bty*)$, $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, and $(\gamma, \sigma_3, \Delta_4, \overline{\chi}, \text{bid}, \text{acc}, x = [m_0, ..., m_{n_1}]) \Downarrow^t_{c_4} (\gamma, \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

By definition 3.3.1, given $c = inp1$, we have $c \simeq_L c'$ if and only if $c' = inp1$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp1} (\gamma, \sigma'_4, \Delta'_4, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking SMC Input Public 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, x')$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n')$, $(\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, e_3) \Downarrow^t_{c'_3} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, n'_1)$, $\gamma(x') = (l', \text{public const } bty'*)$, $\text{InputArray}(x', n', n'_1) = [m'_0, ..., m'_{n'_1}]$, and $(\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, x' = [m'_0, ..., m'_{n'_1}]) \Downarrow^t_{c'_4} (\gamma, \sigma'_4, \Delta'_4, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, x')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $x = x'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n')$, $\sigma_1 = \sigma'_1$, and $\Delta_1 = \Delta'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $n = n'$, and $c_2 \simeq_L c'_2$.

Given $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, e_3) \Downarrow^t_{c_3} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, n_1)$, $(\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, e_3) \Downarrow^t_{c'_3} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, n'_1)$, $\sigma_2 = \sigma'_2$, and $\Delta_2 = \Delta'_2$, by the inductive hypothesis we have $\sigma_3 = \sigma'_3$, $\Delta_3 = \Delta'_3$, $n_1 = n'_1$, and $c_3 \simeq_L c'_3$.

Given $\gamma(x) = (l, \text{public const } bty*)$, $\gamma(x') = (l', \text{public const } bty'*)$, and $x = x'$, we have $l = l'$ and $bty = bty'$.

Given $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, $\text{InputArray}(x', n', n'_1) = [m'_0, ..., m'_{n'_1}]$, $x = x'$, $n = n'$, and $n_1 = n'_1$, by Axiom 3.3.4 we have $[m_0, ..., m_{n_1}] = [m'_0, ..., m'_{n'_1}]$. Therefore, we have $\{x = [m_0, ..., m_{n_1}]\} = \{x' = [m'_0, ..., m'_{n'_1}]\}$.

Given $(\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, x = [m_0, ..., m_{n_1}]) \Downarrow^t_{c_4} (\gamma, \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$, $(\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, x' = [m'_0, ..., m'_{n'_1}]) \Downarrow^t_{c'_4} (\gamma, \sigma'_4, \Delta'_4, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$, $\{x = [m_0, ..., m_{n_1}]\} = \{x' = [m'_0, ..., m'_{n'_1}]\}$, $\sigma_3 = \sigma'_3$, and $\Delta_3 = \Delta'_3$, by the inductive hypothesis we have $\sigma_4 = \sigma'_4$, $\Delta_4 = \Delta'_4$, and $c_4 \simeq_L c'_4$.

Therefore, we have $\gamma = \gamma$, $\sigma_4 = \sigma'_4$, $\Delta_4 = \Delta'_4$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp4} (\gamma, \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcinput}(e_1, e_2, e_3)) \Downarrow^t_{inp1} (\gamma, \sigma_4, \Delta_4, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcoutput}(e_1, e_2)) \Downarrow^t_{out} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcoutput}(e_1, e_2)) \Downarrow^t_{out} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking SMC Output Public Value, we have $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n)$, $\gamma(x) = (l, \text{public } bty)$, $\sigma_2(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1))$, $\text{DecodeVal}(\text{public } bty, 1, \omega) = n_1$, and $\text{OutputValue}(x, n, n_1)$.

By definition 3.3.1, given $c = out$, we have $c \simeq_L c'$ if and only if $c' = out$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcoutput}(e_1, e_2)) \Downarrow^t_{out} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking SMC Output Public Value, we have $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, x')$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n')$, $\gamma(x') = (l', \text{public } bty')$, $\sigma'_2(l') = (\omega', \text{public } bty', 1, \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, 1))$, $\text{DecodeVal}(\text{public } bty', 1, \omega') = n'_1$, and $\text{OutputValue}(x', n', n'_1)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, x')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $x = x'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n')$, $\Delta_1 = \Delta'_1$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $n = n'$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x) = (l, \text{public } bty)$, $\gamma(x') = (l', \text{public } bty')$, and $x = x'$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1))$, $\sigma'_2(l') = (\omega', \text{public } bty', 1, \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodeVal}(\text{public } bty, 1, \omega) = n_1$, $\text{DecodeVal}(\text{public } bty', 1, \omega') = n'_1$, $\omega = \omega'$, and $bty = bty'$, by definition of DecodeVal we have $n_1 = n'_1$.

Given $\text{OutputValue}(x, n, n_1)$, $\text{OutputValue}(x', n', n'_1)$, $x = x'$, $n = n'$, and $n_1 = n'_1$, by definition of OutputValue we will have identical output.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcoutput}(e_1, e_2)) \Downarrow^t_{out3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcoutput}(e_1, e_2)) \Downarrow^t_{out} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out1} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out1} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking SMC Output Public 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n)$, $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, e_3) \Downarrow^t_{c_3} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, n_1)$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_3(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable, public const } bty*, \text{public}, 1)$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_3(l_1) = (\omega_1, \text{public } bty, n_1, \text{PermL}(\text{Freeable, public } bty, \text{public}, n_1))$, $\text{DecodeVal}(\text{public } bty, n_1, \omega_1) = [m_0, ..., m_{n_1}]$, and $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$.

By definition 3.3.1, given $c = out1$, we have $c \simeq_L c'$ if and only if $c' = out1$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out1} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking SMC Output Public 1D Array, we have $\text{Label}(e_2, \gamma) = \text{Label}(e_3, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, x')$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n')$, $(\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, e_3) \Downarrow^t_{c'_3} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, n'_1)$, $\gamma(x') = (l', \text{public const } bty'*)$, $\sigma'_3(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable, public const } bty'*, \text{public}, 1)$, $\text{DecodePtr}(\text{public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_3(l'_1) = (\omega'_1, \text{public } bty, n'_1, \text{PermL}(\text{Freeable, public } bty', \text{public}, n'_1))$, $\text{DecodeVal}(\text{public } bty', n'_1, \omega'_1) = [m'_0, ..., m'_{n'_1}]$, and $\text{OutputArray}(x', n', [m'_0, ..., m'_{n'_1}])$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, x)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, x')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $x = x'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, n)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, n')$, $\Delta_1 = \Delta'_1$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $n = n'$, and $c_2 \simeq_L c'_2$.

Given $(\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, e_3) \Downarrow^t_{c_3} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, n_1)$, $(\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, e_3) \Downarrow^t_{c'_3} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, n'_1)$, $\Delta_2 = \Delta'_2$, and $\sigma_2 = \sigma'_2$, by the inductive hypothesis we have $\sigma_3 = \sigma'_3$, $\Delta_3 = \Delta'_3$, $n_1 = n'_1$, and

$c_3 \simeq_L c_3'$.

Given $\gamma(x) = (l, \text{public const } bty*)$, $\gamma(x') = (l', \text{public const } bty'*)$, and $x = x'$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_3(l) = (\omega, \text{public const } bty*, 1, \text{PermL(Freeable, public const } bty*, \text{public, } 1)$, $\sigma_3'(l') = (\omega', \text{public const } bty'*, 1, \text{PermL(Freeable, public const } bty'*, \text{public, } 1)$, $\sigma_3 = \sigma_3'$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr(public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr(public const } bty'*, 1, \omega') = [1, [(l_1', 0)], [1], 1]$, $\omega = \omega'$, and $bty = bty'$, by definition of DecodePtr we have $[1, [(l_1, 0)], [1], 1] = [1, [(l_1', 0)], [1], 1]$ and therefore $l_1 = l_1'$.

Given $\sigma_3(l_1) = (\omega_1, \text{public } bty, n_1, \text{PermL(Freeable, public } bty, \text{public, } n_1))$, $\sigma_3'(l_1') = (\omega_1', \text{public } bty, n_1', \text{PermL(Freeable, public } bty', \text{public, } n_1'))$, $\sigma_3 = \sigma_3'$, and $l_1 = l_1'$, we have $\omega_1 = \omega_1'$ and $n_1 = n_1'$.

Given $\text{DecodeVal(public } bty, n_1, \omega_1) = [m_0, ..., m_{n_1}]$, $\text{DecodeVal(public } bty', n_1', \omega_1') = [m_0', ..., m_{n_1'}']$, $bty = bty'$, $n_1 = n_1'$, and $\omega_1 = \omega_1'$, by definition of DecodeVal we have $[m_0, ..., m_{n_1}] = [m_0', ..., m_{n_1'}']$.

Given $\text{OutputArray}(x, n, [m_0, ..., m_{n_1}])$, $\text{OutputArray}(x', n', [m_0', ..., m_{n_1'}'])$, $x = x'$, $n = n'$, and $[m_0, ..., m_{n_1}] = [m_0', ..., m_{n_1'}']$, by definition of OutputArray we will have identical output.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma_3'$, $\Delta_3 = \Delta_3'$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc, smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out4} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid, acc, skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc, smcoutput}(e_1, e_2, e_3)) \Downarrow^t_{out1} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid, acc, skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc, } ty\ x(\overline{p})) \Downarrow^t_{df} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid, acc, skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc, } ty\ x(\overline{p})) \Downarrow^t_{df} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid, acc, skip})$ by rule Location-tracking Function Declaration, we have $(\text{acc} = 0) \wedge (\text{bid} = none)$, $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, and $\sigma_1 = \sigma[l \rightarrow (\text{NULL}, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})]$.

By definition 3.3.1, given $c = df$, we have $c \simeq_L c'$ if and only if $c' = df$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x(\overline{p})) \Downarrow^t_{df} (\gamma'_1, \sigma'_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Function Declaration, we have $(\text{acc} = 0) \wedge (\text{bid} = none)$, $\text{GetFunTypeList}(\overline{p}) = \overline{ty}'$, $l' = \phi()$, $\gamma'_1 = \gamma[x \rightarrow (l', \overline{ty}' \rightarrow ty)]$, and $\sigma'_1 = \sigma[l' \rightarrow (\text{NULL}, \overline{ty}' \rightarrow ty, 1, \text{PermL_Fun(public)})]$.

Given $\text{GetFunTypeList}(\overline{p}) = \overline{ty}$ and $\text{GetFunTypeList}(\overline{p}) = \overline{ty}'$, by Lemma 3.3.1 we have $\overline{ty} = \overline{ty}'$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \overline{ty} \rightarrow ty)]$, $\gamma'_1 = \gamma[x \rightarrow (l', \overline{ty}' \rightarrow ty)]$, $l = l'$, and $\overline{ty} = \overline{ty}'$, we have $\gamma_1 = \gamma'_1$.

Given $\sigma_1 = \sigma[l \rightarrow (\text{NULL}, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})]$, $\sigma'_1 = \sigma[l' \rightarrow (\text{NULL}, \overline{ty}' \rightarrow ty, 1, \text{PermL_Fun(public)})]$, $l = l'$, and $\overline{ty} = \overline{ty}'$, we have $\sigma_1 = \sigma'_1$.

Therefore, we have $\gamma_1 = \gamma'_1$, $\sigma_1 = \sigma'_1$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x(\overline{p})\{s\}) \Downarrow^t_{fpd} (\gamma, \sigma_2, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x(\overline{p})\{s\}) \Downarrow^t_{fpd} (\gamma, \sigma_2, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Pre-Declared Function Definition, we have $(\text{acc} = 0) \wedge (\text{bid} = none)$, $x \in \gamma$, $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $\text{CheckPublicEffects}(s, x, \gamma, \sigma) = n$, $\text{EncodeFun}(s, n, \overline{p}) = \omega$, $\sigma = \sigma_1[l \rightarrow (\text{NULL}, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})]$, and $\sigma_2 = \sigma_1[l \rightarrow (\omega, \overline{ty} \rightarrow ty, 1, \text{PermL_Fun(public)})]$.

By definition 3.3.1, given $c = fpd$, we have $c \simeq_L c'$ if and only if $c' = fpd$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x(\overline{p})\{s\}) \Downarrow^t_{fpd} (\gamma, \sigma'_2, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Pre-Declared Function Definition, we have $(\text{acc} = 0) \wedge (\text{bid} = none)$, $x \in \gamma$, $\gamma(x) = (l', \overline{ty}' \rightarrow ty)$, $\text{CheckPublicEffects}(s, x, \gamma, \sigma) = n'$, $\text{EncodeFun}(s, n', \overline{p}) = \omega'$, $\sigma = \sigma'_1[l' \rightarrow (\text{NULL}, \overline{ty}' \rightarrow ty, 1, \text{PermL_Fun(public)})]$, and $\sigma'_2 = \sigma'_1[l' \rightarrow (\omega', \overline{ty}' \rightarrow ty, 1, \text{PermL_Fun(public)})]$.

Given $\gamma(x) = (l, \overline{ty} \rightarrow ty)$, $\gamma(x) = (l', \overline{ty}' \rightarrow ty)$, we have $l = l'$ and $\overline{ty} = \overline{ty}'$.

Given $\mathrm{CheckPublicEffects}(s, x, \gamma, \sigma) = n$, $\mathrm{CheckPublicEffects}(s, x, \gamma, \sigma) = n'$, by definition of $\mathrm{CheckPublicEffects}$ we have $n = n'$.

Given $\mathrm{EncodeFun}(s,\, n,\, \overline{p}) = \omega$, $\mathrm{EncodeFun}(s,\, n',\, \overline{p}) = \omega'$, and $n = n'$, by definition of $\mathrm{EncodeFun}$ we have $\omega = \omega'$.

Given $\sigma = \sigma_1[l \rightarrow (\mathrm{NULL},\, \overline{ty} \rightarrow ty,\, 1,\, \mathrm{PermL_Fun(public)})]$, $\sigma = \sigma_1'[l' \rightarrow (\mathrm{NULL},\, \overline{ty}' \rightarrow ty,\, 1,\, \mathrm{PermL_Fun(public)})]$, $l = l'$, and $\overline{ty} = \overline{ty}'$, we have $\sigma_1 = \sigma_1'$.

Given $\sigma_2 = \sigma_1[l \rightarrow (\omega,\, \overline{ty} \rightarrow ty,\, 1, \mathrm{PermL_Fun(public)})]$, $\sigma_2' = \sigma_1'[l' \rightarrow (\omega',\, \overline{ty}' \rightarrow ty,\, 1,\, \mathrm{PermL_Fun(public)})]$, $\sigma_1 = \sigma_1'$, $l = l'$, $\overline{ty} = \overline{ty}'$, and $\omega = \omega'$, we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\mathrm{bid} = \mathrm{bid}$, $\mathrm{acc} = \mathrm{acc}$, $\mathrm{skip} = \mathrm{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, ty\ x(\overline{p})\{s\}) \Downarrow^t_{fd} (\gamma_1,\, \sigma_1,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{skip})$
Given $\Pi \triangleright (\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, ty\ x(\overline{p})\{s\}) \Downarrow^t_{fd} (\gamma_1,\, \sigma_1,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{skip})$ by rule Location-tracking Function Definition, we have $l = \phi()$, $\mathrm{GetFunTypeList}(\overline{p}) = \overline{ty}$, $x \notin \gamma$, $\gamma_1 = \gamma[x \rightarrow (l,\, \overline{ty} \rightarrow ty)]$, $(\mathrm{acc} = 0) \wedge (\mathrm{bid} = \mathit{none})$, $\mathrm{CheckPublicEffects}(s, x, \gamma, \sigma) = n$, $\mathrm{EncodeFun}(s,\, n,\, \overline{p}) = \omega$, and $\sigma_1 = \sigma[l \rightarrow (\omega,\, \overline{ty} \rightarrow ty,\, 1, \mathrm{PermL_Fun(public)})]$.

By definition 3.3.1, given $c = fd$, we have $c \simeq_L c'$ if and only if $c' = fd$.

Given $\Sigma \triangleright (\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, ty\ x(\overline{p})\{s\}) \Downarrow^t_{fd} (\gamma_1',\, \sigma_1',\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{skip})$ by rule Location-tracking Function Definition, we have $l' = \phi()$, $\mathrm{GetFunTypeList}(\overline{p}) = \overline{ty}'$, $x \notin \gamma$, $\gamma_1' = \gamma[x \rightarrow (l',\, \overline{ty}' \rightarrow ty)]$, $(\mathrm{acc} = 0) \wedge (\mathrm{bid} = \mathit{none})$, $\mathrm{CheckPublicEffects}(s,\, x,\, \gamma,\, \sigma) = n'$, $\mathrm{EncodeFun}(s,\, n',\, \overline{p}) = \omega'$, and $\sigma_1' = \sigma[l' \rightarrow (\omega',\, \overline{ty}' \rightarrow ty,\, 1,\, \mathrm{PermL_Fun(public)})]$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\mathrm{GetFunTypeList}(\overline{p}) = \overline{ty}$ and $\mathrm{GetFunTypeList}(\overline{p}) = \overline{ty}'$, by Lemma 3.3.1 we have $\overline{ty} = \overline{ty}'$.

Given $\gamma_1 = \gamma[x \to (l, \overline{ty} \to ty)]$, $\gamma_1' = \gamma[x \to (l', \overline{ty}' \to ty)]$, $l = l'$, and $\overline{ty} = \overline{ty}'$, we have $\gamma_1 = \gamma_1'$.

Given $\mathrm{CheckPublicEffects}(s, x, \gamma, \sigma) = n$, $\mathrm{CheckPublicEffects}(s, x, \gamma, \sigma) = n'$, by definition of $\mathrm{CheckPublicEffects}$ we have $n = n'$.

Given $\mathrm{EncodeFun}(s, n, \overline{p}) = \omega$, $\mathrm{EncodeFun}(s, n', \overline{p}) = \omega'$, and $n = n'$, by definition of $\mathrm{EncodeFun}$ we have $\omega = \omega'$.

Given $\sigma_1 = \sigma[l \to (\omega, \overline{ty} \to ty, 1, \mathrm{PermL_Fun(public)})]$, $\sigma_1' = \sigma[l' \to (\omega', \overline{ty}' \to ty, 1, \mathrm{PermL_Fun(public)})]$, $l = l'$, $\overline{ty} = \overline{ty}'$, and $\omega = \omega'$, we have $\sigma_1 = \sigma_1'$.

Therefore, we have $\gamma_1 = \gamma_1'$, $\sigma_1 = \sigma_1'$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\mathrm{bid} = \mathrm{bid}$, $\mathrm{acc} = \mathrm{acc}$, $\mathrm{skip} = \mathrm{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, x(\overline{e})) \Downarrow^t_{fc} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{NULL})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, x(\overline{e})) \Downarrow^t_{fc} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{NULL})$ by rule Location-tracking Function Call No Return With Public Side Effects, we have $\gamma(x) = (l, \overline{ty} \to ty)$, $\sigma(l) = (\omega, \overline{ty} \to ty, 1, \mathrm{PermL_Fun(public)})$, $\mathrm{DecodeFun}(\omega) = (s, 1, \overline{p})$, $\mathrm{acc} = 0$, $\mathrm{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$, $(\gamma, \sigma, \mathrm{acc}, \Delta, \overline{\chi}, \mathrm{bid}, s_1) \Downarrow^t_{c_1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$, and $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \mathrm{bid}, \mathrm{acc}, s) \Downarrow^t_{c_2} (\gamma_2, \sigma_2, \Delta_2, \overline{\chi}_2, \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$.

By definition 3.3.1, given $c = fc$, we have $c \simeq_L c'$ if and only if $c' = fc$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, x(\overline{e})) \Downarrow^t_{fc} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{NULL})$ by rule Location-tracking Function Call No Return With Public Side Effects, we have $\gamma(x) = (l', \overline{ty}' \to ty')$, $\sigma(l') = (\omega', \overline{ty}' \to ty', 1, \mathrm{PermL_Fun(public)})$, $\mathrm{DecodeFun}(\omega') = (s', 1, \overline{p}')$, $\mathrm{acc} = 0$, $\mathrm{GetFunParamAssign}(\overline{p}', \overline{e}) = s_1'$, $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s_1') \Downarrow^t_{c_1'} (\gamma_1', \sigma_1', \Delta_1', \overline{\chi}_1', \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$, and $(\gamma_1', \sigma_1', \Delta_1', \overline{\chi}_1', \mathrm{bid}, \mathrm{acc}, s) \Downarrow^t_{c_2'} (\gamma_2', \sigma_2', \Delta_2', \overline{\chi}_2', \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$.

Given $\gamma(x) = (l, \overline{ty} \to ty)$ and $\gamma(x) = (l', \overline{ty}' \to ty')$, we have $l = l'$, $\overline{ty} = \overline{ty}'$, and $ty = ty'$.

Given $\sigma(l) = (\omega, \overline{ty} \to ty, 1, \mathrm{PermL_Fun(public)})$, $\sigma(l') = (\omega', \overline{ty}' \to ty', 1, \mathrm{PermL_Fun(public)})$, and $l = l'$, we have $\omega = \omega'$.

Given $\mathrm{DecodeFun}(\omega) = (s, 1, \overline{p})$, $\mathrm{DecodeFun}(\omega') = (s', 1, \overline{p}')$, and $\omega = \omega'$, by definition of DecodeFun we have $s = s'$ and $\overline{p} = \overline{p}'$.

Given $\mathrm{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$, $\mathrm{GetFunParamAssign}(\overline{p}', \overline{e}) = s_1'$, and $\overline{p} = \overline{p}'$, by Lemma 3.3.2 we have $s_1 = s_1'$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s_1) \Downarrow^t_{c_1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$, $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, s_1') \Downarrow^t_{c_1'} (\gamma_1', \sigma_1', \Delta_1', \overline{\chi}_1', \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$, and $s_1 = s_1'$, by the inductive hypothesis we have $\gamma_1 = \gamma_1'$, $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $\overline{\chi}_1 = \overline{\chi}_1'$, and $c_1 \simeq_L c_1'$.

Given $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \mathrm{bid}, \mathrm{acc}, s) \Downarrow^t_{c_2} (\gamma_2, \sigma_2, \Delta_2, \overline{\chi}_2, \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$, $(\gamma_1', \sigma_1', \Delta_1', \overline{\chi}_1', \mathrm{bid}, \mathrm{acc}, s) \Downarrow^t_{c_2'} (\gamma_2', \sigma_2', \Delta_2', \overline{\chi}_2', \mathrm{bid}, \mathrm{acc}, \mathrm{skip})$, $\Delta_1 = \Delta_1'$, $\overline{\chi}_1 = \overline{\chi}_1'$, $\gamma_1 = \gamma_1'$, and $\sigma_1 = \sigma_1'$, by the inductive hypothesis we have $\gamma_2 = \gamma_2'$, $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $\overline{\chi}_2 = \overline{\chi}_2'$, and $c_2 \simeq_L c_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $\overline{\chi} = \overline{\chi}$, $\mathrm{bid} = \mathrm{bid}$, $\mathrm{acc} = \mathrm{acc}$, $\mathrm{NULL} = \mathrm{NULL}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, x(\overline{e})) \Downarrow^t_{fc1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{NULL})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, x(\overline{e})) \Downarrow^t_{fc} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{NULL})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{malloc}(e)) \Downarrow^t_{mal} (\gamma, \sigma_2, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, (l, 0))$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{malloc}(e)) \Downarrow^t_{mal} (\gamma, \sigma_2, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, (l, 0))$ by rule Location-tracking Public Malloc, we have $\mathrm{Label}(e, \gamma) = \mathrm{public}$, $(\mathrm{acc} = 0) \wedge (\mathrm{bid} = none)$, $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n)$, $l = \phi()$, and $\sigma_2 = \sigma_1\big[l \rightarrow \big(\mathrm{NULL}, \mathrm{void}*, n, \big[(0, \mathrm{public}, \mathrm{Freeable}), ..., (n-1, \mathrm{public}, \mathrm{Freeable})\big]\big)\big]$.

By definition 3.3.1, given $c = mal$, we have $c \simeq_L c'$ if and only if $c' = mal$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, \mathrm{malloc}(e)) \Downarrow^t_{mal} (\gamma, \sigma_2', \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, (l', 0))$ by rule Location-tracking Public Malloc, we have $\mathrm{Label}(e, \gamma) = \mathrm{public}$, $(\mathrm{acc} = 0) \wedge (\mathrm{bid} = none)$, $(\gamma, \sigma, \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, e) \Downarrow^t_{c_1'} (\gamma, \sigma_1', \Delta, \overline{\chi}, \mathrm{bid}, \mathrm{acc}, n')$, $l' = \phi()$, and $\sigma_2' = \sigma_1'\big[l' \rightarrow \big(\mathrm{NULL}, \mathrm{void}*, n', \big[(0, \mathrm{public}, \mathrm{Freeable}), ..., (n'-1, \mathrm{public}, \mathrm{Freeable})\big]\big)\big]$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, n)$ and $(\gamma, \sigma, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n, \big[(0, \text{public}, \text{Freeable}), ..., (n-1, \text{public}, \text{Freeable})\big]\big)\big]$, $\sigma'_2 = \sigma'_1\big[l' \rightarrow \big(\text{NULL}, \text{void}*, n', \big[(0, \text{public}, \text{Freeable}), ..., (n'-1, \text{public}, \text{Freeable})\big]\big)\big]$, $\sigma_1 = \sigma'_1$, $l = l'$, and $n = n'$, we have $\sigma_2 = \sigma'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $(l, 0) = (l', 0)$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{pmalloc}(e,\ ty)) \Downarrow^t_{malp} (\gamma, \sigma_2, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{pmalloc}(e,\ ty)) \Downarrow^t_{malp} (\gamma, \sigma_2, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l, 0))$ by rule Location-tracking Private Malloc, we have $\text{Label}(e, \gamma) = \text{public}$, $(ty = \text{private int}) \vee (ty = \text{private float})$ $(\gamma, \sigma, \text{acc}, \Delta, \overline{\chi}, \text{bid},\ e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, n)$, $(\text{acc} = 0) \wedge (\text{bid} = \mathit{none})$, $l = \phi()$, and $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL},\ ty, n, \text{PermL}(\text{Freeable}, ty, \text{private}, n)\big]$.

By definition 3.3.1, given $c = \mathit{malp}$, we have $c \simeq_L c'$ if and only if $c' = \mathit{malp}$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{pmalloc}(e,\ ty)) \Downarrow^t_{malp} (\gamma, \sigma'_2, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l', 0))$ by rule Location-tracking Private Malloc, we have $\text{Label}(e, \gamma) = \text{public}$, $(ty = \text{private int}) \vee (ty = \text{private float})$ $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, n')$, $(\text{acc} = 0) \wedge (\text{bid} = \mathit{none})$, $l' = \phi()$, and $\sigma'_2 = \sigma'_1\big[l' \rightarrow \big(\text{NULL},\ ty, n', \text{PermL}(\text{Freeable}, ty, \text{private}, n')\big]$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, n)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL},\ ty, n, \text{PermL}(\text{Freeable}, ty, \text{private}, n)\big]$, $\sigma'_2 = \sigma'_1\big[l' \rightarrow \big(\text{NULL},\ ty, n',$

$\mathrm{PermL}(\mathrm{Freeable},\, ty,\, \mathrm{private},\, n')\big]$, $\sigma_1 = \sigma_1'$, $l = l'$, and $n = n'$, we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\mathrm{bid} = \mathrm{bid}$, $\mathrm{acc} = \mathrm{acc}$, $(l, 0) = (l', 0)$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{free}(e)) \Downarrow^t_{fre} (\gamma,\, \sigma_2,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{skip})$
Given $\Pi \triangleright (\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{free}(e)) \Downarrow^t_{fre} (\gamma,\, \sigma_2,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{skip})$ by rule Location-tracking Public Free, we have $(\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, e) \;\Downarrow^t_{c_1} (\gamma,\, \sigma_1,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, x)$, $\gamma(x) = (l,\, \mathrm{public}\ bty*)$, $(bty = \mathrm{int}) \vee (bty = \mathrm{float}) \vee (bty = \mathrm{char}) \vee (bty = \mathrm{void})$, $(\mathrm{acc} = 0) \wedge (\mathrm{bid} = none)$, and $\mathrm{Free}(\sigma_1, l, \gamma) = \sigma_2$.

By definition 3.3.1, given $c = fre$, we have $c \simeq_L c'$ if and only if $c' = fre$.

Given $\Sigma \triangleright (\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{free}(e)) \Downarrow^t_{fre} (\gamma,\, \sigma_2',\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{skip})$ by rule Location-tracking Public Free, we have $(\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, e) \;\Downarrow^t_{c_1'} (\gamma,\, \sigma_1',\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, x')$, $\gamma(x') = (l',\, \mathrm{public}\ bty'*)$, $(bty' = \mathrm{int}) \vee (bty' = \mathrm{float}) \vee (bty' = \mathrm{char}) \vee (bty' = \mathrm{void})$, $(\mathrm{acc} = 0) \wedge (\mathrm{bid} = none)$, and $\mathrm{Free}(\sigma_1', l', \gamma) = \sigma_2'$.

Given $(\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, e) \;\Downarrow^t_{c_1} (\gamma,\, \sigma_1,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, x)$ and $(\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, e) \;\Downarrow^t_{c_1'} (\gamma,\, \sigma_1',\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, x')$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $x = x'$, and $c_1 \simeq_L c_1'$.

Given $\gamma(x) = (l,\, \mathrm{public}\ bty*)$, $\gamma(x') = (l',\, \mathrm{public}\ bty'*)$, and $x = x'$, we have $l = l'$ and $bty = bty'$.

Given $\mathrm{Free}(\sigma_1, l, \gamma) = \sigma_2$, $\mathrm{Free}(\sigma_1', l', \gamma) = \sigma_2'$, $\sigma_1 = \sigma_1'$, and $l = l'$, by Lemma 3.3.9 we have $\sigma_2 = \sigma_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\mathrm{bid} = \mathrm{bid}$, $\mathrm{acc} = \mathrm{acc}$, $\mathrm{skip} = \mathrm{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{pfree}(e)) \Downarrow^t_{frep} (\gamma,\, \sigma_2,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{skip})$
Given $\Pi \triangleright (\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{pfree}(e)) \Downarrow^t_{frep} (\gamma,\, \sigma_2,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, \mathrm{skip})$ by rule Location-tracking Private Free, we have $(\gamma,\, \sigma,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, e) \;\Downarrow^t_{c_1} (\gamma,\, \sigma_1,\, \Delta,\, \overline{\chi},\, \mathrm{bid},\, \mathrm{acc},\, x)$, $\gamma(x) = (l,\, \mathrm{private}\ bty*)$, $(\mathrm{acc} = 0) \wedge (\mathrm{bid} = none)$, $(bty = \mathrm{int}) \vee (bty = \mathrm{float})$, and $\mathrm{PFree}(\gamma,\, \sigma_1,\, l) = (\sigma_2,\, \overline{j})$.

By definition 3.3.1, given $c = \mathit{frep}$, we have $c \simeq_L c'$ if and only if $c' = \mathit{frep}$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{pfree}(e)) \Downarrow^t_{\mathit{frep}} (\gamma, \sigma'_2, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Free, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x')$, $\gamma(x') = (l', \text{ private } \mathit{bty}'*)$, $(\text{acc} = 0) \wedge (\text{bid} = \mathit{none})$, $(\mathit{bty}' = \text{int}) \vee (\mathit{bty}' = \text{float})$, and $\text{PFree}(\gamma, \sigma'_1, l') = (\sigma'_2, \overline{j}')$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $x = x'$, and $c_1 \simeq_L c'_1$.

Given $\gamma(x) = (l, \text{ private } \mathit{bty}*)$, $\gamma(x') = (l', \text{ private } \mathit{bty}'*)$, and $x = x'$, we have $l = l'$ and $\mathit{bty} = \mathit{bty}'$.

Given $\text{PFree}(\gamma, \sigma_1, l) = (\sigma_2, \overline{l}, \overline{j})$, $\text{PFree}(\gamma, \sigma'_1, l') = (\sigma'_2, \overline{l}', \overline{j}')$, $\sigma_1 = \sigma'_1$, and $l = l'$, by Lemma 3.3.10 we have $\sigma_2 = \sigma'_2$, $\overline{l} = \overline{l}'$, and $\overline{j} = \overline{j}'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \mathit{ty}\ x = e) \Downarrow^t_{ds} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \mathit{ty}\ x = e) \Downarrow^t_{ds} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Declaration Assignment, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \mathit{ty}\ x) \Downarrow^t_{c_1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$, and $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, x = e) \Downarrow^t_{c_2} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$.

By definition 3.3.1, given $c = ds$, we have $c \simeq_L c'$ if and only if $c' = ds$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \mathit{ty}\ x = e) \Downarrow^t_{ds} (\gamma'_1, \sigma'_2, \Delta'_2, \overline{\chi}'_1, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Declaration Assignment, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \mathit{ty}\ x) \Downarrow^t_{c'_1} (\gamma'_1, \sigma'_1, \Delta'_1, \overline{\chi}'_1, \text{bid}, \text{acc}, \text{skip})$, and $(\gamma'_1, \sigma'_1, \Delta'_1, \overline{\chi}'_1, \text{bid}, \text{acc}, x = e) \Downarrow^t_{c'_2} (\gamma'_1, \sigma'_2, \Delta'_2, \overline{\chi}'_1, \text{bid}, \text{acc}, \text{skip})$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \mathit{ty}\ x) \Downarrow^t_{c_1} (\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \mathit{ty}\ x) \Downarrow^t_{c'_1} (\gamma'_1, \sigma'_1, \Delta'_1, \overline{\chi}'_1, \text{bid}, \text{acc}, \text{skip})$, by the inductive hypothesis we have $\gamma_1 = \gamma'_1$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\overline{\chi}_1 = \overline{\chi}'_1$, and $c_1 \simeq_L c'_1$.

Given $(\gamma_1, \sigma_1, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, x = e) \Downarrow^t_{c_2} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$, $(\gamma'_1, \sigma'_1, \Delta'_1, \overline{\chi}'_1, \text{bid}, \text{acc}, x = e) \Downarrow^t_{c'_2}$

$(\gamma_1', \sigma_2', \Delta_2', \overline{\chi}_1', \text{bid}, \text{acc}, \text{skip})$, $\gamma_1 = \gamma_1'$, $\Delta_1 = \Delta_1'$, $\overline{\chi}_1 = \overline{\chi}_1'$, and $\sigma_1 = \sigma_1'$, by the inductive hypothesis we have $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, and $c_2 \simeq_L c_2'$.

Therefore, we have $\gamma_1 = \gamma_1'$, $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $\overline{\chi}_1 = \overline{\chi}_1'$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e_1] = e_2) \Downarrow^t_{das} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$
This case is similar to case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x = e) \Downarrow^t_{ds} (\gamma_1, \sigma_2, \Delta_2, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_d (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_d (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public Declaration, we have $(ty = \text{public}\ bty) \vee (ty = \text{char})$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\omega = \text{EncodeVal}(ty, \text{NULL})$, and $\sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$.

By definition 3.3.1, given $c = d$, we have $c \simeq_L c'$ if and only if $c' = d$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^t_d (\gamma_1', \sigma_1', \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public Declaration, we have $(ty = \text{public}\ bty) \vee (ty = \text{char})$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, $l' = \phi()$, $\gamma_1' = \gamma[x \rightarrow (l', ty)]$, $\omega' = \text{EncodeVal}(ty, \text{NULL})$, and $\sigma_1' = \sigma[l' \rightarrow (\omega', ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\gamma_1' = \gamma[x \rightarrow (l', ty)]$, and $l = l'$, we have $\gamma_1 = \gamma_1'$.

Given $\omega = \text{EncodeVal}(ty, \text{NULL})$ and $\omega' = \text{EncodeVal}(ty, \text{NULL})$, by Lemma 3.3.3 we have $\omega = \omega'$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$, $\sigma_1' = \sigma[l' \rightarrow (\omega', ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$, $l = l'$, and $\omega = \omega'$, we have $\sigma_1 = \sigma_1'$.

Therefore, we have $\gamma_1 = \gamma_1'$, $\sigma_1 = \sigma_1'$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{d1} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{d} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{d2} (\gamma_1, \sigma_1, \Delta, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{d} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Declaration (Inside a Private - Conditioned If Else Branch), we have $((ty = bty) \vee (ty = \text{private}\ bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $(\text{acc} > 0) \wedge ((\text{bid} = then) \vee (\text{bid} = else))$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\omega = \text{EncodeVal}(ty, \text{NULL})$, $\sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{private}, 1))]$, and $\overline{\chi}_1 = l :: \overline{\chi}[\text{acc}]$.

By definition 3.3.1, given $c = d$, we have $c \simeq_L c'$ if and only if $c' = d$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{d} (\gamma'_1, \sigma'_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Declaration (Inside a Private - Conditioned If Else Branch), we have $((ty = bty) \vee (ty = \text{private}\ bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $(\text{acc} > 0) \wedge ((\text{bid} = then) \vee (\text{bid} = else))$, $l' = \phi()$, $\gamma'_1 = \gamma[x \rightarrow (l', ty)]$, $\omega' = \text{EncodeVal}(ty, \text{NULL})$, $\sigma'_1 = \sigma[l' \rightarrow (\omega', ty, 1, \text{PermL}(\text{Freeable}, ty, \text{private}, 1))]$, and $\overline{\chi}'_1 = l' :: \overline{\chi}[\text{acc}]$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\gamma'_1 = \gamma[x \rightarrow (l', ty)]$, and $l = l'$, we have $\gamma_1 = \gamma'_1$.

Given $\omega = \text{EncodeVal}(ty, \text{NULL})$ and $\omega' = \text{EncodeVal}(ty, \text{NULL})$, by Lemma 3.3.3 we have $\omega = \omega'$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$, $\sigma'_1 = \sigma[l' \rightarrow (\omega', ty, 1, \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$, $l = l'$, and $\omega = \omega'$, we have $\sigma_1 = \sigma'_1$.

Given $\overline{\chi}_1 = l :: \overline{\chi}[\text{acc}]$, $\overline{\chi}'_1 = l' :: \overline{\chi}[\text{acc}]$, and $l = l'$, we have $\overline{\chi}_1 = \overline{\chi}'_1$.

Therefore, we have $\gamma_1 = \gamma'_1$, $\sigma_1 = \sigma'_1$, $\Delta = \Delta$, $\overline{\chi}_1 = \overline{\chi}'_1$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{dp} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{dp} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public Pointer Declaration, we have $(ty = \text{public}\ bty*) \vee ((ty = bty*) \wedge ((bty = \text{char}) \vee (bty = \text{void})))$, $\text{GetIndirection}(*) = i$, $\text{acc} = 0$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\omega = \text{EncodePtr}(ty, [1, [l_{default}], [1], i])$, and $\sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1))]$.

By definition 3.3.1, given $c = dp$, we have $c \simeq_L c'$ if and only if $c' = dp$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{dp} (\gamma'_1, \sigma'_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public Pointer Declaration, we have $(ty = \text{public}\ bty'*) \vee ((ty = bty'*) \wedge ((bty' = \text{char}) \vee (bty' = \text{void})))$, $\text{GetIndirection}(*) = i'$, $\text{acc} = 0$, $l' = \phi()$, $\gamma'_1 = \gamma[x \rightarrow (l', ty)]$, $\omega' = \text{EncodePtr}(ty, [1, [l_{default}], [1], i'])$, and $\sigma'_1 = \sigma[l' \rightarrow (\omega', ty, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1))]$.

Given $(ty = \text{public}\ bty*) \vee ((ty = bty*) \wedge ((bty = \text{char}) \vee (bty = \text{void})))$ and $(ty = \text{public}\ bty'*) \vee ((ty = bty'*) \wedge ((bty' = \text{char}) \vee (bty' = \text{void})))$, we have $bty = bty'$.

Given $\text{GetIndirection}(*) = i$ and $\text{GetIndirection}(*) = i'$, by Lemma 3.3.11 we have $i = i'$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\gamma'_1 = \gamma[x \rightarrow (l', ty)]$, and $l = l'$, we have $\gamma_1 = \gamma'_1$.

Given $\omega = \text{EncodePtr}(ty, [1, [l_{default}], [1], i])$, $\omega' = \text{EncodePtr}(ty, [1, [l_{default}], [1], i'])$, and $i = i'$, by Lemma 3.3.5 we have $\omega = \omega'$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, \text{public}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1))]$, $\sigma'_1 = \sigma[l' \rightarrow (\omega', \text{public}\ bty'*, 1, \text{PermL}(\text{Freeable}, \text{public}\ bty'*, \text{public}, 1))]$, $l = l'$, $\omega = \omega'$, and $bty = bty'$, we have $\sigma_1 = \sigma'_1$.

Therefore, we have $\gamma_1 = \gamma'_1$, $\sigma_1 = \sigma'_1$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{dp1} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{dp} (\gamma_1, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{dp2} (\gamma_1, \sigma_1, \Delta, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{dp} (\gamma_1, \sigma_1, \Delta, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Pointer Declaration (Inside a Private - Conditioned If Else Branch), we have $((ty = bty*) \vee (ty = \text{private } bty*)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $\text{GetIndirection}(*) = i$, $(\text{acc} > 0) \wedge ((\text{bid} = then) \vee (\text{bid} = else))$, $l = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\omega = \text{EncodePtr}(ty, [1, [l_{default}], [1], i])$, $\sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))]$, and $\overline{\chi}_1 = l :: \overline{\chi}[\text{acc}]$.

By definition 3.3.1, given $c = dp$, we have $c \simeq_L c'$ if and only if $c' = dp$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x) \Downarrow^{t}_{dp} (\gamma'_1, \sigma'_1, \Delta, \overline{\chi}'_1, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Pointer Declaration (Inside a Private - Conditioned If Else Branch), we have $((ty = bty'*) \vee (ty = \text{private } bty'*)) \wedge ((bty' = \text{int}) \vee (bty' = \text{float}))$, $\text{GetIndirection}(*) = i'$, $(\text{acc} > 0) \wedge ((\text{bid} = then) \vee (\text{bid} = else))$, $l' = \phi()$, $\gamma'_1 = \gamma[x \rightarrow (l', ty)]$, $\omega' = \text{EncodePtr}(ty, [1, [l_{default}], [1], i'])$, and $\sigma'_1 = \sigma[l' \rightarrow (\omega', ty, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))]$, and $\overline{\chi}'_1 = l' :: \overline{\chi}[\text{acc}]$.

Given $((ty = bty*) \vee (ty = \text{private } bty*)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$ and $((ty = bty'*) \vee (ty = \text{private } bty'*)) \wedge ((bty' = \text{int}) \vee (bty' = \text{float}))$, we have $bty = bty'$.

Given $\text{GetIndirection}(*) = i$ and $\text{GetIndirection}(*) = i'$, by Lemma 3.3.11 we have $i = i'$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $\gamma'_1 = \gamma[x \rightarrow (l', ty)]$, and $l = l'$, we have $\gamma_1 = \gamma'_1$.

Given $\omega = \text{EncodePtr}(ty, [1, [l_{default}], [1], i])$, $\omega' = \text{EncodePtr}(ty, [1, [l_{default}], [1], i'])$, and $i = i'$, by Lemma 3.3.5 we have $\omega = \omega'$.

Given $\sigma_1 = \sigma[l \rightarrow (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))]$, $\sigma'_1 = \sigma[l' \rightarrow (\omega', \text{private } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, 1))]$, $l = l'$, $\omega = \omega'$, and $bty = bty'$, we have $\sigma_1 = \sigma'_1$.

Given $\overline{\chi}_1 = l :: \overline{\chi}[\text{acc}]$, $\overline{\chi}'_1 = l' :: \overline{\chi}[\text{acc}]$, and $l = l'$, we have $\overline{\chi}_1 = \overline{\chi}'_1$.

Therefore, we have $\gamma_1 = \gamma'_1$, $\sigma_1 = \sigma'_1$, $\Delta = \Delta$, $\overline{\chi}_1 = \overline{\chi}'_1$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow_r^t (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow_r^t (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$ by rule Location-tracking Read Public Variable, we have $\gamma(x) = (l, \text{public } bty)$, $\sigma(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1))$, and $\text{DecodeVal}(\text{public } bty, 1, \omega) = v$.

By definition 3.3.1, given $c = r$, we have $c \simeq_L c'$ if and only if $c' = r$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow_r^t (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v')$ by rule Location-tracking Read Public Variable, we have $\gamma(x) = (l', \text{public } bty')$, $\sigma(l') = (\omega', \text{public } bty', 1, \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, 1))$, and $\text{DecodeVal}(\text{public } bty', 1, \omega') = v'$.

Given $\gamma(x) = (l, \text{public } bty)$ and $\gamma(x) = (l', \text{public } bty')$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{public } bty, 1, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1))$, $\sigma(l') = (\omega', \text{public } bty', 1, \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodeVal}(\text{public } bty, 1, \omega) = v$, $\text{DecodeVal}(\text{public } bty', 1, \omega') = v'$, $bty = bty'$, and $\omega = \omega'$, we have $v = v'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $v = v$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow_{r1}^t (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$

This case is similar to case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow_r^t (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_w (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_w (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public Write Variable, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$, $v \neq \text{skip}$, $\text{acc} = 0$, $\gamma(x) = (l, \text{public } bty)$, and $\text{T_UpdateVal}(\sigma_1, l, v, \Delta_1, \overline{\chi}, \text{bid}, \text{public } bty) = (\sigma_2, \Delta_2)$.

By definition 3.3.1, given $c = w$, we have $c \simeq_L c'$ if and only if $c' = w$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_w (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Write Public Variable, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, v')$, $v' \neq \text{skip}$, $\text{acc} = 0$, $\gamma(x) = (l', \text{public } bty')$, and $\text{T_UpdateVal}(\sigma'_1, l', v', \Delta'_1, \overline{\chi}, \text{bid}, \text{public } bty') = (\sigma'_2, \Delta'_2)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, v')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $v = v'$, and $c_1 \simeq_L c'_1$.

Given $\gamma(x) = (l, \text{public } bty)$ and $\gamma(x) = (l', \text{public } bty')$, we have $l = l'$ and $bty = bty'$.

Given $\text{T_UpdateVal}(\sigma_1, l, v, \Delta, \overline{\chi}, \text{bid}, \text{public } bty) = (\sigma_2, \Delta_2)$, $\text{T_UpdateVal}(\sigma'_1, l', v', \Delta, \overline{\chi}, \text{bid}, \text{public } bty') = (\sigma'_2, \Delta'_2)$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $l = l'$, $bty = bty'$, and $v = v'$, by Lemma 3.3.12 we have $\sigma_2 = \sigma'_2$ and $\Delta_2 = \Delta'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{w2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_w (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{w1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{w1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Write Private Variable Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n)$, $\gamma(x) = (l, \text{private } bty)$, and $\text{T_UpdateVal}(\sigma_1, l, \text{encrypt}(n), \Delta_1, \overline{\chi}, \text{bid}, \text{private } bty) = (\sigma_2, \Delta_2)$.

By definition 3.3.1, given $c = w1$, we have $c \simeq_L c'$ if and only if $c' = w1$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{w1} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Write Private Variable Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n')$, $\gamma(x) = (l', \text{private } bty')$, and $\text{T_UpdateVal}(\sigma'_1, l', \text{encrypt}(n'), \Delta'_1, \overline{\chi}, \text{bid}, \text{private } bty') = (\sigma'_2, \Delta'_2)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $\gamma(x) = (l, \text{private } bty)$ and $\gamma(x) = (l', \text{private } bty')$, we have $l = l'$ and $bty = bty'$.

Given $\text{T_UpdateVal}(\sigma_1, l, \text{encrypt}(n), \Delta_1, \overline{\chi}, \text{bid}, \text{private } bty) = (\sigma_2, \Delta_2)$, $\text{T_UpdateVal}(\sigma'_1, l', \text{encrypt}(n'), \Delta'_1, \overline{\chi}, \text{bid}, \text{private } bty') = (\sigma'_2, \Delta'_2)$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $l = l'$, $bty = bty'$, and $n = n'$, by Axiom 3.3.1 we have $\text{encrypt}(n) = \text{encrypt}(n')$ and therefore by Lemma 3.3.12 we have $\sigma_2 = \sigma'_2$ and $\Delta_2 = \Delta'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l_1, \mu_1))$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l_1, \mu_1))$ by rule Location-tracking Public Pointer Read Single Location, we have $\gamma(x) = (l, \text{public } bty*)$, $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, and $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$.

By definition 3.3.1, given $c = rp$, we have $c \simeq_L c'$ if and only if $c' = rp$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l'_1, \mu'_1))$ by rule Location-tracking Public Pointer Read Single Location, we have $\gamma(x) = (l', \text{public } bty'*)$, $\sigma(l') = (\omega', \text{public } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, and $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l'_1, \mu'_1)], [1], i']$.

Given $\gamma(x) = (l, \text{public } bty*)$ and $\gamma(x) = (l', \text{public } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\sigma(l') = (\omega', \text{public } bty'*, 1,$

PermL(Freeable, public $bty'*$, public, 1)), and $l = l'$, we have $\omega = \omega'$.

Given DecodePtr(public $bty*$, 1, ω) = [1, [(l_1, μ_1)], [1], i], DecodePtr(public $bty'*$, 1, ω') = [1, [(l'_1, μ'_1)], [1], i'], $bty = bty'$, and $\omega = \omega'$, we have $l_1 = l'_1$, $\mu_1 = \mu'_1$, and $i = i'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, $(l_1, \mu_1) = (l'_1, \mu'_1)$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp2} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l_1, \mu_1))$
This case is similar to case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l_1, \mu_1))$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp1} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \overline{l}, \overline{j}, i])$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp1} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \overline{l}, \overline{j}, i])$ by rule Location-tracking Private Pointer Read Multiple Locations, we have $\gamma(x) = (l,$ private $bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega,$ private $bty*,\ \alpha$, PermL(Freeable, private $bty*$, private, α)), and DecodePtr(private $bty*$, $\alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$.

By definition 3.3.1, given $c = rp1$, we have $c \simeq_L c'$ if and only if $c' = rp1$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{rp1} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [\alpha', \overline{l}', \overline{j}', i'])$ by rule Location-tracking Private Pointer Read Multiple Locations, we have $\gamma(x) = (l'$, private $bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma(l') = (\omega'$, private $bty'*$, α', PermL(Freeable, private $bty'*$, private, α')), and DecodePtr(private $bty'*$, $\alpha', \omega') = [\alpha', \overline{l}', \overline{j}', i']$.

Given $\gamma(x) = (l$, private $bty*)$ and $\gamma(x) = (l'$, private $bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega$, private $bty*$, α, PermL(Freeable, private $bty*$, private, α)), $\sigma(l') = (\omega'$, private $bty'*$, α', PermL(Freeable, private $bty'*$, private, α')), and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given DecodePtr(private $bty*$, $\alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$, DecodePtr(private $bty'*$, $\alpha', \omega') = [\alpha', \overline{l}', \overline{j}', i']$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \overline{l}, \overline{j}, i] = [\alpha', \overline{l}', \overline{j}', i']$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, $[\alpha, \overline{l}, \overline{j}, i] = [\alpha', \overline{l}', \overline{j}', i']$, and, by definition

4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wp} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wp} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public Pointer Write Single Location, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_e, \mu_e))$, $\gamma(x) = (l, \text{public } bty*)$, $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\text{acc} = 0$, $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, and $\text{T_UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty*) = (\sigma_2, \Delta_2, 1)$.

By definition 3.3.1, given $c = wp$, we have $c \simeq_L c'$ if and only if $c' = wp$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wp} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public Pointer Write Single Location, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, (l'_e, \mu'_e))$, $\gamma(x) = (l', \text{public } bty'*)$, $\sigma'_1(l') = (\omega', \text{public } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, $\text{acc} = 0$, $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l'_1, \mu'_1)], [1], i']$, and $\text{T_UpdatePtr}(\sigma'_1, (l', 0), [1, [(l'_e, \mu'_e)], [1], i'], \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty'*) = (\sigma'_2, \Delta'_2, 1)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_e, \mu_e))$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, (l'_e, \mu'_e))$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $l_e = l'_e$, $\mu_e = \mu'_e$, and $c_1 \simeq_L c'_1$.

Given $\gamma(x) = (l, \text{public } bty*)$ and $\gamma(x) = (l', \text{public } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\sigma'_1(l') = (\omega', \text{public } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l'_1, \mu'_1)], [1], i']$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $l_1 = l'_1$, $\mu_1 = \mu'_1$, and $i = i'$.

Given $\text{T_UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty*) = (\sigma_2, \Delta_2, 1)$, $\text{T_UpdatePtr}(\sigma'_1, (l', 0), [1, [(l'_e, \mu'_e)], [1], i'], \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty'*) = (\sigma'_2, \Delta'_2, 1)$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $l = l'$, $l_e = l'_e$, $\mu_e = \mu'_e$, $i = i'$, and $bty = bty'$, by Lemma 3.3.15 we have $\sigma_2 = \sigma'_2$ and $\Delta_2 = \Delta'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wp1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wp} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Pointer Write Multiple Locations, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}, \bar{j}, i])$, $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, and $\text{T_UpdatePtr}(\sigma_1, (l, 0), [\alpha, \bar{l}, \bar{j}, i], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*) = (\sigma_2, \Delta_2, 1)$.

By definition 3.3.1, given $c = wp2$, we have $c \simeq_L c'$ if and only if $c' = wp2$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wp2} (\gamma, \sigma_2', \Delta_2', \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Pointer Write Multiple Locations, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \Delta_1', \overline{\chi}, \text{bid}, \text{acc}, [\alpha', \bar{l}', \bar{j}', i'])$, $\gamma(x) = (l', \text{private } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, and $\text{T_UpdatePtr}(\sigma_1', (l', 0), [\alpha', \bar{l}', \bar{j}', i'], \Delta_1', \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty'*) = (\sigma_2', \Delta_2', 1)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}, \bar{j}, i])$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow_{c_1'} (\gamma, \sigma_1', \Delta_1', \overline{\chi}, \text{bid}, \text{acc}, [\alpha', \bar{l}', \bar{j}', i'])$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$, and $c_1 \simeq_L c_1'$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\text{T_UpdatePtr}(\sigma_1, (l, 0), [\alpha, \bar{l}, \bar{j}, i], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*) = (\sigma_2, \Delta_2, 1)$, $\text{T_UpdatePtr}(\sigma_1', (l, 0), [\alpha', \bar{l}', \bar{j}', i'], \Delta_1', \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty'*) = (\sigma_2', \Delta_2', 1)$, $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $l = l'$, $bty = bty'$, and $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$, by Lemma 3.3.15 we have $\sigma_2 = \sigma_2'$ and $\Delta_2 = \Delta_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $\overline{\chi} = \overline{\chi}$, bid = bid,acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^{t}_{rdp} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^{t}_{rdp} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$ by rule Location-tracking Public Pointer Dereference Single Location, we have $\gamma(x) = (l, \text{public } bty*)$, $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, and $\text{DerefPtr}(\sigma, \text{public } bty, (l_1, \mu_1)) = (v, 1)$.

By definition 3.3.1, given $c = rdp$, we have $c \simeq_L c'$ if and only if $c' = rdp$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^{t}_{rdp} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$ by rule Location-tracking Public Pointer Dereference Single Location, we have $\gamma(x) = (l', \text{public } bty'*)$, $\sigma(l') = (\omega', \text{public } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l'_1, \mu'_1)], [1], 1]$, and $\text{DerefPtr}(\sigma, \text{public } bty', (l'_1, \mu'_1)) = (v', 1)$.

Given $\gamma(x) = (l, \text{public } bty*)$ and $\gamma(x) = (l', \text{public } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\sigma(l') = (\omega', \text{public } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l'_1, \mu_1)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, \mu_1)], [1], 1] = [1, [(l'_1, \mu_1)], [1], 1]$ and therefore $(l_1, \mu_1) = (l'_1, \mu'_1)$.

Given $\text{DerefPtr}(\sigma, \text{public } bty, (l_1, \mu_1)) = (v, 1)$, $\text{DerefPtr}(\sigma, \text{public } bty', (l'_1, \mu'_1)) = (v', 1)$, $bty = bty'$, $(l_1, \mu_1) = (l'_1, \mu'_1)$, by Lemma 3.3.19 we have $v = v'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $v = v'$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^{t}_{rdp1} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^{t}_{rdp1} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$ by rule Location-tracking Public Pointer Dereference Single Location Higher Level Indirection, we have $\gamma(x) = (l, \text{public } bty*)$, $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i]$, $i > 1$, and

DerefPtrHLI(σ, public $bty*$, (l_1, μ_1)) = ([1, [(l_2, μ_2)], [1], $i-1$], 1).

By definition 3.3.1, given $c = rdp1$, we have $c \simeq_L c'$ if and only if $c' = rdp$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^t_{rdp1} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, (l'_2, \mu'_2))$ by rule Location-tracking Public Pointer Dereference Single Location Higher Level Indirection, we have $\gamma(x) = (l', \text{public } bty'*)$, $\sigma(l') = (\omega', \text{public } bty'*, 1, \text{PermL(Freeable, public } bty'*, \text{public}, 1))$, DecodePtr(public $bty'*$, $1, \omega'$) = [1, [(l'_1, μ'_1)], [1], i'], $i' > 1$, and DerefPtrHLI(σ, public $bty'*$, (l'_1, μ'_1)) = ([1, [(l'_2, μ'_2)], [1], $i'-1$], 1).

Given $\gamma(x) = (l, \text{public } bty*)$ and $\gamma(x) = (l', \text{public } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL(Freeable, public } bty*, \text{public}, 1))$, $\sigma(l') = (\omega', \text{public } bty'*, 1, \text{PermL(Freeable, public } bty'*, \text{public}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given DecodePtr(public $bty*$, $1, \omega$) = [1, [(l_1, μ_1)], [1], i], DecodePtr(public $bty'*$, $1, \omega'$) = [1, [(l'_1, μ'_1)], [1], i'], $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have [1, [(l_1, μ_1)], [1], i] = [1, [(l'_1, μ'_1)], [1], i'] and therefore $(l_1, \mu_1) = (l'_1, \mu'_1)$ and $i = i'$.

Given DerefPtrHLI(σ, public $bty*$, (l_1, μ_1)) = ([1, [(l_2, μ_2)], [1], $i-1$], 1), DerefPtrHLI(σ, public $bty'*$, (l'_1, μ'_1)) = ([1, [(l'_2, μ'_2)], [1], $i'-1$], 1), $bty = bty'$, and $(l_1, \mu_1) = (l'_1, \mu'_1)$, by Lemma 3.3.20 we have [1, [(l_2, μ_2)], [1], $i-1$] = [1, [(l'_2, μ'_2)], [1], $i'-1$] and therefore $(l_2, \mu_2) = (l'_2, \mu'_2)$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, $(l_2, \mu_2) = (l'_2, \mu'_2)$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^t_{rdp2} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^t_{rdp2} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$ by rule Location-tracking Private Pointer Dereference, we have $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL(Freeable, private } bty*, \text{private}, \alpha))$, DecodePtr(private $bty*$, α, ω) = $[\alpha, \overline{l}, \overline{j}, 1]$, and Retrieve_vals($\alpha, \overline{l}, \overline{j}$, private bty, σ) = $(v, 1)$.

By definition 3.3.1, given $c = rdp2$, we have $c \simeq_L c'$ if and only if $c' = rdp2$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^{t}_{rdp2} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$ by rule Location-tracking Private Pointer Dereference, we have $\gamma(x) = (l', \text{private } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, \alpha'))$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega) = [\alpha', \overline{l}', \overline{j}', 1]$, and $\text{Retrieve_vals}(\alpha', \overline{l}', \overline{j}', \text{private } bty', \sigma) = (v', 1)$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\sigma(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, \alpha'))$, and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, 1]$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega) = [\alpha', \overline{l}', \overline{j}', 1]$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \overline{l}, \overline{j}, 1] = [\alpha', \overline{l}, \overline{j}, 1]$. Therefore, we have $\overline{l} = \overline{l}'$ and $\overline{j} = \overline{j}'$.

Given $\text{Retrieve_vals}(\alpha, \overline{l}, \overline{j}, \text{private } bty, \sigma) = (v, 1)$, $\text{Retrieve_vals}(\alpha', \overline{l}', \overline{j}', \text{private } bty', \sigma) = (v', 1)$, $\alpha = \alpha'$, $\overline{l} = \overline{l}'$, $\overline{j} = \overline{j}'$, and $bty = bty'$, by Lemma 3.3.17 we have $v = v'$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $v = v'$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^{t*}_{rdp2} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^{t}_{rdp2} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, v)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^{t}_{rdp3} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [\alpha', \overline{l}', \overline{j}', i-1])$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^{t}_{rdp3} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [\alpha', [(l'_0, \mu'_0), ..., (l'_{\alpha'-1}, \mu'_{\alpha'-1})], \overline{j}', i-1])$ by rule Location-tracking Private Pointer Dereference Higher Level Indirection, we have $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$, $i > 1$, and $\text{DerefPrivPtr}(\alpha, \overline{l}, \overline{j}, \text{private } bty*, \sigma) = ((\alpha', \overline{l}', \overline{j}'), 1)$.

By definition 3.3.1, given $c = rdp3$, we have $c \simeq_L c'$ if and only if $c' = rdp3$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x) \Downarrow^{t}_{rdp3} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [\alpha''', [(l'''_0, \mu'''_0), ..., (l'''_{\alpha'''-1}, \mu'''_{\alpha'''-1})], [j'''_0, ...,$

$j'''_{\alpha'''-1}], i'-1])$ by rule Location-tracking Private Pointer Dereference Higher Level Indirection, we have $\gamma(x) = (l'$, private $bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma(l') = (\omega'$, private $bty'*$, $\alpha'', \text{PermL}(\text{Freeable, private } bty'*$, private, $\alpha''))$, $\text{DecodePtr}(\text{private } bty'*, \alpha'', \omega') = [\alpha'', \bar{l}'', \bar{j}'', i']$, $i' > 1$, and $\text{DerefPrivPtr}(\alpha'', \bar{l}'', \bar{j}''$, private $bty'*, \sigma) = ((\alpha''', \bar{l}''', \bar{j}'''), 1)$.

Given $\gamma(x) = (l$, private $bty*)$ and $\gamma(x) = (l'$, private $bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega$, private $bty*$, $\alpha, \text{PermL}(\text{Freeable, private } bty*$, private, $\alpha))$, $\sigma(l') = (\omega'$, private $bty'*$, α'', $\text{PermL}(\text{Freeable, private } bty'*$, private, $\alpha''))$, and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha''$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$, $\text{DecodePtr}(\text{private } bty'*, \alpha'', \omega') = [\alpha'', \bar{l}'', \bar{j}'', i']$, $bty = bty'$, $\alpha = \alpha''$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \bar{l}, \bar{j} = [\alpha'', \bar{l}'', \bar{j}'', i']$. Therefore, we have $\bar{l} = \bar{l}''$, $\bar{j} = \bar{j}''$, and $i = i'$.

Given $\text{DerefPrivPtr}(\alpha, \bar{l}, \bar{j}$, private $bty*, \sigma) = ((\alpha', \bar{l}', \bar{j}'), 1)$, $\text{DerefPrivPtr}(\alpha'', \bar{l}'', \bar{j}''$, private $bty'*, \sigma) = ((\alpha'''$, $\bar{l}''', \bar{j}'''), 1)$, $\alpha = \alpha''$, $\bar{l} = \bar{l}''$, $\bar{j} = \bar{j}''$, and $bty = bty'$, by Lemma 3.3.18 we have $(\alpha', \bar{l}', \bar{j}') = (\alpha''', \bar{l}''', \bar{j}''')$.

Given $(\alpha', \bar{l}', \bar{j}') = (\alpha''', \bar{l}''', \bar{j}''')$ and $i = i'$, we have $[\alpha', \bar{l}', \bar{j}', i-1] = ([\alpha''', \bar{l}''', \bar{j}''', i'-1]$.

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, $[\alpha', \bar{l}', \bar{j}', i-1] = ([\alpha''', \bar{l}''', \bar{j}''', i'-1]$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc}, *x) \Downarrow^t_{rdp3} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc}, [\alpha', \bar{l}', \bar{j}', i-1])$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc}, *x) \Downarrow^t_{rdp3} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc}, [\alpha', \bar{l}', \bar{j}', i-1])$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc}, *x = e) \Downarrow^t_{wdp} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid, acc, skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc}, *x = e) \Downarrow^t_{wdp} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid, acc, skip})$ by rule Location-tracking Public Pointer Dereference Write Public Value, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid, acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l$, public $bty*)$, $\sigma_1(l) = (\omega$, public $bty*$, 1, $\text{PermL}(\text{Freeable, public } bty*$, public, 1)), $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{T_UpdateOffset}(\sigma_1, (l_1, \mu_1), v, \Delta_1, \overline{\chi}, \text{bid, acc, public } bty) = (\sigma_2, \Delta_2, 1)$, $\text{Label}(e, \gamma) = \text{public}$, and $(\text{acc} = 0) \wedge (\text{bid} = none)$.

By definition 3.3.1, given $c = wdp$, we have $c \simeq_L c'$ if and only if $c' = wdp$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public Pointer Dereference Write Public Value, we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, v')$, $v' \neq \text{skip}$, $\gamma(x) = (l', \text{public } bty'*)$, $\sigma'_1(l') = (\omega', \text{public } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\text{T_UpdateOffset}(\sigma'_1, (l'_1, \mu'_1), v', \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty') = (\sigma'_2, \Delta'_2, 1)$, $\text{Label}(e, \gamma) = \text{public}$, and $(\text{acc} = 0) \wedge (\text{bid} = none)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, v')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, Δ'_1, $v = v'$, and $c_1 \simeq_L c'_1$.

Given $\gamma(x) = (l, \text{public } bty*)$ and $\gamma(x) = (l', \text{public } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, $\sigma'_1(l') = (\omega', \text{public } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$, and therefore $l_1 = l'_1$.

Given $\text{T_UpdateOffset}(\sigma_1, (l_1, \mu_1), v, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty) = (\sigma_2, 1)$, $\text{T_UpdateOffset}(\sigma'_1, (l'_1, \mu'_1), v', \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty') = (\sigma'_2, 1, \text{public } bty')$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $l_1 = l'_1$, $bty = bty$, and $v = v'$, by Lemma 3.3.13 we have $\sigma_2 = \sigma'_2$ and $\Delta_2 = \Delta'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t*}_{wdp} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
This case is similar to case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public Pointer Dereference Write Higher Level Indirection, we have $(\text{acc} = 0) \wedge (\text{bid} = none)$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_e, \mu_e))$, $\gamma(x) = (l, \text{public } bty*)$, $\sigma_1(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable},$

public $bty*$, public, 1)), DecodePtr(public $bty*$, 1, ω) = [1, [(l_1, μ_1)], [1], i], $i > 1$, Label(e, γ) = public, and T_UpdatePtr($\sigma_1, (l_1, \mu_1)$, [1, [(l_e, μ_e)], [1], $i - 1$], Δ_1, $\overline{\chi}$, bid, acc, public $bty*$) = (σ_2, Δ_2, 1).

By definition 3.3.1, given $c = wdp1$, we have $c \simeq_L c'$ if and only if $c' = wdp1$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $*x = e) \Downarrow^t_{wdp1} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}$, bid, acc, skip) by rule Location-tracking Public Pointer Dereference Write Higher Level Indirection, we have (acc = 0) $\wedge$ (bid = $none$), (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c'_1}$ (γ, σ'_1, Δ'_1, $\overline{\chi}$, bid, acc, (l'_e, μ'_e)), $\gamma(x) = (l'$, public $bty'*$), $\sigma'_1(l') = (\omega'$, public $bty'*$, 1, PermL(Freeable, public $bty'*$, public, 1)), DecodePtr(public $bty'*$, 1, ω') = [1, [(l'_1, μ'_1)], [1], i'], $i' > 1$, Label(e, γ) = public, and T_UpdatePtr($\sigma'_1, (l'_1, \mu'_1)$, [1, [$(l'_e, \mu'_e)$], [1], $i' - 1$], Δ'_1, $\overline{\chi}$, bid, acc, public $bty'*$) = (σ'_2, Δ'_2, 1).

Given (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, (l_e, μ_e)) and (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c'_1}$ (γ, σ'_1, Δ'_1, $\overline{\chi}$, bid, acc, (l'_e, μ'_e)), by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $(l_e, \mu_e) = (l'_e, \mu'_e)$, and $c_1 \simeq_L c'_1$.

Given $\gamma(x) = (l$, public $bty*$) and $\gamma(x) = (l'$, public $bty'*$) we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega$, public $bty*$, 1, PermL(Freeable, public $bty*$, public, 1)), $\sigma'_1(l') = (\omega'$, public $bty'*$, 1, PermL(Freeable, public $bty'*$, public, 1)), $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$.

Given DecodePtr(public $bty*$, 1, ω) = [1, [(l_1, μ_1)], [1], i], DecodePtr(public $bty'*$, 1, ω') = [1, [(l'_1, μ'_1)], [1], i'], $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have [1, [(l_1, μ_1)], [1], i] = [1, [(l'_1, μ'_1)], [1], i']. Therefore, we have (l_1, μ_1) = (l'_1, μ'_1) and $i = i'$.

Given T_UpdatePtr(σ_1, (l_1, μ_1), [1, [$(l_e$, $\mu_e)$], [1], i−1], Δ, $\overline{\chi}$, bid, acc, public $bty*$) = ($\sigma_2, \Delta_2, 1$), T_UpdatePtr(σ'_1, (l'_1, μ'_1), [1, [(l'_e, μ'_e)], [1], $i' - 1$], Δ'_1, $\overline{\chi}$, bid, acc, public $bty'*$) = (σ'_2, Δ'_2, 1), $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $(l_1, \mu_1) = (l'_1$, $\mu'_1)$, $(l_e, \mu_e) = (l'_e, \mu'_e)$, $bty = bty'$, and $i = i'$, by Lemma 3.3.15 we have $\sigma_2 = \sigma'_2$ and $\Delta_2 = \Delta'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma$, acc, $*x = e) \Downarrow^{t*}_{wdp1} (\gamma, \sigma_2$, acc, skip)
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $*x = e) \Downarrow^t_{wdp1} (\gamma, \sigma_2, \Delta_2, \overline{\chi}$, bid, acc, skip)

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Pointer Dereference Write Higher Level Indirection, we have $\gamma(x) = (l, \text{private } bty*)$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_e, \mu_e))$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$, and $\text{T_UpdatePrivPtr}(\sigma_1, [\alpha, \bar{l}, \bar{j}, i], [1, [(l_e, \mu_e)], [1], i-1], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*) = (\sigma_2, \Delta_2, 1)$.

By definition 3.3.1, given $c = wdp2$, we have $c \simeq_L c'$ if and only if $c' = wdp2$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Pointer Dereference Write Higher Level Indirection, we have $\gamma(x) = (l', \text{private } bty'*)$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, (l'_e, \mu'_e))$, $\sigma'_1(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, \alpha'))$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', i']$, and $\text{T_UpdatePrivPtr}(\sigma'_1, [\alpha', \bar{l}', \bar{j}', i'], [1, [(l'_e, \mu'_e)], [1], i'-1], \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty'*) = (\sigma'_2, \Delta'_2, 1)$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_e, \mu_e))$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, (l'_e, \mu'_e))$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $(l_e, \mu_e) = (l'_e, \mu'_e)$, and $c_1 \simeq_L c'_1$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\sigma'_1(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, \alpha'))$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i]$, and $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', i']$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$. Therefore we have $i = i'$.

Given $\text{T_UpdatePrivPtr}(\sigma_1, [\alpha, \bar{l}, \bar{j}, i], [1, [(l_e, \mu_e)], [1], i-1], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*) = (\sigma_2, \Delta_2, 1)$, $\text{T_UpdatePrivPtr}(\sigma'_1, [\alpha', \bar{l}', \bar{j}', i'], [1, [(l'_e, \mu'_e)], [1], i'-1], \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty'*) = (\sigma'_2, \Delta'_2, 1)$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $[\alpha, \bar{l}, \bar{j}, i] = [\alpha', \bar{l}', \bar{j}', i']$, $(l_e, \mu_e) = (l'_e, \mu'_e)$, $bty = bty'$, and $i = i'$, by Lemma 3.3.16 we have $\sigma_2 = \sigma'_2$ and $\Delta_2 = \Delta'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1,

we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t*}_{wdp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Pointer Dereference Write Private Value, we have $\gamma(x) = (l, \text{private } bty*)$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$, $v \neq \text{skip}$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{Label}(e, \gamma) = \text{private}$, $(bty = \text{int}) \vee (bty = \text{float})$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, and $\text{T_UpdatePriv}(\sigma_1, \alpha, \bar{l}, \bar{j}, \text{private } bty, v, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_2, \Delta_2, 1)$.

By definition 3.3.1, given $c = wdp3$, we have $c \simeq_L c'$ if and only if $c' = wdp3$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp3} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Pointer Dereference Write Private Value, we have $\gamma(x) = (l', \text{private } bty'*)$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, v')$, $v' \neq \text{skip}$, $\sigma'_1(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, \alpha'))$, $\text{Label}(e, \gamma) = \text{private}$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', 1]$, and $\text{T_UpdatePriv}(\sigma'_1, \alpha', \bar{l}', \bar{j}', \text{private } bty', v', \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma'_2, \Delta'_2, 1)$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, v')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $v = v'$, and $c_1 \simeq_L c'_1$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\sigma'_1(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, \alpha'))$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, 1]$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \bar{l}', \bar{j}', 1]$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \bar{l}, \bar{j}, 1] = [\alpha', \bar{l}', \bar{j}', 1]$. Therefore, we have $\bar{l} = \bar{l}'$ and $\bar{j} = \bar{j}'$.

Given $\text{T_UpdatePriv}(\sigma_1, \alpha, \bar{l}, \bar{j}, \text{private } bty, v, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_2, \Delta_2, 1)$, $\text{T_UpdatePriv}(\sigma'_1, \alpha', \bar{l}',$

$\overline{j}'$, private bty', v', Δ_1', $\overline{\chi}$, bid, acc) $= (\sigma_2', \Delta_2', 1)$, $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $\alpha = \alpha'$, $\overline{l} = \overline{l}'$, $\overline{j} = \overline{j}'$, $bty = bty'$ and $v = v'$, by Lemma 3.3.14 we have $\sigma_2 = \sigma_2'$ and $\Delta_2 = \Delta_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t*}_{wdp3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp3} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp4} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp4} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Pointer Dereference Write Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, 1]$, and $\text{T_UpdatePriv}(\sigma_1, \alpha, \overline{l}, \overline{j}, \text{private } bty, \text{encrypt}(v), \Delta_1, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_2, \Delta_2, 1)$.

By definition 3.3.1, given $c = wdp4$, we have $c \simeq_L c'$ if and only if $c' = wdp4$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp4} (\gamma, \sigma_2', \Delta_2', \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Pointer Dereference Write Public Value, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1'} (\gamma, \sigma_1', \Delta_1', \overline{\chi}, \text{bid}, \text{acc}, v')$, $v' \neq \text{skip}$, $\gamma(x) = (l', \text{private } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma_1(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, \alpha'))$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \overline{l}', \overline{j}', 1]$, and $\text{T_UpdatePriv}(\sigma_1', \alpha', \overline{l}', \overline{j}', \text{private } bty', \text{encrypt}(v'), \Delta_1', \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_2', \Delta_2', 1)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1'} (\gamma, \sigma_1', \Delta_1', \overline{\chi}, \text{bid}, \text{acc}, v')$, by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $v = v'$, and $c_1 \simeq_L c_1'$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))$, $\sigma_1(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL}(\text{Freeable}, \text{private } bty'*, \text{private}, \alpha'))$, $\sigma_1 = \sigma_1'$, and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given DecodePtr(private $bty*$, α, ω) $= [\alpha, \bar{l}, \bar{j}, 1]$, DecodePtr(private $bty'*$, α', ω') $= [\alpha', \bar{l}', \bar{j}', 1]$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \bar{l}, \bar{j}, 1] = [\alpha', \bar{l}', \bar{j}', 1]$. Therefore, we have $\bar{l} = \bar{l}'$ and $\bar{j} = \bar{j}'$.

Given T_UpdatePriv(σ_1, α, $\bar{l}$, $\bar{j}$, private bty, encrypt(v), Δ_1, $\overline{\chi}$, bid, acc) $= (\sigma_2, \Delta_2, 1)$, T_UpdatePriv($\sigma_1'$, α', $\bar{l}'$, $\bar{j}'$, private bty', encrypt(v'), Δ_1', $\overline{\chi}$, bid, acc) $= (\sigma_2', \Delta_2', 1)$, $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $\alpha = \alpha'$, $\bar{l} = \bar{l}'$, $\bar{j} = \bar{j}'$, $bty = bty'$, and $v = v'$, by Axiom 3.3.1 we have encrypt(v) $=$ encrypt(v'), and therefore by Lemma 3.3.14 we have $\sigma_2 = \sigma_2'$ and $\Delta_2 = \Delta_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $\overline{\chi} = \overline{\chi}$, bid $=$ bid, acc $=$ acc, skip $=$ skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t*}_{wdp4} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp4} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp5} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp5} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Pointer Dereference Write Higher Level Indirection Multiple Locations, we have $\gamma(x) = (l, \text{private } bty*)$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha_e, \bar{l}_e, \bar{j}_e, i-1])$, $\sigma_1(l) = (\omega, \text{private } bty*, \alpha, \text{PermL(Freeable, private } bty*, \text{private}, \alpha))$, Label$(e, \gamma) =$ private, acc $= 0$, DecodePtr(private $bty*$, α, ω) $= [\alpha, \bar{l}, \bar{j}, i]$, $i > 1$, and T_UpdatePrivPtr(σ_1, $[\alpha, \bar{l}, \bar{j}, i]$, $[\alpha_e, \bar{l}_e, \bar{j}_e, i-1]$, Δ_1, $\overline{\chi}$, bid, acc, private $bty*$) $= (\sigma_2, \Delta_2, 1)$.

By definition 3.3.1, given $c = wdp5$, we have $c \simeq_L c'$ if and only if $c' = wdp5$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, *x = e) \Downarrow^{t}_{wdp5} (\gamma, \sigma_2', \Delta_2', \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private Pointer Dereference Write Higher Level Indirection Multiple Locations, we have $\gamma(x) = (l', \text{private } bty'*)$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1'} (\gamma, \sigma_1', \Delta_1', \overline{\chi}, \text{bid}, \text{acc}, [\alpha_e', \bar{l}_e', \bar{j}_e', i'-1])$, $\sigma_1'(l') = (\omega', \text{private } bty'*, \alpha', \text{PermL(Freeable, private } bty'*, \text{private}, \alpha'))$, Label$(e, \gamma) =$ private, acc $= 0$, DecodePtr(private $bty'*$, α', ω') $= [\alpha', \bar{l}', \bar{j}', i']$, $i' > 1$, and T_UpdatePrivPtr(σ_1', $[\alpha', \bar{l}', \bar{j}', i']$, $[\alpha_e', \bar{l}_e', \bar{j}_e', i'-1]$, Δ_1', $\overline{\chi}$, bid, acc, private $bty'*$) $= (\sigma_2', \Delta_2', 1)$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha_e, \bar{l}_e, \bar{j}_e, i-1])$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1'} (\gamma,$

σ'_1, Δ'_1, $\overline{\chi}$, bid, acc, $[\alpha'_e, \overline{l}'_e, \overline{j}'_e, i'-1]$), by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1] = [\alpha'_e, \overline{l}'_e, \overline{j}'_e, i'-1]$, and $c_1 \simeq_L c'_1$.

Given $\sigma_1(l) = (\omega$, private $bty*$, α, PermL(Freeable, private $bty*$, private, $\alpha))$, $\sigma'_1(l') = (\omega'$, private $bty'*$, α', PermL(Freeable, private $bty'*$, private, $\alpha'))$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given DecodePtr(private $bty*$, $\alpha, \omega) = [\alpha, \overline{l}, \overline{j}, i]$, DecodePtr(private $bty'*$, $\alpha', \omega') = [\alpha', \overline{l}', \overline{j}', i']$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \overline{l}, \overline{j}, i] = [\alpha', \overline{l}', \overline{j}', i']$.

Given T_UpdatePrivPtr($\sigma_1, [\alpha, \overline{l}, \overline{j}, i], [\alpha_e, \overline{l}_e, \overline{j}_e, i-1], \Delta_1, \overline{\chi}$, bid, acc, private $bty*) = (\sigma_2, \Delta_2, 1)$, T_UpdatePrivPtr($\sigma'_1$, $[\alpha', \overline{l}', \overline{j}', i']$, $[\alpha'_e, \overline{l}'_e, \overline{j}'_e, i'-1]$, Δ'_1, $\overline{\chi}$, bid, acc, private $bty'*) = (\sigma'_2, \Delta'_2, 1)$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $[\alpha, \overline{l}, \overline{j}, i] = [\alpha', \overline{l}', \overline{j}', i']$, $bty = bty'$, and $[\alpha_e, \overline{l}_e, \overline{j}_e, i-1] = [\alpha'_e, \overline{l}'_e, \overline{j}'_e, i'-1]$, by Lemma 3.3.16 we have $\sigma_2 = \sigma'_2$ and $\Delta_2 = \Delta'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $*x = e) \Downarrow^{t*}_{wdp5} (\gamma, \sigma_2, \Delta_2, \overline{\chi}$, bid, acc, skip)
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $*x = e) \Downarrow^{t}_{wdp5} (\gamma, \sigma_2, \Delta_2, \overline{\chi}$, bid, acc, skip).

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $++\ x) \Downarrow^{t}_{pin} (\gamma, \sigma_1, \Delta, \overline{\chi}$, bid, acc, $v_1)$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $++\ x) \Downarrow^{t}_{pin} (\gamma, \sigma_1, \Delta, \overline{\chi}$, bid, acc, $v_1)$ by rule Location-tracking Pre-Increment Public Variable, we have $\gamma(x) = (l$, public $bty)$, $\sigma(l) = (\omega$, public bty, 1, PermL(Freeable, public bty, public, 1)), DecodeVal(public bty, $1, \omega) = v$, $v_1 =_{\text{public}} v +_{\text{public}} 1$, and T_UpdateVal($\sigma, l$, v_1, Δ, $\overline{\chi}$, bid, acc, public bty) $= (\sigma_1, \Delta)$.

By definition 3.3.1, given $c = pin$, we have $c \simeq_L c'$ if and only if $c' = pin$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $++\ x) \Downarrow^{t}_{pin} (\gamma, \sigma'_1, \Delta, \overline{\chi}$, bid, acc, $v'_1)$ by rule Location-tracking Pre-Increment Public Variable, we have $\gamma(x) = (l'$, public $bty')$, $\sigma(l') = (\omega'$, public bty', 1, PermL(Freeable, public bty', public, 1)), DecodeVal(public bty', $1, \omega') = v'$, $v'_1 =_{\text{public}} v' +_{\text{public}} 1$, and T_UpdateVal($\sigma$, l', v'_1, Δ, $\overline{\chi}$, bid, acc, public $bty') = (\sigma'_1, \Delta)$.

Given $\gamma(x) = (l, \text{public } bty)$ and $\gamma(x) = (l', \text{public } bty')$ we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{ public } bty, 1, \text{PermL(Freeable, public } bty\text{, public, 1))}$, $\sigma(l') = (\omega', \text{ public } bty', 1, \text{PermL(Freeable, public } bty'\text{, public, 1))}$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodeVal(public } bty, 1, \omega) = v$, $\text{DecodeVal(public } bty', 1, \omega') = v'$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.4 we have $v = v'$.

Given $v_1 =_{\text{public}} v +_{\text{public}} 1$, $v_1' =_{\text{public}} v' +_{\text{public}} 1$, and $v = v'$, we have $v_1 = v_1'$.

Given $\text{T_UpdateVal}(\sigma, l, v_1, \Delta, \overline{\chi}, \text{bid, acc, public } bty) = (\sigma_1, \Delta)$, $\text{T_UpdateVal}(\sigma, l', v_1', \Delta, \overline{\chi}, \text{bid, acc, public } bty') = (\sigma_1', \Delta)$, $l = l'$, $bty = bty$, and $v_1 = v_1'$, by Lemma 3.3.12 we have $\sigma_1 = \sigma_1'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma_1'$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $v_1 = v_1'$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc}, ++\ x) \Downarrow^t_{pin1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid, acc}, v_1)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc}, ++\ x) \Downarrow^t_{pin1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid, acc}, v_1)$ by rule Location-tracking Pre-Increment Private Variable, we have $\gamma(x) = (l, \text{private } bty)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega, \text{private } bty, 1, \text{PermL(Freeable, private } bty\text{, private, 1))}$, $\text{DecodeVal(private } bty, 1, \omega) = v$, $v_1 =_{\text{private}} v +_{\text{private}} \text{encrypt}(1)$, and $\text{T_UpdateVal}(\sigma, l, v_1, \Delta, \overline{\chi}, \text{bid, acc, private } bty) = (\sigma_1, \Delta_1)$.

By definition 3.3.1, given $c = pin1$, we have $c \simeq_L c'$ if and only if $c' = pin1$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid, acc}, ++\ x) \Downarrow^t_{pin1} (\gamma, \sigma_1', \Delta_1', \overline{\chi}, \text{bid, acc}, v_1')$ by rule Location-tracking Pre-Increment Private Variable, we have $\gamma(x) = (l', \text{private } bty')$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma(l') = (\omega', \text{private } bty', 1, \text{PermL(Freeable, private } bty'\text{, private, 1))}$, $\text{DecodeVal(private } bty', 1, \omega') = v'$, $v_1' =_{\text{private}} v' +_{\text{private}} \text{encrypt}(1)$, and $\text{T_UpdateVal}(\sigma, l', v_1', \Delta, \overline{\chi}, \text{bid, acc, private } bty') = (\sigma_1', \Delta_1')$.

Given $\gamma(x) = (l, \text{private } bty)$ and $\gamma(x) = (l', \text{private } bty')$ we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{ private } bty, 1, \text{PermL(Freeable, private } bty\text{, private, 1))}$, $\sigma(l') = (\omega', \text{ private } bty', 1,$

PermL(Freeable, private bty', private, 1)), and $l = l'$, we have $\omega = \omega'$.

Given DecodeVal(private bty, $1, \omega) = v$, DecodeVal(private bty', $1, \omega) = v'$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.4 we have $v = v'$.

Given $v_1 =_{\text{private}} v +_{\text{private}} \text{encrypt}(1)$, $v_1' =_{\text{private}} v' +_{\text{private}} \text{encrypt}(1)$, and $v = v'$, by Axiom 3.3.2 we have $v_1 = v_1'$.

Given T_UpdateVal(σ, l, v_1, Δ, $\overline{\chi}$, bid, acc, private bty) $= (\sigma_1, \Delta_1)$, T_UpdateVal(σ, l, v_1, Δ, $\overline{\chi}$, bid, acc, private bty') $= (\sigma_1', \Delta_1')$, $l = l'$, $bty = bty'$, and $v_1 = v_1'$, by Lemma 3.3.12 we have $\sigma_1 = \sigma_1'$ and $\Delta_1 = \Delta_1'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, $v_1 = v_1'$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$ by rule Location-tracking Pre-Increment Public Pointer Single Location, we have $\gamma(x) = (l, \text{public } bty*)$, $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL(Freeable, public } bty*, \text{public}, 1))$, DecodePtr(public $bty*$, $1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, GetLocation($(l_1, \mu_1)$, τ(public bty), $\sigma) = ((l_2, \mu_2), 1)$, and T_UpdatePtr($\sigma$, $(l, 0)$, $[1, [(l_2, \mu_2)], [1], 1]$, Δ, $\overline{\chi}$, bid, acc, public $bty*$) $= (\sigma_1, \Delta_1, 1)$.

By definition 3.3.1, given $c = pin2$, we have $c \simeq_L c'$ if and only if $c' = pin2$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^t_{pin2} (\gamma, \sigma_1', \Delta_1', \overline{\chi}, \text{bid}, \text{acc}, (l_2', \mu_2'))$ by rule Location-tracking Pre-Increment Public Pointer Single Location, we have $\gamma(x) = (l', \text{public } bty'*)$, $\sigma(l') = (\omega', \text{public } bty'*, 1, \text{PermL(Freeable, public } bty'*, \text{public}, 1))$, DecodePtr(public $bty'*$, $1, \omega') = [1, [(l_1', \mu_1')], [1], 1]$, GetLocation($(l_1', \mu_1')$, τ(public bty'), $\sigma) = ((l_2', \mu_2'), 1)$, and T_UpdatePtr($\sigma$, $(l', 0)$, $[1, [(l_2', \mu_2')], [1], 1]$, Δ, $\overline{\chi}$, bid, acc, public $bty'*$) $= (\sigma_1', \Delta_1', 1)$.

Given $\gamma(x) = (l, \text{public } bty*)$ and $\gamma(x) = (l', \text{public } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL(Freeable, public } bty*, \text{public}, 1))$, $\sigma(l') = (\omega', \text{public } bty'*, 1, \text{PermL(Freeable, public } bty'*, \text{public}, 1))$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], 1]$, $\text{DecodePtr}(\text{public } bty'*, 1, \omega') = [1, [(l'_1, \mu'_1)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, \mu_1)], [1], 1] = [1, [(l'_1, \mu'_1)], [1], 1]$. Therefore, we have $(l_1, \mu_1) = (l'_1, \mu'_1)$.

Given $((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty), \sigma)$, $((l'_2, \mu'_2), 1) = \text{GetLocation}((l'_1, \mu'_1), \tau(\text{public } bty'), \sigma)$, $(l_1, \mu_1) = (l'_1, \mu'_1)$, and $bty = bty'$, by Lemma 3.3.21 we have $(l_2, \mu_2) = (l'_2, \mu'_2)$.

Given $\text{T_UpdatePtr}(\sigma, (l, 0), [1, [(l_2, \mu_2)], [1], 1], \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty*) = (\sigma_1, \Delta_1, 1)$, $\text{T_UpdatePtr}(\sigma, (l', 0), [1, [(l'_2, \mu'_2)], [1], 1], \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{public } bty'*) = (\sigma'_1, \Delta'_1, 1)$, $l = l'$, $bty = bty$, and $(l_2, \mu_2) = (l'_2, \mu'_2)$, by Lemma 3.3.15 we have $\sigma_1 = \sigma'_1$ and $\Delta_1 = \Delta'_1$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $(l_2, \mu_2) = (l'_2, \mu'_2)$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\, x) \Downarrow^{t*}_{pin2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\, x) \Downarrow^{t}_{pin2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\, x) \Downarrow^{t}_{pin6} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\, x) \Downarrow^{t}_{pin2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\, x) \Downarrow^{t*}_{pin6} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\, x) \Downarrow^{t}_{pin2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\, x) \Downarrow^{t}_{pin3} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\, x) \Downarrow^{t}_{pin2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\, x) \Downarrow^{t*}_{pin3} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t}_{pin2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t}_{pin7} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t}_{pin2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t*}_{pin7} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t}_{pin2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, (l_2, \mu_2))$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t}_{pin4} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \overline{l}', \overline{j}, 1])$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t}_{pin4} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \overline{l}', \overline{j}, 1])$ by rule Location-tracking Pre-Increment Private Pointer Multiple Locations, we have $\gamma(x) = (l, \text{private } bty*)$, $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable, private } bty*, \text{private}, \alpha))$, $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, 1]$, $\text{IncrementList}(\overline{l}, \tau(\text{private } bty), \sigma) = (\overline{l}', 1)$, and $\text{T_UpdatePtr}(\sigma, (l, 0), [\alpha, \overline{l}', \overline{j}, 1], \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*) = (\sigma_1, \Delta_1, 1)$.

By definition 3.3.1, given $c = pin4$, we have $c \simeq_L c'$ if and only if $c' = pin4$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t}_{pin4} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha', \overline{l}''', \overline{j}', 1])$ by rule Location-tracking Pre-Increment Private Pointer Multiple Locations, we have $\gamma(x) = (l', \text{private } bty'*)$, $\sigma(l') = (\omega', \text{private } bty'*, \alpha, \text{PermL}(\text{Freeable, private } bty*, \text{private}, \alpha'))$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \overline{l}'', \overline{j}', 1]$, $\text{IncrementList}(\overline{l}'', \tau(\text{private } bty'), \sigma) = (\overline{l}''', 1)$, and $\text{T_UpdatePtr}(\sigma, (l', 0), [\alpha', \overline{l}''', \overline{j}', 1], \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty'*) = (\sigma'_1, \Delta'_1, 1)$.

Given $\gamma(x) = (l, \text{private } bty*)$ and $\gamma(x) = (l', \text{private } bty'*)$ we have $l = l'$ and $bty = bty'$.

Given $\sigma(l) = (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable, private } bty*, \text{private}, \alpha))$, $\sigma(l') = (\omega', \text{private } bty'*, \alpha, \text{PermL}(\text{Freeable, private } bty*, \text{private}, \alpha'))$, $l = l'$, we have $\omega = \omega'$ and $\alpha = \alpha'$.

Given $\text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \overline{l}, \overline{j}, 1]$, $\text{DecodePtr}(\text{private } bty'*, \alpha', \omega') = [\alpha', \overline{l}'', \overline{j}', 1]$, $bty = bty'$, $\alpha = \alpha'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[\alpha, \overline{l}, \overline{j}, 1] = [\alpha', \overline{l}'', \overline{j}', 1]$. Therefore, we have $\overline{l} = \overline{l}''$ and $\overline{j} = \overline{j}'$.

Given $\text{IncrementList}(\bar{l}, \tau(\text{private } bty), \sigma) = (\bar{l}', 1)$, $\text{IncrementList}(\bar{l}'', \tau(\text{private } bty'), \sigma) = (\bar{l}''', 1)$, $\bar{l} = \bar{l}''$, and $bty = bty'$, by Lemma 3.3.22 we have $\tau(\text{private } bty) = \tau(\text{private } bty')$, and by Lemma 3.3.23 we have $\bar{l}' = \bar{l}'''$.

Given $\text{T_UpdatePtr}(\sigma, (l, 0), [\alpha, \bar{l}', \bar{j}, 1], \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty*) = (\sigma_1, \Delta_1, 1)$, $\text{T_UpdatePtr}(\sigma, (l', 0), [\alpha', \bar{l}''', \bar{j}', 1], \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty'*) = (\sigma_1', \Delta_1', 1)$, $l = l'$, $bty = bty'$, and $[\alpha, \bar{l}, \bar{j}, 1] = [\alpha', \bar{l}'', \bar{j}', 1]$, by Lemma 3.3.15 we have $\sigma_1 = \sigma_1'$ and $\Delta_1 = \Delta_1'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $[\alpha, \bar{l}', \bar{j}, 1] = [\alpha', \bar{l}''', \bar{j}', 1]$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t*}_{pin4} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1])$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t}_{pin4} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1])$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t}_{pin5} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}', \bar{j}, i])$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t}_{pin4} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1])$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t*}_{pin5} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}', \bar{j}, i])$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ++\ x) \Downarrow^{t}_{pin4} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [\alpha, \bar{l}', \bar{j}, 1])$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow^{t}_{da} (\gamma_1, \sigma_3, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow^{t}_{da} (\gamma_1, \sigma_3, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public 1 Dimension Array Declaration, we have $((ty = \text{public } bty) \wedge ((bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{int}))) \vee (ty = \text{char})$, $l = \phi()$, $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, n)$, $\gamma_1 = \gamma[x \rightarrow (l, \text{public const } bty*)]$, $l_1 = \phi()$, $\omega = \text{EncodePtr}(\text{public const } bty*, [1, [(l_1, 0)], [1], 1])$, $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))]$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, $n > 0$, $\omega_1 = \text{EncodeVal}(\text{private } bty, \text{NULL})$, and $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))]$.

By definition 3.3.1, given $c = da$, we have $c \simeq_L c'$ if and only if $c' = da$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow^{t}_{da} (\gamma'_1, \sigma'_3, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public 1 Dimension Array Declaration, we have $((ty = \text{public}\ bty') \wedge ((bty' = \text{float}) \vee (bty' = \text{char}) \vee (bty' = \text{int}))) \vee (ty = \text{char})$, $l' = \phi()$, $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, n')$, $\gamma'_1 = \gamma[x \rightarrow (l', \text{public const}\ bty'*)]$, $l'_1 = \phi()$, $\omega' = \text{EncodePtr}(\text{public const}\ bty'*, [1, [(l'_1, 0)], [1], 1])$, $\sigma'_2 = \sigma'_1[l \rightarrow (\omega', \text{public const}\ bty'*, 1, \text{PermL}(\text{Freeable}, \text{public const}\ bty'*, \text{public}, 1))]$, $(\text{acc} = 0) \wedge (\text{bid} = none)$, $n' > 0$, $\omega'_1 = \text{EncodeVal}(\text{private}\ bty', \text{NULL})$, and $\sigma'_3 = \sigma'_2[l'_1 \rightarrow (\omega'_1, \text{public}\ bty', n', \text{PermL}(\text{Freeable}, \text{public}\ bty', \text{public}, n'))]$.

Given $(ty = \text{public}\ bty)$ and $(ty = \text{public}\ bty')$, we have $bty = bty'$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, n)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta, \overline{\chi}, \text{bid}, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{public const}\ bty*)]$, $\gamma'_1 = \gamma[x \rightarrow (l', \text{public const}\ bty'*)]$, $l = l'$, $bty = bty'$, we have $\gamma_1 = \gamma'_1$.

Given $l_1 = \phi()$ and $l'_1 = \phi()$, by Axiom 3.3.5 we have $l_1 = l'_1$.

Given $\omega = \text{EncodePtr}(\text{public const}\ bty*, [1, [(l_1, 0)], [1], 1])$, $\omega' = \text{EncodePtr}(\text{public const}\ bty'*, [1, [(l'_1, 0)], [1], 1])$, $bty = bty'$, and $l_1 = l'_1$, by Lemma 3.3.5 we have $\omega = \omega'$.

Given $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{public const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public const}\ bty*, \text{public}, 1))]$, $\sigma'_2 = \sigma'_1[l \rightarrow (\omega', \text{public const}\ bty'*, 1, \text{PermL}(\text{Freeable}, \text{public const}\ bty'*, \text{public}, 1))]$, $\sigma_1 = \sigma'_1$, $l = l'$, $\omega = \omega'$, and $bty = bty'$, we have $\sigma_2 = \sigma'_2$.

Given $\omega_1 = \text{EncodeVal}(\text{public}\ bty, \text{NULL})$, $\omega'_1 = \text{EncodeVal}(\text{public}\ bty', \text{NULL})$, and $bty = bty'$, by Lemma 3.3.3 we have $\omega_1 = \omega'_1$.

Given $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{public}\ bty, n, \text{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, n))]$, $\sigma'_3 = \sigma'_2[l'_1 \rightarrow (\omega'_1, \text{public}\ bty', n', \text{PermL}(\text{Freeable}, \text{public}\ bty', \text{public}, n'))]$, $\sigma_2 = \sigma'_2$, $l_1 = l'_1$, $\omega_1 = \omega'_1$, $n = n'$, and $bty = bty'$, we have $\sigma_3 = \sigma'_3$.

Therefore, we have $\gamma_1 = \gamma_1'$, $\sigma_3 = \sigma_3'$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow^t_{da1} (\gamma_1, \sigma_3, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow^t_{da} (\gamma_1, \sigma_3, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow^t_{da2} (\gamma_1, \sigma_3, \Delta_1, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow^t_{da2} (\gamma_1, \sigma_3, \Delta, \overline{\chi}_1, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private 1 Dimension Array Declaration (Inside a Private - Conditioned If Else Branch), we have $\text{Label}(e, \gamma) = \text{public}$, $((ty = \text{private } bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_e (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n)$, $n > 0$, $l = \phi()$, $l_1 = \phi()$, $\gamma_1 = \gamma[x \rightarrow (l, \text{private const } bty*)]$, $\omega = \text{EncodePtr}(\text{private const } bty*, [1, [l_1], [1], 1])$, $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))]$, $\sigma_3 = \sigma_2[l_1 \rightarrow (\text{NULL}, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))]$, $(\text{acc} > 0) \wedge ((\text{bid} = then) \vee (\text{bid} = else))$, and $\overline{\chi}_1 = l :: l_1 :: \overline{\chi}[\text{acc}]$.

By definition 3.3.1, given $c = da2$, we have $c \simeq_L c'$ if and only if $c' = da2$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, ty\ x[e]) \Downarrow^t_{da2} (\gamma_1', \sigma_3', \Delta_1, \overline{\chi}_1', \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private 1 Dimension Array Declaration (Inside a Private - Conditioned If Else Branch), we have $\text{Label}(e, \gamma) = \text{public}$, $((ty = \text{private } bty') \vee (ty = bty')) \wedge ((bty' = \text{int}) \vee (bty' = \text{float}))$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_e (\gamma, \sigma_1', \Delta_1', \overline{\chi}, \text{bid}, \text{acc}, n')$, $n' > 0$, $l' = \phi()$, $l_1' = \phi()$, $\gamma_1' = \gamma[x \rightarrow (l', \text{private const } bty'*)]$, $\omega' = \text{EncodePtr}(\text{private const } bty'*, [1, [l_1'], [1], 1])$, $\sigma_2' = \sigma_1'[l' \rightarrow (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty'*, \text{private}, 1))]$, $\sigma_3' = \sigma_2'[l_1' \rightarrow (\text{NULL}, \text{private } bty', n', \text{PermL}(\text{Freeable}, \text{private } bty', \text{private}, n'))]$, $(\text{acc} > 0) \wedge ((\text{bid} = then) \vee (\text{bid} = else))$, and $\overline{\chi}_1' = l' :: l_1' :: \overline{\chi}[\text{acc}]$.

Given $((ty = \text{private } bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float}))$, and $((ty = \text{private } bty') \vee (ty = bty')) \wedge ((bty' = \text{int}) \vee (bty' = \text{float}))$, we have $bty = bty'$.

Given $l = \phi()$ and $l' = \phi()$, by Axiom 3.3.5 we have $l = l'$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, n)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, n')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $n = n'$, and $c_1 \simeq_L c'_1$.

Given $\gamma_1 = \gamma[x \rightarrow (l, \text{private const } bty*)]$, $\gamma'_1 = \gamma[x \rightarrow (l', \text{private const } bty'*)]$, $l = l'$, $bty = bty'$, we have $\gamma_1 = \gamma'_1$.

Given $l_1 = \phi()$ and $l'_1 = \phi()$, by Axiom 3.3.5 we have $l_1 = l'_1$.

Given $\omega = \text{EncodePtr}(\text{private const } bty*, [1, [(l_1, 0)], [1], 1])$, $\omega' = \text{EncodePtr}(\text{private const } bty'*, [1, [(l'_1, 0)], [1], 1])$, $bty = bty'$, and $l_1 = l'_1$, by Lemma 3.3.5 we have $\omega = \omega'$.

Given $\sigma_2 = \sigma_1[l \rightarrow (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))]$, $\sigma'_2 = \sigma'_1[l \rightarrow (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty'*, \text{private}, 1))]$, $\sigma_1 = \sigma'_1$, $l = l'$, $\omega = \omega'$, and $bty = bty'$, we have $\sigma_2 = \sigma'_2$.

Given $\omega_1 = \text{EncodeVal}(\text{private } bty, \text{NULL})$, $\omega'_1 = \text{EncodeVal}(\text{private } bty', \text{NULL})$, and $bty = bty'$, by Lemma 3.3.3 we have $\omega_1 = \omega'_1$.

Given $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))]$, $\sigma'_3 = \sigma'_2[l'_1 \rightarrow (\omega'_1, \text{private } bty', n', \text{PermL}(\text{Freeable}, \text{private } bty', \text{private}, n'))]$, $\sigma_2 = \sigma'_2$, $l_1 = l'_1$, $\omega_1 = \omega'_1$, $n = n'$, and $bty = bty'$, we have $\sigma_3 = \sigma'_3$.

Given $\overline{\chi}_1 = l :: l_1 :: \overline{\chi}[\text{acc}]$, $\overline{\chi}'_1 = l' :: l'_1 :: \overline{\chi}[\text{acc}]$, $l = l'$, and $l_1 = l'_1$, we have $\overline{\chi}_1 = \overline{\chi}'_1$.

Therefore, we have $\gamma_1 = \gamma'_1$, $\sigma_3 = \sigma'_3$, $\Delta_1 = \Delta'_1$, $\overline{\chi}_1 = \overline{\chi}'_1$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^t_{ra} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_i)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^t_{ra} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_i)$ by rule Location-tracking Public 1D Array Read Public Index, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public}$

bty, public, n)), DecodeVal(public bty, n, ω_1) = $[v_0, ..., v_{n-1}]$, and $0 \leq i \leq n-1$.

By definition 3.3.1, given $c = ra$, we have $c \simeq_L c'$ if and only if $c' = ra$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $x[e]) \Downarrow^t_{ra} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}$, bid, acc, $v'_i)$ by rule Location-tracking Public 1D Array Read Public Index, we have Label(e, γ) = public, $(\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}$, bid, acc, $i')$, $\gamma(x) = (l'$, public const $bty'*)$, $\sigma'_1(l') = (\omega'$, public const $bty'*$, 1, PermL(Freeable, public const $bty'*$, public, 1)), DecodePtr(public const $bty'*$, 1, ω') = [1, $[(l'_1, 0)]$, [1], 1], $\sigma'_1(l'_1) = (\omega'_1$, public bty', n', PermL(Freeable, public bty', public, n')), DecodeVal(public bty', n', ω'_1) = $[v'_0, ..., v'_{n'-1}]$, and $0 \leq i' \leq n'-1$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}$, bid, acc, $i)$ and $(\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}$, bid, acc, $i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $\gamma(x) = (l$, public const $bty*)$ and $\gamma(x) = (l'$, public const $bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega$, public const $bty*$, 1, PermL(Freeable, public const $bty*$, public, 1)), $\sigma'_1(l') = (\omega'$, public const $bty'*$, 1, PermL(Freeable, public const $bty'*$, public, 1)), $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$.

Given DecodePtr(public const $bty*$, $1, \omega$) = [1, $[(l_1, 0)]$, [1], 1], DecodePtr(public const $bty'*$, $1, \omega'$) = [1, $[(l'_1, 0)]$, [1], 1], $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have [1, $[(l_1, 0)]$, [1], 1] = [1, $[(l'_1, 0)]$, [1], 1]. Therefore, we have $l_1 = l'_1$.

Given $\sigma_1(l_1) = (\omega_1$, public bty, n, PermL(Freeable, public bty, public, n)), $\sigma'_1(l'_1) = (\omega'_1$, public bty', n', PermL(Freeable, public bty', public, n')), $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given DecodeVal(public bty, n, ω_1) = $[v_0, ..., v_{n-1}]$, DecodeVal(public bty', n', ω'_1) = $[v'_0, ..., v'_{n'-1}]$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v'_0, ..., v'_{n'-1}]$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v'_m$.

Given $0 \leq i \leq n-1$, $0 \leq i' \leq n'-1$, $i = i'$, $n = n'$, and $\forall m \in \{0, ..., n-1\}$, $v_m = v'_m$, we have $v_i = v'_i$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, $v_i = v'_i$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^{t}_{ra3} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_i)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^{t}_{ra} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v_i)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^{t}_{ra1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^{t}_{ra1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$ by rule Location-tracking Private 1D Array Read Private Index we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$, $\gamma(x) = (l, \text{private const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, and $v = \bigvee_{m=0}^{n-1}(i = \text{encrypt}(m)) \wedge v_m$.

By definition 3.3.1, given $c = ra1$, we have $c \simeq_L c'$ if and only if $c' = ra1$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^{t}_{ra1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, v')$ by rule Location-tracking Private 1D Array Read Private Index we have $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, $\gamma(x) = (l', \text{private const } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma'_1(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty'*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_1(l'_1) = (\omega'_1, \text{private } bty', n', \text{PermL}(\text{Freeable}, \text{private } bty', \text{private}, n'))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{DecodeVal}(\text{private } bty', n', \omega'_1) = [v'_0, ..., v'_{n'-1}]$, and $v' = \bigvee_{m'=0}^{n'-1}(i' = \text{encrypt}(m')) \wedge v'_m$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $\sigma_1(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\sigma'_1(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty'*, \text{private}, 1))$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore,

we have $l_1 = l'_1$.

Given $\sigma_1(l_1) = (\omega_1, \text{private } bty, n, \text{PermL(Freeable, private } bty, \text{private}, n))$, $\sigma'_1(l'_1) = (\omega'_1, \text{private } bty', n', \text{PermL(Freeable, private } bty', \text{private}, n'))$, $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodeVal(private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{DecodeVal(private } bty', n', \omega'_1) = [v'_0, ..., v'_{n'-1}]$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v'_0, ..., v'_{n'-1}]$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v'_m$.

Given $v = \bigvee_{m=0}^{n-1}(i = \text{encrypt}(m)) \wedge v_m$ and $v' = \bigvee_{m'=0}^{n'-1}(i' = \text{encrypt}(m')) \wedge v'_m$, we have $m \in \{0, ..., n-1\}$ and $m' \in \{0, ..., n'-1\}$. Given $n = n'$, we have $m, m' \in \{0, ..., n-1\}$ and $m = m'$. Given $m = m'$, we have $\text{encrypt}(m) = \text{encrypt}(m')$. Given $\forall m \in \{0, ..., n-1\}$, $v_m = v'_m$ and $i = i'$, we have $v = v'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, $v = v'$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^t_{ra2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^t_{ra2} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$ by rule Location-tracking Public 1D Array Read Private Index, we have $\gamma(x) = (l, \text{public const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$, $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL(Freeable, public const } bty*, \text{public}, 1))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{DecodePtr(public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL(Freeable, public } bty, \text{public}, n))$, $\text{DecodeVal(public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, and $v = \bigvee_{m=0}^{n-1}(i = \text{encrypt}(m)) \wedge \text{encrypt}(v_m)$.

By definition 3.3.1, given $c = ra2$, we have $c \simeq_L c'$ if and only if $c' = ra2$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^t_{ra2} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, v')$ by rule Location-tracking Public 1D Array Read Private Index, we have $\gamma(x) = (l', \text{public const } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, $\sigma'_1(l') = (\omega', \text{public const } bty'*, 1, \text{PermL(Freeable, public const } bty'*, \text{public}, 1))$, $\text{Label}(e, \gamma) = \text{private}$, $\text{DecodePtr(public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_1(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL(Freeable, public } bty', \text{public}, n))$, $\text{DecodeVal(public } bty', n', \omega'_1) = [v'_0, ..., v'_{n'-1}]$, and $v' = \bigvee_{m'=0}^{n'-1}(i' = \text{encrypt}(m')) \wedge \text{encrypt}(v'_{m'})$.

Given $\gamma(x) = (l, \text{public const } bty*)$ and $\gamma(x) = (l', \text{public const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable, public const } bty*, \text{public}, 1))$, $\sigma'_1(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable, public const } bty'*, \text{public}, 1))$, $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr}(\text{public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we have $l_1 = l'_1$.

Given $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable, public } bty, \text{public}, n))$, $\sigma'_1(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL}(\text{Freeable, public } bty', \text{public}, n))$, $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodeVal}(\text{public } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{DecodeVal}(\text{public } bty', n', \omega'_1) = [v'_0, ..., v'_{n'-1}]$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v'_0, ..., v'_{n'-1}]$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v'_m$.

Given $v = \bigvee_{m=0}^{n-1}(i = \text{encrypt}(m)) \wedge \text{encrypt}(v_m)$, $v' = \bigvee_{m'=0}^{n'-1}(i' = \text{encrypt}(m')) \wedge \text{encrypt}(v'_{m'})$, $i = i'$, $n = n'$, and $\forall m \in \{0, ..., n-1\}$, $v_m = v'_m$, we have $m, m' \in \{0, ..., n-1\}$. By Axiom 3.3.1 we have $\forall m \in \{0, ..., n-1\}$, $\text{encrypt}(m) = \text{encrypt}(m')$ and $\text{encrypt}(v_m) = \text{encrypt}(v'_{m'})$. Therefore, we have $v = v'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $v = v'$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wa} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wa} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public 1D Array Write Public Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_2(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable, public const } bty*, \text{public}, 1))$,

DecodePtr(public const $bty*$, $1, \omega$) = [1, [$(l_1, 0)$], [1], 1], $\sigma_2(l_1)$ = (ω_1, public bty, n, PermL(Freeable, public bty, public, n)), DecodeVal(public bty, n, ω_1) = $[v_0, ..., v_{n-1}]$, $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{v}{v_i}\right)$, $0 \leq i \leq n-1$, and T_UpdateVal(σ_2, l_1, $[v'_0, ..., v'_{n-1}]$, Δ_2, $\overline{\chi}$, bid, acc, public bty) = (σ_3, Δ_3).

By definition 3.3.1, given $c = wa$, we have $c \simeq_L c'$ if and only if $c' = wa$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $x[e_1] = e_2) \Downarrow^t_{wa} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}$, bid, acc, skip) by rule Location-tracking Public 1D Array Write Public Value Public Index, we have Label(e_1, γ) = Label(e_2, γ) = public, acc = 0, (γ, σ, Δ, $\overline{\chi}$, bid, acc, e_1) $\Downarrow^t_{c'_1}$ (γ, σ'_1, Δ'_1, $\overline{\chi}$, bid, acc, i'), (γ, σ'_1, Δ'_1, $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c'_2}$ (γ, σ'_2, Δ'_2, $\overline{\chi}$, bid, acc, v''), $v'' \neq$ skip, $\gamma(x)$ = (l', public const $bty'*$), $\sigma'_2(l')$ = (ω', public const $bty'*$, 1, PermL(Freeable, public const $bty'*$, public, 1)), DecodePtr(public const $bty'*$, $1, \omega'$) = [1, [$(l'_1, 0)$], [1], 1], $\sigma'_2(l'_1)$ = (ω'_1, public bty', n', PermL(Freeable, public bty', public, n')), DecodeVal(public bty', n', ω'_1) = $[v''_0, ..., v''_{n'-1}]$, $[v'''_0, ..., v'''_{n'-1}] = [v''_0, ..., v''_{n-1}]\left(\frac{v''}{v''_{i'}}\right)$, $0 \leq i' \leq n'-1$, and T_UpdateVal(σ'_2, l'_1, $[v'''_0, ..., v'''_{n'-1}]$, Δ'_2, $\overline{\chi}$, bid, acc, public bty') = (σ'_3, Δ'_3).

Given (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, i) and (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c'_1}$ (γ, σ'_1, Δ'_1, $\overline{\chi}$, bid, acc, i'), by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c_2}$ (γ, σ_2, Δ_2, $\overline{\chi}$, bid, acc, v), (γ, σ'_1, Δ'_1, $\overline{\chi}$, bid, acc, e_2) $\Downarrow^t_{c'_2}$ (γ, σ'_2, Δ'_2, $\overline{\chi}$, bid, acc, v''), $\sigma_1 = \sigma'_1$, and $\Delta_1 = \Delta'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $v = v''$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x)$ = (l, public const $bty*$) and $\gamma(x)$ = (l', public const $bty'*$), we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l)$ = (ω, public const $bty*$, 1, PermL(Freeable, public const $bty*$, public, 1)), $\sigma'_2(l')$ = (ω', public const $bty'*$, 1, PermL(Freeable, public const $bty'*$, public, 1)), $\sigma_2 = \sigma'_2$, and $l = l'$, we have $\omega = \omega'$.

Given DecodePtr(public const $bty*$, $1, \omega$) = [1, [$(l_1, 0)$], [1], 1], DecodePtr(public const $bty'*$, 1, ω') = [1, [$(l'_1$, 0)], [1], 1], $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have [1, [$(l_1, 0)$], [1], 1] = [1, [$(l'_1, 0)$], [1], 1]. Therefore, we have $l_1 = l'_1$.

Given $\sigma_2(l_1)$ = (ω_1, public bty, n, PermL(Freeable, public bty, public, n)), $\sigma'_2(l'_1)$ = (ω'_1, public bty', n', PermL(Freeable, public bty', public, n')), $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given DecodeVal(public bty, n, ω_1) = $[v_0, ..., v_{n-1}]$, DecodeVal(public bty', n', ω'_1) = $[v''_0, ..., v''_{n'-1}]$, $bty = bty'$,

$n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0,\ ...,\ v_{n-1}] = [v''_0,\ ...,\ v''_{n'-1}]$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$.

Given $0 \leq i \leq n-1$, $0 \leq i' \leq n'-1$, $i = i'$, $n = n'$, and $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$, we have $v_i = v''_{i'}$.

Given $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{v}{v_i}\right)$, $[v'''_0, ..., v'''_{n'-1}] = [v''_0, ..., v''_{n'-1}]\left(\frac{v''}{v''_{i'}}\right)$, $[v_0,\ ...,\ v_{n-1}] = [v''_0,\ ...,\ v''_{n'-1}]$, $v = v''$, and $v_i = v''_{i'}$, we have $[v'_0, ..., v'_{n-1}] = [v'''_0, ..., v'''_{n'-1}]$.

Given $\text{T_UpdateVal}(\sigma_2,\ l_1,\ [v'_0, ..., v'_{n-1}],\ \Delta_2,\ \overline{\chi},\ \text{bid},\ \text{acc},\ \text{public}\ bty) = (\sigma_3,\ \Delta_3)$, $\text{T_UpdateVal}(\sigma'_2, l'_1, [v'''_0, ..., v'''_{n'-1}], \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, \text{public}\ bty') = (\sigma'_3, \Delta'_3)$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $l_1 = l'_1$, $bty = bty'$, and $[v'_0, ..., v'_{n-1}] = [v'''_0, ..., v'''_{n'-1}]$, by Lemma 3.3.12 we have $\sigma_3 = \sigma'_3$ and $\Delta_3 = \Delta'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\Delta_3 = \Delta'_3$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1]\ = e_2) \Downarrow^t_{wa4} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
This case is similar to case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1]\ = e_2) \Downarrow^t_{wa} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wa1} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1]\ = e_2) \Downarrow^t_{wa1} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private 1D Array Write Public Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const}\ bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_2(l) = (\omega, \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const}\ bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private}\ bty, n, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, n))$, $\text{DecodeVal}(\text{private}\ bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}]\left(\frac{\text{encrypt}(v)}{v_i}\right)$, $0 \leq i \leq n-1$, and $\text{T_UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{private}\ bty) = (\sigma_3, \Delta_3)$.

By definition 3.3.1, given $c = wa1$, we have $c \simeq_L c'$ if and only if $c' = wa1$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1]\ = e_2) \Downarrow^t_{wa1} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking

Private 1D Array Write Public Value Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, v'')$, $v'' \neq \text{skip}$, $\gamma(x) = (l', \text{private const } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_2(l'_1) = (\omega'_1, \text{private } bty, n', \text{PermL}(\text{Freeable}, \text{private } bty', \text{private}, n'))$, $\text{DecodeVal}(\text{private } bty', n', \omega'_1) = [v''_0, ..., v''_{n'-1}]$, $[v'''_0, ..., v'''_{n'-1}] = [v''_0, ..., v''_{n'-1}](\frac{\text{encrypt}(v'')}{v''_{i'}})$, $0 \leq i' \leq n' - 1$, and $\text{T_UpdateVal}(\sigma'_2, l'_1, [v'''_0, ..., v'''_{n'-1}], \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty') = (\sigma'_3, \Delta'_3)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, v)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, v'')$, $\sigma_1 = \sigma'_1$, and $\Delta_1 = \Delta'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $v = v''$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\sigma_2 = \sigma'_2$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we have $l_1 = l'_1$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $\sigma'_2(l'_1) = (\omega'_1, \text{private } bty, n', \text{PermL}(\text{Freeable}, \text{private } bty', \text{private}, n'))$, $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodeVal}(\text{private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{DecodeVal}(\text{private } bty', n', \omega'_1) = [v''_0, ..., v''_{n'-1}]$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v''_0, ..., v''_{n'-1}]$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$.

Given $0 \leq i \leq n - 1$, $0 \leq i' \leq n' - 1$, $i = i'$, $n = n'$, and $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$, we have $v_i = v''_{i'}$.

Given $[v'_0, ..., v'_{n-1}] = [v_0, ..., v_{n-1}](\frac{\text{encrypt}(v)}{v_i})$, $[v'''_0, ..., v'''_{n'-1}] = [v''_0, ..., v''_{n'-1}](\frac{\text{encrypt}(v'')}{v''_{i'}})$, $[v_0, ..., v_{n-1}] =$

$[v''_0, ..., v''_{n'-1}]$, $v = v''$, and $v_i = v''_{i'}$, by Axiom 3.3.1 we have $\text{encrypt}(v) = \text{encrypt}(v'')$. Therefore, we have $[v'_0, ..., v'_{n-1}] = [v'''_0, ..., v'''_{n'-1}]$.

Given T_UpdateVal(σ_2, l_1, $[v'_0, ..., v'_{n-1}]$, Δ_2, $\overline{\chi}$, bid, acc, private bty) = (σ_3, Δ_3), T_UpdateVal(σ'_2, l'_1, $[v'''_0, ..., v'''_{n'-1}]$, Δ'_2, $\overline{\chi}$, bid, acc, private bty') = (σ'_3, Δ'_3), $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $l_1 = l'_1$, $bty = bty'$, and $[v'_0, ..., v'_{n-1}] = [v'''_0, ..., v'''_{n'-1}]$, by Lemma 3.3.12 we have $\sigma_3 = \sigma'_3$ and $\Delta_3 = \Delta'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\Delta_3 = \Delta'_3$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wa2} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wa2} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private 1D Array Write Public Value Private Index, we have $\text{Label}(e_1, \gamma) = \text{private}$, $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{private}, 1))$, $\text{DecodePtr(private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL(Freeable, private } bty, \text{private}, n))$, $\text{DecodeVal(private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $(bty = \text{int}) \vee (bty = \text{float})$, $v' = \text{encrypt}(v)$, $\forall v_m \in [v_0, ..., v_{n-1}].\ v'_m = ((i = \text{encrypt}(m)) \wedge v') \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, and T_UpdateVal($\sigma_2$, l_1, $[v'_0, ..., v'_{n-1}]$, Δ_2, $\overline{\chi}$, bid, acc, private bty) = (σ_3, Δ_3).

By definition 3.3.1, given $c = wa2$, we have $c \simeq_L c'$ if and only if $c' = wa2$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wa2} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private 1D Array Write Public Value Private Index, we have $\text{Label}(e_1, \gamma) = \text{private}$, $\text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, v'')$, $v'' \neq \text{skip}$, $\gamma(x) = (l', \text{private const } bty'*)$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL(Freeable, private const } bty'*, \text{private}, 1))$, $\text{DecodePtr(private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_2(l'_1) = (\omega'_1, \text{private } bty', n', \text{PermL(Freeable, private } bty', \text{private}, n'))$, $\text{DecodeVal(private } bty', n', \omega'_1) = [v''_0, ..., v''_{n'-1}]$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $v''' = \text{encrypt}(v'')$, $\forall v''_{m'} \in [v''_0, ..., v''_{n'-1}].\ v'''_{m'} = ((i' = \text{encrypt}(m')) \wedge v''') \vee (\neg(i' = \text{encrypt}(m')) \wedge v''_{m'})$, and T_UpdateVal($\sigma'_2$, l'_1, $[v'''_0, ..., v'''_{n'-1}]$, Δ'_2, $\overline{\chi}$, bid, acc, private bty') = (σ'_3, Δ'_3).

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid},$

acc, i'), by the inductive hypothesis we have $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $i = i'$, and $c_1 \simeq_L c_1'$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, v)$, $(\gamma, \sigma_1', \Delta_1', \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2'} (\gamma, \sigma_2', \Delta_2', \overline{\chi}, \text{bid}, \text{acc}, v'')$, $\sigma_1 = \sigma_1'$, and $\Delta_1 = \Delta_1'$, by the inductive hypothesis we have $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $v = v''$, and $c_2 \simeq_L c_2'$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{private}, 1))$, $\sigma_2'(l') = (\omega', \text{private const } bty'*, 1, \text{PermL(Freeable, private const } bty'*, \text{private}, 1))$, $\sigma_2 = \sigma_2'$, and $l = l'$, we have $\omega = \omega'$.

Given $\text{DecodePtr(private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr(private const } bty'*, 1, \omega') = [1, [(l_1', 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l_1', 0)], [1], 1]$. Therefore, we have $l_1 = l_1'$.

Given $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL(Freeable, private } bty, \text{private}, n))$, $\sigma_2'(l_1') = (\omega_1', \text{private } bty', n', \text{PermL(Freeable, private } bty', \text{private}, n'))$, $\sigma_2 = \sigma_2'$, and $l_1 = l_1'$, we have $\omega_1 = \omega_1'$ and $n = n'$.

Given $\text{DecodeVal(private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $\text{DecodeVal(private } bty', n', \omega_1') = [v_0'', ..., v_{n'-1}'']$, $bty = bty'$, $n = n'$, and $\omega_1 = \omega_1'$, by Lemma 3.3.4 we have $[v_0, ..., v_{n-1}] = [v_0'', ..., v_{n'-1}'']$. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v_m''$.

Given $v' = \text{encrypt}(v)$, $v''' = \text{encrypt}(v'')$, and $v = v''$, by Axiom 3.3.1 we have $v' = v'''$.

Given $\forall v_m \in [v_0, ..., v_{n-1}].\ v_m' = ((i = \text{encrypt}(m)) \wedge v') \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, $\forall v_{m'}'' \in [v_0'', ..., v_{n'-1}''].\ v_{m'}''' = ((i' = \text{encrypt}(m')) \wedge v''') \vee (\neg(i' = \text{encrypt}(m')) \wedge v_{m'}'')$, $v' = v'''$, $i = i'$, $n = n'$, and $\forall m \in \{0, ..., n-1\}$, $v_m = v_m''$, we have $m, m' \in \{0, ..., n-1\}$ and by Axiom 3.3.1 we have $\text{encrypt}(m) = \text{encrypt}(m')$. Therefore, we have $[v_0', ..., v_{n-1}'] = [v_0''', ..., v_{n'-1}''']$.

Given $\text{T_UpdateVal}(\sigma_2, l_1, [v_0', ..., v_{n-1}'], \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty) = (\sigma_3, \Delta_3)$, $\text{T_UpdateVal}(\sigma_2', l_1', [v_0''', ..., v_{n'-1}'''], \Delta_2', \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty') = (\sigma_3', \Delta_3')$, $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $l_1 = l_1'$, $bty = bty'$, and $[v_0', ..., v_{n-1}'] = [v_0''', ..., v_{n'-1}''']$, by Lemma 3.3.12 we have $\sigma_3 = \sigma_3'$ and $\Delta_3 = \Delta_3'$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma_3'$, $\Delta_3 = \Delta_3'$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1,

we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wa3} (\gamma, \sigma_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wa3} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private 1D Array Write Private Value Private Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \text{acc}, \Delta, \overline{\chi}, \text{bid}, e_1) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{ private const } bty*)$, $\sigma_2(l) = (\omega, \text{ private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{private}, 1))$, $\text{DecodePtr(private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{ private } bty, n, \text{PermL(Freeable, private } bty, \text{private}, n))$, $\text{DecodeVal(private } bty, n, \omega_1) = [v_0, ..., v_{n-1}]$, $(bty = \text{int}) \vee (bty = \text{float})$, $\forall v_m \in [v_0, ..., v_{n-1}]. v'_m = ((i = \text{encrypt}(m)) \wedge v) \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, and $\text{T_UpdateVal}(\sigma_2, l_1, [v'_0, ..., v'_{n-1}], \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty) = (\sigma_3, \Delta_3)$.

By definition 3.3.1, given $c = wa3$, we have $c \simeq_L c'$ if and only if $c' = wa3$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wa3} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private 1D Array Write Private Value Private Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{private}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, v'')$, $v'' \neq \text{skip}$, $\gamma(x) = (l', \text{private const } bty'*)$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL(Freeable, private const } bty'*, \text{private}, 1))$, $\text{DecodePtr(private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_2(l'_1) = (\omega'_1, \text{ private } bty', n', \text{PermL(Freeable, private } bty', \text{private}, n'))$, $\text{DecodeVal(private } bty', n', \omega'_1) = [v''_0, ..., v''_{n'-1}]$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\forall v''_{m'} \in [v''_0, ..., v''_{n'-1}]. v'''_{m'} = ((i' = \text{encrypt}(m')) \wedge v'') \vee (\neg(i' = \text{encrypt}(m')) \wedge v''_{m'})$, and $\text{T_UpdateVal}(\sigma'_2, l'_1, [v'''_0, ..., v'''_{n'-1}], \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty') = (\sigma'_3, \Delta'_3)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, v)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, v'')$, $\sigma_1 = \sigma'_1$, and $\Delta_1 = \Delta'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $v = v''$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega, \text{ private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{private}, 1))$, $\sigma'_2(l') = (\omega', \text{private}$

const $bty'*$, 1, PermL(Freeable, private const $bty'*$, private, 1)), $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given DecodePtr(private const $bty*$, $1, \omega$) = [1, [(l_1, 0)], [1], 1], DecodePtr(private const $bty'*$, 1, ω') = [1, [(l'_1, 0)], [1], 1], $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have [1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]. Therefore, we have $l_1 = l'_1$.

Given $\sigma_2(l_1)$ = (ω_1, private bty, n, PermL(Freeable, private bty, private, n)), $\sigma'_2(l'_1)$ = (ω'_1, private bty', n', PermL(Freeable, private bty', private, n')), $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given DecodeVal(private bty, n, ω_1) = [v_0, ..., v_{n-1}], DecodeVal(private bty', n', ω'_1) = [v''_0, ..., $v''_{n'-1}$], $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have [v_0, ..., v_{n-1}] = [v''_0, ..., $v''_{n'-1}$]. Therefore, we have $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$.

Given $\forall v_m \in [v_0, ..., v_{n-1}]$. $v'_m = ((i = \text{encrypt}(m)) \wedge v) \vee (\neg(i = \text{encrypt}(m)) \wedge v_m)$, $\forall v''_{m'} \in [v''_0, ..., v''_{n'-1}]$. $v'''_{m'} = ((i' = \text{encrypt}(m')) \wedge v'') \vee (\neg(i' = \text{encrypt}(m')) \wedge v''_{m'})$, $v' = v'''$, $i = i'$, $n = n'$, and $\forall m \in \{0, ..., n-1\}$, $v_m = v''_m$, we have $m, m' \in \{0, ..., n-1\}$ and by Axiom 3.3.1 we have $\text{encrypt}(m) = \text{encrypt}(m')$. Therefore, we have [$v'_0$, ..., v'_{n-1}] = [v'''_0, ..., $v'''_{n'-1}$].

Given T_UpdateVal(σ_2, l_1, [v'_0, ..., v'_{n-1}], Δ_2, $\overline{\chi}$, bid, acc, private bty) = (σ_3, Δ_3), T_UpdateVal(σ'_2, l'_1, [v'''_0, ..., $v'''_{n'-1}$], Δ'_2, $\overline{\chi}$, bid, acc, private bty') = (σ'_3, Δ'_3), $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $l_1 = l'_1$, $bty = bty'$, and [v'_0, ..., v'_{n-1}] = [v'''_0, ..., $v'''_{n'-1}$], by Lemma 3.3.12 we have $\sigma_3 = \sigma'_3$ and $\Delta_3 = \Delta'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\Delta_3 = \Delta'_3$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^t_{rao} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^t_{rao} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$ by rule Location-tracking Public 1D Array Read Out of Bounds Public Index, we have Label(e, γ) = public, $\gamma(x)$ = (l, public const $bty*$), (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, i), $\sigma_1(l)$ = (ω, public const $bty*$, 1, PermL(Freeable, public const $bty*$, public, 1)), DecodePtr(public const $bty*$, $1, \omega$) = [1, [(l_1, 0)], [1], 1], $\sigma_1(l_1)$ = (ω_1, publicbty, n, PermL(Freeable, public bty, public, n)), $(i < 0) \vee (i \geq n)$, and ReadOOB(i, n, l_1, public bty, σ_1) = (v, 1).

By definition 3.3.1, given $c = rao$, we have $c \simeq_L c'$ if and only if $c' = rao$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^t_{rao} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, v')$ by rule Location-tracking Public 1D Array Read Out of Bounds Public Index, we have $\text{Label}(e, \gamma) = \text{public}$, $\gamma(x) = (l', \text{public const } bty'*)$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, $\sigma'_1(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty'*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_1(l'_1) = (\omega'_1, \text{public } bty, n', \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, n'))$, $(i' < 0) \vee (i' \geq n')$, and $\text{ReadOOB}(i', n', l'_1, \text{public } bty', \sigma'_1) = (v', 1)$.

Given $\gamma(x) = (l, \text{public const } bty*)$ and $\gamma(x) = (l', \text{public const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $\sigma_1(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\sigma'_1(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty'*, \text{public}, 1))$, $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr}(\text{public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we have $l_1 = l'_1$.

Given $\sigma_1(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $\sigma'_1(l'_1) = (\omega'_1, \text{public } bty, n', \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, n'))$, $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{ReadOOB}(i, n, l_1, \text{public } bty, \sigma_1) = (v, 1)$, $\text{ReadOOB}(i', n', l'_1, \text{public } bty', \sigma'_1) = (v', 1)$, $i = i'$, $n = n'$, $l_1 = l'_1$, $bty = bty'$, and $\sigma_1 = \sigma'_1$, by Lemma 3.3.24 we have $v = v'$.

Therefore, we have $\gamma = \gamma$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $v = v'$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^{t*}_{rao} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^{t}_{rao} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^{t}_{rao1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^{t}_{rao} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^{t*}_{rao1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e]) \Downarrow^{t}_{rao} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, v)$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^{t}_{wao} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^{t}_{wao} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public 1D Array Write Out of Bounds Public Index Public Value, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^{t}_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{public const } bty*)$, $\sigma_2(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{public } bty, n, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, n))$, $(i < 0) \vee (i \geq n)$, and $\text{WriteOOB}(v, i, n, l_1, \text{public } bty, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_3, \Delta_3, 1)$.

By definition 3.3.1, given $c = wao$, we have $c \simeq_L c'$ if and only if $c' = wao$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^{t}_{wao} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public 1D Array Write Out of Bounds Public Index Public Value, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $\text{acc} = 0$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^{t}_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, v')$, $v' \neq \text{skip}$, $\gamma(x) = (l', \text{public const } bty'*)$, $\sigma'_2(l') = (\omega', \text{public const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty'*, \text{public}, 1))$, $\text{DecodePtr}(\text{public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_2(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL}(\text{Freeable}, \text{public } bty', \text{public}, n'))$, $(i' < 0) \vee (i' \geq n')$, and $\text{WriteOOB}(v', i', n', l'_1, \text{public } bty', \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma'_3, \Delta'_3, 1)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, v)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^t_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, v'')$, $\Delta_1 = \Delta'_1$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $v = v''$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x) = (l, \text{public const } bty*)$ and $\gamma(x) = (l', \text{public const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega, \text{public const } bty*, 1, \text{PermL(Freeable, public const } bty*, \text{public}, 1))$, $\sigma'_2(l') = (\omega', \text{public const } bty'*, 1, \text{PermL(Freeable, public const } bty'*, \text{public}, 1))$, $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{DecodePtr(public const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\text{DecodePtr(public const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore, we have $l_1 = l'_1$.

Given $\sigma_2(l_1) = (\omega_1, \text{public } bty, n, \text{PermL(Freeable, public } bty, \text{public}, n))$, $\sigma'_2(l'_1) = (\omega'_1, \text{public } bty', n', \text{PermL(Freeable, public } bty', \text{public}, n'))$, $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $\text{WriteOOB}(v, i, n, l_1, \text{public } bty, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_3, \Delta_3, 1)$, $\text{WriteOOB}(v', i', n', l'_1, \text{public } bty', \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma'_3, \Delta'_3 1)$, $v = v'$, $i = i'$, $n = n'$, $l_1 = l'_1$, $bty = bty'$, $\Delta_2 = \Delta'_2$, and $\sigma_2 = \sigma'_2$, by Lemma 3.3.25 we have $\sigma_3 = \sigma'_3$ and $\Delta_3 = \Delta'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\Delta_3 = \Delta'_3$, $\overline{\chi} = \overline{\chi}$, $\text{bid} = \text{bid}$, $\text{acc} = \text{acc}$, $\text{skip} = \text{skip}$, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^{t*}_{wao} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wao} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wao2} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^t_{wao} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^{t*}_{wao2} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^{t}_{wao} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^{t}_{wao1} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^{t}_{wao1} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private 1D Array Write Public Value Out of Bounds Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$, $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^{t}_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, v)$, $v \neq \text{skip}$, $\gamma(x) = (l, \text{private const } bty*)$, $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\sigma_2(l_1) = (\omega_1, \text{private } bty, n, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, n))$, $(i < 0) \vee (i \geq n)$, and $\text{WriteOOB}(\text{encrypt}(v), i, n, l_1, \text{private } bty, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma_3, \Delta_3, 1)$.

By definition 3.3.1, given $c = wao1$, we have $c \simeq_L c'$ if and only if $c' = wao1$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^{t}_{wao1} (\gamma, \sigma'_3, \Delta'_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private 1D Array Write Public Value Out of Bounds Public Index, we have $\text{Label}(e_1, \gamma) = \text{Label}(e_2, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e_1) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^{t}_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, v')$, $v' \neq \text{skip}$, $\gamma(x) = (l', \text{private const } bty'*)$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty'*, \text{private}, 1))$, $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_2(l'_1) = (\omega'_1, \text{private } bty', n', \text{PermL}(\text{Freeable}, \text{private } bty', \text{private}, n'))$, $(i' < 0) \vee (i' \geq n')$, and $\text{WriteOOB}(\text{encrypt}(v'), i', n', l'_1, \text{private } bty', \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}) = (\sigma'_3, \Delta'_3, 1)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, i)$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^{t}_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, i')$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $i = i'$, and $c_1 \simeq_L c'_1$.

Given $(\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^{t}_{c_2} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, v)$, $(\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, e_2) \Downarrow^{t}_{c'_2} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, v'')$, $\Delta_1 = \Delta'_1$, and $\sigma_1 = \sigma'_1$, by the inductive hypothesis we have $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $v = v''$, and $c_2 \simeq_L c'_2$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, $\sigma'_2(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty'*, \text{private}, 1))$, $\sigma_2 = \sigma'_2$, and $l = l'$, we have $\omega = \omega'$.

Given DecodePtr(private const $bty*$, $1, \omega$) $=$ [1, [$(l_1, 0)$], [1], 1], DecodePtr(private const $bty'*$, 1, ω') $=$ [1, [$(l'_1, 0)$], [1], 1], $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have [1, [$(l_1, 0)$], [1], 1] $=$ [1, [$(l'_1, 0)$], [1], 1]. Therefore we have $l_1 = l'_1$.

Given $\sigma_2(l_1) = (\omega_1$, private bty, n, PermL(Freeable, private bty, private, n)), $\sigma'_2(l'_1) = (\omega'_1$, private bty', n', PermL(Freeable, private bty', private, n')), $\sigma_2 = \sigma'_2$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given $v = v'$, by Axiom 3.3.1 we have encrypt(v) $=$ encrypt(v').

Given WriteOOB(encrypt(v), i, n, l_1, private bty, σ_2, Δ_2, $\overline{\chi}$, bid, acc) $= (\sigma_3, \Delta_3, 1)$, WriteOOB(encrypt($v'$), i', n', l'_1, private bty', σ'_2, Δ'_2, $\overline{\chi}$, bid, acc) $= (\sigma'_3, \Delta'_3, 1)$, encrypt($v$) $=$ encrypt(v'), $i = i'$, $n = n'$, $l_1 = l'_1$, $bty = bty'$, $\Delta_2 = \Delta'_2$, and $\sigma_2 = \sigma'_2$, by Lemma 3.3.25 we have $\sigma_3 = \sigma'_3$ and $\Delta_3 = \Delta'_3$.

Therefore, we have $\gamma = \gamma$, $\sigma_3 = \sigma'_3$, $\Delta_3 = \Delta'_3$, $\overline{\chi} = \overline{\chi}$, bid $=$ bid, acc $=$ acc, skip $=$ skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^{t*}_{wao} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
This case is similar to case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x[e_1] = e_2) \Downarrow^{t}_{wao1} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^{t}_{ra5} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^{t}_{ra5} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [v_0, ..., v_{n-1}])$ by rule Location-tracking Private 1D Array Read Entire Array, we have $\gamma(x) = (l$, private const $bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma(l) = (\omega$, private const $bty*$, 1, PermL(Freeable, private const $bty*$, private, 1)), DecodePtr(private const $bty*$, $1, \omega$) $=$ [1, [$(l_1$, 0)], [1], 1], $\sigma(l_1) = (\omega_1$, private bty, n, PermL(Freeable, private bty, private, n)), and DecodeVal(private bty, n, ω_1) $= [v_0, ..., v_{n-1}]$.

By definition 3.3.1, given $c = ra5$, we have $c \simeq_L c'$ if and only if $c' = ra5$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^{t}_{ra5} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [v'_0, ..., v'_{n'-1}])$ by rule Location-tracking Private 1D Array Read Entire Array, we have $\gamma(x) = (l'$, private const $bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma(l') = (\omega'$, private

const $bty'*$, 1, PermL(Freeable, private const $bty'*$, private, 1)), DecodePtr(private const $bty'*$, $1, \omega'$) = [1, [(l'_1, 0)], [1], 1], $\sigma(l'_1)$ = (ω'_1, private bty', n', PermL(Freeable, private bty', private, n')), and DecodeVal(private bty', n', ω'_1) = [v'_0, ..., $v'_{n'-1}$].

Given $\gamma(x)$ = (l, private const $bty*$) and $\gamma(x)$ = (l', private const $bty'*$), we have $l = l'$ and $bty = bty'$.

Given $\sigma(l)$ = (ω, private const $bty*$, 1, PermL(Freeable, private const $bty*$, private, 1)), $\sigma(l')$ = (ω', private const $bty'*$, 1, PermL(Freeable, private const $bty'*$, private, 1)), and $l = l'$, we have $\omega = \omega'$.

Given DecodePtr(private const $bty*$, $1, \omega$) = [1, [(l_1, 0)], [1], 1], DecodePtr(private const $bty'*$, 1, ω') = [1, [(l'_1, 0)], [1], 1], $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have [1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]. Therefore we have $l_1 = l'_1$.

Given $\sigma(l_1)$ = (ω_1, private bty, n, PermL(Freeable, private bty, private, n)), $\sigma(l'_1)$ = (ω'_1, private bty', n', PermL(Freeable, private bty', private, n')), and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given DecodeVal(private bty, n, ω_1) = [v_0, ..., v_{n-1}], DecodeVal(private bty', n', ω'_1) = [v'_0, ..., $v'_{n'-1}$], $bty = bty'$, $n = n'$, and $\omega_1 = \omega'_1$, by Lemma 3.3.4 we have [v_0, ..., v_{n-1}] = [v'_0, ..., $v'_{n'-1}$].

Therefore, we have $\gamma = \gamma$, $\sigma = \sigma$, $\Delta = \Delta$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, [v_0, ..., v_{n-1}] = [v'_0, ..., $v'_{n'-1}$], and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{ra4} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [v_0, ..., v_{n-1}])$
This case is similar to case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x) \Downarrow^t_{ra5} (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, [v_0, ..., v_{n-1}])$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e_1) \Downarrow^t_{wa5} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$
Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e) \Downarrow^t_{wa5} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Public 1D Array Write Entire Array, we have Label(e, γ) = public, (γ, σ, Δ, $\overline{\chi}$, bid, acc, e) $\Downarrow^t_{c_1}$ (γ, σ_1, Δ_1, $\overline{\chi}$, bid, acc, [v_0, ..., v_{n_e-1}]), $\forall v_m \in$ [v_0, ..., v_{n_e-1}]. $v_m \neq$ skip, $\gamma(x)$ = (l, public const $bty*$), $\sigma_1(l)$ = (ω, public const $bty*$, 1, PermL(Freeable, public const $bty*$, public, 1)), DecodePtr(public const $bty*$, $1, \omega$) = [1, [(l_1, 0)], [1], 1], $\sigma_1(l_1)$ = (ω_1, public bty, n, PermL(Freeable, public bty, public, n)), $n_e = n$, and T_UpdateVal(σ_1, l_1, [v_0, ...,

$v_{n_e-1}]$, Δ_1, $\overline{\chi}$, bid, acc, public $bty) = (\sigma_2, \Delta_2)$.

By definition 3.3.1, given $c = wa5$, we have $c \simeq_L c'$ if and only if $c' = wa5$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $x = e) \Downarrow^t_{wa5} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}$, bid, acc, skip) by rule Location-tracking Public 1D Array Write Entire Array, we have Label$(e, \gamma) =$ public, $(\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}$, bid, acc, $[v'_0$, ..., $v'_{n'_e-1}])$, $\forall v'_m \in [v'_0, ..., v'_{n'_e-1}]$. $v'_m \neq$ skip, $\gamma(x) = (l'$, public const $bty'*)$, $\sigma'_1(l') = (\omega'$, public const $bty'*$, 1, PermL(Freeable, public const $bty'*$, public, 1)), DecodePtr(public const $bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\sigma'_1(l'_1) = (\omega'_1$, public bty', n', PermL(Freeable, public bty', public, $n'))$, $n'_e = n'$, and T_UpdateVal$(\sigma'_1, l'_1, [v'_0$, ..., $v'_{n'_e-1}]$, Δ, $\overline{\chi}$, bid, acc, public $bty') = (\sigma'_2, \Delta'_2)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}$, bid, acc, $[v_0, ..., v_{n_e-1}])$ and $(\gamma, \sigma, \Delta, \overline{\chi}$, bid, acc, $e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}$, bid, acc, $[v'_0, ..., v'_{n'_e-1}])$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $[v_0, ..., v_{n_e-1}] = [v'_0, ..., v'_{n'_e-1}]$, and $c_1 \simeq_L c'_1$.

Given $\gamma(x) = (l$, public const $bty*)$ and $\gamma(x) = (l'$, public const $bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega$, public const $bty*$, 1, PermL(Freeable, public const $bty*$, public, 1)), $\sigma'_1(l') = (\omega'$, public const $bty'*$, 1, PermL(Freeable, public const $bty'*$, public, 1)), $\sigma_1 = \sigma'_1$, and $l = l'$, we have $\omega = \omega'$.

Given DecodePtr(public const $bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, DecodePtr(public const $bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l'_1, 0)], [1], 1]$. Therefore we have $l_1 = l'_1$.

Given $\sigma_1(l_1) = (\omega_1$, public bty, n, PermL(Freeable, public bty, public, $n))$, $\sigma'_1(l'_1) = (\omega'_1$, public bty', n', PermL(Freeable, public bty', public, $n'))$, $\sigma_1 = \sigma'_1$, and $l_1 = l'_1$, we have $\omega_1 = \omega'_1$ and $n = n'$.

Given T_UpdateVal$(\sigma_1, l_1, [v_0, ..., v_{n_e-1}], \Delta_1, \overline{\chi}$, bid, acc, public $bty) = (\sigma_2, \Delta_2)$, T_UpdateVal$(\sigma'_1, l'_1, [v'_0, ..., v'_{n'_e-1}], \Delta'_1, \overline{\chi}$, bid, acc, public $bty') = (\sigma'_2, \Delta'_2)$, $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $l_1 = l'_1$, $bty = bty'$, and $[v_0, ..., v_{n_e-1}] = [v'_0, ..., v'_{n'_e-1}]$, by Lemma 3.3.12 we have $\sigma_2 = \sigma'_2$ and $\Delta_2 = \Delta'_2$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e_1) \Downarrow^t_{wa6} (\gamma, \sigma_3, \Delta_3, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

This case is similar to Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e_1) \Downarrow^t_{wa5} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$.

Case $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e_1) \Downarrow^t_{wa7} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$

Given $\Pi \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e_1) \Downarrow^t_{wa7} (\gamma, \sigma_2, \Delta_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private 1D Array Write Entire Public Array, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [v_0, ..., v_{n_e-1}])$, $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v_m \neq \text{skip}$ $\gamma(x) = (l, \text{ private const } bty*)$, $(bty = \text{int}) \vee (bty = \text{float})$, $\sigma_1(l) = (\omega, \text{ private const } bty*, 1, \text{PermL(Freeable, private const } bty*, \text{private}, 1))$, $\text{DecodePtr(private const } bty*, 1, \omega) = [1, [(l_1, 0)], [1], 1]$, $\forall v_m \in [v_0, ..., v_{n_e-1}].\ v'_m = \text{encrypt}(v_m)$, $\sigma_1(l_1) = (\omega_1, \text{ private } bty, n, \text{PermL(Freeable, private } bty, \text{private}, n))$, $n_e = n$, and $\text{T_UpdateVal}(\sigma_1, l_1, [v'_0, ..., v'_{n_e-1}], \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty) = (\sigma_2, \Delta_2)$.

By definition 3.3.1, given $c = wa7$, we have $c \simeq_L c'$ if and only if $c' = wa7$.

Given $\Sigma \triangleright (\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, x = e_1) \Downarrow^t_{wa7} (\gamma, \sigma'_2, \Delta'_2, \overline{\chi}, \text{bid}, \text{acc}, \text{skip})$ by rule Location-tracking Private 1D Array Write Entire Public Array, we have $\text{Label}(e, \gamma) = \text{public}$, $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, [v''_0, ..., v''_{n'_e-1}])$, $\forall v''_m \in [v''_0, ..., v''_{n'_e-1}].\ v_m \neq \text{skip}$ $\gamma(x) = (l', \text{private const } bty'*)$, $(bty' = \text{int}) \vee (bty' = \text{float})$, $\sigma'_1(l') = (\omega', \text{private const } bty'*, 1, \text{PermL(Freeable, private const } bty'*, \text{private}, 1))$, $\text{DecodePtr(private const } bty'*, 1, \omega') = [1, [(l'_1, 0)], [1], 1]$, $\forall v''_{m'} \in [v''_0, ..., v''_{n'_e-1}].\ v'''_{m'} = \text{encrypt}(v''_{m'})$, $\sigma'_1(l'_1) = (\omega'_1, \text{ private } bty', n', \text{PermL(Freeable, private } bty', \text{private}, n'))$, $n'_e = n'$, and $\text{T_UpdateVal}(\sigma'_1, l'_1, [v'''_0, ..., v'''_{n'_e-1}], \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, \text{private } bty') = (\sigma'_2, \Delta'_2)$.

Given $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c_1} (\gamma, \sigma_1, \Delta_1, \overline{\chi}, \text{bid}, \text{acc}, [v_0, ..., v_{n_e-1}])$ and $(\gamma, \sigma, \Delta, \overline{\chi}, \text{bid}, \text{acc}, e) \Downarrow^t_{c'_1} (\gamma, \sigma'_1, \Delta'_1, \overline{\chi}, \text{bid}, \text{acc}, [v''_0, ..., v''_{n'_e-1}])$, by the inductive hypothesis we have $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $[v_0, ..., v_{n_e-1}] = [v''_0, ..., v''_{n'_e-1}]$, and $c_1 \simeq_L c'_1$.

Given $[v_0, ..., v_{n_e-1}] = [v''_0, ..., v''_{n'_e-1}]$, we have $n_e = n'_e$.

Given $\gamma(x) = (l, \text{private const } bty*)$ and $\gamma(x) = (l', \text{private const } bty'*)$, we have $l = l'$ and $bty = bty'$.

Given $\sigma_1(l) = (\omega,$ private const $bty*$, 1, PermL(Freeable, private const $bty*$, private, 1)), $\sigma_1'(l') = (\omega'$, private const $bty'*$, 1, PermL(Freeable, private const $bty'*$, private, 1)), $\sigma_1 = \sigma_1'$, and $l = l'$, we have $\omega = \omega'$.

Given DecodePtr(private const $bty*$, $1, \omega) = [1, [(l_1, 0)], [1], 1]$, DecodePtr(private const $bty'*$, 1, $\omega') = [1, [(l_1', 0)], [1], 1]$, $bty = bty'$, and $\omega = \omega'$, by Lemma 3.3.6 we have $[1, [(l_1, 0)], [1], 1] = [1, [(l_1', 0)], [1], 1]$. Therefore we have $l_1 = l_1'$.

Given $\forall v_m \in [v_0, ..., v_{n_e-1}]$. $v_m' = \text{encrypt}(v_m)$, $\forall v_{m'}'' \in [v_0'', ..., v_{n_e'-1}'']$. $v_{m'}''' = \text{encrypt}(v_{m'}'')$, $n_e = n_e'$, and $[v_0, ..., v_{n_e-1}] = [v_0'', ..., v_{n_e'-1}'']$, by Axiom 3.3.1 we have $\text{encrypt}(v_m) = \text{encrypt}(v_{m'}'')$ and therefore $[v_0', ..., v_{n_e-1}'] = [v_0''', ..., v_{n_e'-1}''']$.

Given $\sigma_1(l_1) = (\omega_1$, private bty, n, PermL(Freeable, private bty, private, n)), $\sigma_1'(l_1') = (\omega_1'$, private bty', n', PermL(Freeable, private bty', private, n')), $\sigma_1 = \sigma_1'$, and $l_1 = l_1'$, we have $\omega_1 = \omega_1'$ and $n = n'$.

Given T_UpdateVal(σ_1, l_1, $[v_0', ..., v_{n_e-1}']$, Δ, $\overline{\chi}$, bid, acc, private bty) $= (\sigma_2, \Delta_2)$, T_UpdateVal(σ_1', l_1', $[v_0''', ..., v_{n_e'-1}''']$, Δ, $\overline{\chi}$, bid, acc, private bty') $= (\sigma_2', \Delta_2')$, $\sigma_1 = \sigma_1'$, $\Delta_1 = \Delta_1'$, $l_1 = l_1'$, $bty = bty'$, and $[v_0', ..., v_{n_e-1}'] = [v_0''', ..., v_{n_e'-1}''']$, by Lemma 3.3.12 we have $\sigma_2 = \sigma_2'$ and $\Delta_2 = \Delta_2'$.

Therefore, we have $\gamma = \gamma$, $\sigma_2 = \sigma_2'$, $\Delta_2 = \Delta_2'$, $\overline{\chi} = \overline{\chi}$, bid = bid, acc = acc, skip = skip, and, by definition 4.3.1, we have $\Pi \simeq_L \Sigma$.

□

5 Multiparty SMC^2

In this Chapter, we present Multiparty SMC^2, a formalization of a general-purpose SMC compiler with explicit multiparty semantics and optimized conditional code block tracking. In both Basic SMC^2 and Location-tracking SMC^2, we had formalized the multiparty operations implicitly, assuming that each party would be doing the same thing and the portions that required communication would do so implicitly. However, we noted that this left a gap in understanding between the formal semantics and how the model would actual behave in a multiparty environment. We have resolved this gap within Multiparty SMC^2, explicitly showing the configurations of each party involved in the evaluation of a program, when multiparty operations will occur, and what data each party supplies to these operations. Multiparty SMC^2 also provides an enhanced ability to substitute SMC protocols as newer, improved protocols are developed, with each SMC protocol being called from within the semantics and shown separately as an algorithm. With this compartmentalization, the main formal model and proofs will not need to be updated as protocols are substituted in; these substitutions simply require updating the algorithms for the model and the proofs of the axioms for the given protocol to ensure it upholds the properties required for the proofs.

Additionally, to be able to handle pointers and arrays correctly, we must use a location based tracking instead of a variable based tracking, which we formalized in Location-tracking SMC^2, discussed in Chapter 4. However, as our Location-tracking SMC^2 semantic have illustrated, this requires additional tracking structures and dynamically checking to ensure that the locations modified during pointer dereference write statements are tracked. We, thus, propose a small optimization to full location tracking in Multiparty SMC^2, which analyzes each top-level private-conditioned branching statement to see if it contains a pointer dereference writes or array writes at public indices in the **`then`** or **`else`** clauses, as well as in any nested branching statements present in those clauses. If no such writes occur, we are able to use simple variable tracking, as shown in rule Private If Else Variable Tracking in Figure 5.16.

When any such write statement occurs in either branch, we switch to tracking by location, as shown in rule Private If Else Location Tracking in Figure 5.16. For example, consider a program using a pointer to iterate through and modify elements of an array. Allowing pointer dereference writes enables us to

perform a different operation on the array depending on whether a private condition holds. For location based conditional code block tracking, we create a map to store the `original` and **`then`** values for each location that is modified within each branch, as well as a tag to indicate whether this location has a value stored for the **`then`** branch. This tag ensures that even when a location is modified for the first time in the **`else`** branch, we are still able to properly resolve the value for that location by using the original value stored at that location. This corresponds to the rules and explanation given in subsection 5.1.2.

As with the *conditional code block* variable tracking scheme, first we find a list of all modified variables, excluding those only used for pointer dereference writes or array writes at a public index. We exclude pointer dereference writes as we will grab the location that is pointed to dynamically to ensure we are tracking the modification at the correct location. We use this list to store the original values at the location referred to by each of these variables before the execution of the **`then`** branch. Between branches, our restoration is similar to that formalized for Basic SMC^2, just by location. We iterate through our map, storing the current value of each location as the **`then`** value, and restoring the value at the location as the original value. We set the tag associated with each location to be 1, as we have added **`then`** values for each of these locations. When executing the **`then`** and **`else`** branches, we check before the execution of a pointer dereference write to see if the location we will modify is already being tracked. If it is not, we store the current value as the original value for that location, and then continue to execute as normal; if it is already tracked, we proceed as we do not need to store anything additional. We set the tag for each new location to 0, as we do not currently have a **`then`** value stored for those locations. After the execution of the **`else`** branch, we proceed to resolve similarly to Basic SMC^2, just by location. For each location in the map, we securely compute whether to keep the **`then`** value (or `original` value, if the tag is 0) stored in the map or the current value at that location based on the private condition.

5.1 Formal Semantics

$$
\begin{array}{rlcl}
C & \in \mathit{Configuration} & ::= & \epsilon \mid (\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s) \parallel C \\
\delta & \in \mathit{LocationMap} & ::= & f : (l, \mu) \rightarrow (v, v, j, ty) \\
\mathcal{D} & \in \mathit{PartyEvaluationCodeList} & ::= & \epsilon \mid (\mathrm{p}, \overline{d}) \parallel \mathcal{D} \\
\mathrm{p}, \mathrm{q} & \in \mathit{PartyIdentifier} & \in \mathbb{N} &
\end{array}
$$

Figure 5.1: Multiparty SMC^2 configuration: party identifier p, environment γ, memory σ, location map Δ, accumulator acc, and statement s.

In this section, we will present the Multiparty SMC^2 semantics with respect to the grammar (Figure 3.1). The semantic judgements in Multiparty SMC^2 are defined over six-tuple configurations $C = (\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s)$, where each party has its own configuration and each rule is a reduction from a set of party-wise configurations $C^1 \parallel ... \parallel C^{\mathrm{q}}$ to a subsequent. We denote the party's identifier as p; the environment as γ; memory as σ; location map Δ; the level of nesting of private-conditioned branches as acc; and a big-step evaluation of a statement s to a value v using $\Downarrow$. We annotate each evaluation with evaluation codes (i.e., $\Downarrow_d$) to facilitate reasoning over evaluation trees, and we annotate evaluations that are not *well-aligned* with a star (i.e., $\Downarrow_d^*$) to identify the rules that we cannot prove correctness over, as they produce unpredictable behavior. The assertions in each semantic rule can be read in sequential order, from left to right and top to bottom.

When proving correctness for Multiparty SMC^2, we will use the Multiparty Vanilla C semantics, shown in the following section. We chose to develop an additional set of Vanilla C semantics which allows us to show multiple parties evaluating at the same time in order to allow us to easily reason about the correctness of Multiparty SMC^2. In the Multiparty Vanilla C semantics, the parties do not communicate, they simply all evaluate the same program at the same time, and we provide "synchronization" rules that require all parties to be evaluating the same rule at the same time. These "synchronization" rules allow us to easily prove the correctness of the *interactive* Multiparty SMC^2 rules also requiring "synchronization" in order to evaluate the multiparty protocols requiring communication. The Multiparty Vanilla C configuration mirrors the Multiparty SMC^2 configuration, replacing location map Δ and accumulator acc each with $\square$, as they are unused with Multiparty Vanilla C.

In this semantics, we introduce party identifiers p and modifications to location map Δ to better manage tracking changes by location within private-conditioned branches. We will briefly discuss each here, and more fully discuss their uses when we discuss the private-conditioned **`if else`** statement and its algorithms. The party identifiers allow us to easily keep track of which configuration belongs to which party, and which statement each party is currently evaluating. This is useful in both the "synchronization" rules, where we view all parties at the same time as we are evaluating the rule, as well as in the single party rules, where we need to keep track of which party is evaluating the rule to be able to properly add to our party-wise location trace and our party-wise evaluation code trace.

We modify location map Δ, our list of maps δ, one for each level of nesting within the current private-conditioned **`if else`** statement. Like in Location-tracking SMC^2, at the start of a private-conditioned **`if else`** statement, we append a new sublist to Δ to constitute the current scope of changes within this

specific private-conditioned **`if else`** statement. This sublist is removed from Δ once we have completed resolution and the changes are out of scope. Of course, it is not removed before changes are propagated to any higher level of tracking when we are in a nested private-conditioned **`if else`** statement – this is discussed in more detail when we discuss the private-conditioned **`if else`** statement and the algorithms used within it. Each location map δ now maps memory block identifiers to a four-tuple of information – the original value stored in the block, the value stored within the block by the end of the **`then`** branch, a public tag indicating whether the value was modified within the **`then`** branch, and the type for the memory block. The public tag is set to 0 upon a new mapping being added to Δ, and updated to 1 during the restoration phase between branches, when we store the **`then`** value before restoring the `original` values. This removes the need to track what branch we are in, if any, at all times, as we can use this tag during the resolution of true values to tell us whether we have a **`then`** value stored or whether we need to use the `original` value as the value from the evaluation of the **`then`** branch.

It is important to note here that the two values stored in each mapping will be of the type given by δ (which is identical to the expected type of that memory block), and either a singular value, a list of values, or a pointer data structure. These mappings are only accessed at specific times: (1) when we are initializing known locations that will change before evaluation of either branch, (2) when we encounter a potentially unknown location (i.e., a non-absolute location, such as from a pointer dereference write or array write at a public index), (3) when we store the **`then`** values in between branches before restoring the `original` values, and (4) when we resolve the true values after finishing evaluation of both branches. In this way, we attempt to reduce the number of times we need to perform lookups within the location map, as all modifications for the first level of pointers, for regular variables, and for updating an entire array or an array at a private index no longer need to be tracked at every occurance.

5.1.1 Multiparty Vanilla C

In order to facilitate the correspondence between the Multiparty Vanilla C and Multiparty SMC^2 semantics, we continue to model our semantics using big-step evaluation judgements and define our Multiparty Vanilla C semantics with respect to multiple *non interacting* parties that evaluate the same program. In Multiparty Vanilla C, we use $\widehat{\ }$ to distinguish elements in this semantics from those we use in the next section for Multiparty SMC^2 semantics, which may differ due to privacy labels and private data being encrypted. The semantic judgements in Multiparty Vanilla C are defined over a six-tuple configuration $\widehat{C} = (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s})$

where each party has its own configuration and each rule is a reduction from a set of party-wise configurations $\widehat{C}^1 \parallel ... \parallel \widehat{C}^{\mathrm{q}}$, and each rule is a reduction from one configuration to a subsequent. We denote the party's identifier as p; the environment as $\widehat{\gamma}$; memory as $\widehat{\sigma}$; two placeholders as $\square, \square$; and a big-step evaluation of a statement $\widehat{s}$ to a value $\widehat{v}$ using $\Downarrow'$. We use $\square, \square$ in Multiparty Vanilla C as placeholders for the both the location map Δ and the level of nesting of private-conditioned branches acc in order to maintain the same shape of configurations as that of Multiparty SMC^2 used in the next section. We annotate each evaluation with evaluation codes (i.e., $\Downarrow'_{\widehat{d}}$) to facilitate reasoning over evaluation trees, and we annotate evaluations that are not *well-aligned* with a star (i.e., $\Downarrow'^{*}_{\widehat{d}}$) to identify the rules that we cannot prove correctness over, as they produce unpredictable behavior in implementation. We do not show the semantic rules that are not *well-aligned* in this chapter, as they are nearly identical to their corresponding rules and the proof of noninterference over these rules are handled similarly to the cases of the corresponding rules. Such rules are shown in the basic Vanilla C semantics for the interested reader. The assertions in each semantic rule can be read in sequential order, from left to right and top to bottom.

In this section, we will present the Multiparty Vanilla C semantics with respect to the grammar (Figure 3.1). These semantic rules follow standard C. Recall the Vanilla C semantics in Section 3.1.3 – the Multiparty Vanilla C semantics are not significantly different, except in the configuration, which is discussed above. It is worthwhile to note here that all permissions in Multiparty Vanilla C will be set to public, and all types will be implicitly public, as there is no notion of privacy labels in standard C. We will store pointer data within the pointer data structure to facilitate reasoning about pointers between Multiparty Vanilla C and Multiparty SMC^2, but the Multiparty Vanilla C pointers can only have a single location and will always have the single tag in the tag list set to 1, as that is the only possible location for the pointer to refer to. It is worthwhile to note again that we show the rules with multiple parties evaluating the same program to be able to reason about the behavior and structure of the multiparty semantics to prove correctness, but the parties never interact with each other in Multiparty Vanilla C.

Multiparty evaluation of rules are shown in Figures 5.2 and 5.3 only for Multiparty Vanilla C. Figure 5.2 gives the semantic rules for multiparty binary operations, multiparty if else statements, and multiparty pre-increment float variables. Figure 5.3 gives the semantic rules multiparty pointer dereferences, multiparty free, and multiparty array operations. All other figures give single party evaluation of statements. Figure 5.4 gives the semantic rules binary operations. Figure 5.5 gives the semantic rules for sequencing; obtaining the size of a type or location of a variable; declarations, pre-incrementing, reading and writing to regular

(non-pointer, non-array) variables; reading from a pointer; and loops. Figure 5.6 gives the semantic rules for if else statements, function operations, and casting values. Figure 5.7 gives the semantic rules for input and output of data, memory allocation and deallocation, and casting a location. Figure 5.8 gives the semantic rules for pointer operations. Figure 5.9 gives the semantic rules for array operations. Figure 5.10 gives the semantic rules for out-of-bounds array operations.

Multiparty Binary Operation

$$\frac{\begin{array}{c}((1,\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1)) \Downarrow'_{\widehat{\mathcal{D}}_1} ((1,\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1)) \\ ((1,\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{e}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_2} ((1,\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2)) \\ \widehat{n}_1\, bop\, \widehat{n}_2 = \widehat{n}_3 \qquad bop \in \{\cdot,+,-,\div\}\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}_1\, bop\, \widehat{e}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}_1\, bop\, \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{ALL},[\widehat{mpb}])} \\ ((1,\widehat{\gamma},\widehat{\sigma}_2,\square,\square,\widehat{n}_3) \qquad \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,\widehat{n}_3))\end{array}}$$

Multiparty Comparison True Operation

$$\frac{\begin{array}{c}((1,\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1)) \Downarrow'_{\widehat{\mathcal{D}}_1} ((1,\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1)) \\ ((1,\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{e}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_2} ((1,\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2)) \\ (\widehat{n}_1\, bop\, \widehat{n}_2) = 1 \qquad bop \in \{==,!=,<\}\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\ \square,\square,\widehat{e}_1\, bop\, \widehat{e}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\ \square,\square,\widehat{e}_1\, bop\, \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{ALL},[\widehat{mpcmpt}])} \\ ((1,\widehat{\gamma},\widehat{\sigma}_2,\square,\square,1) \qquad \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,1))\end{array}}$$

Multiparty Comparison False Operation

$$\frac{\begin{array}{c}((1,\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1)) \Downarrow'_{\widehat{\mathcal{D}}_1} ((1,\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1)) \\ ((1,\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{e}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_2} ((1,\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2)) \\ (\widehat{n}_1\, bop\, \widehat{n}_2) = 0 \qquad bop \in \{==,!=,<\}\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\ \square,\square,\widehat{e}_1\, bop\, \widehat{e}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\ \square,\square,\widehat{e}_1\, bop\, \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{ALL},[\widehat{mpcmpf}])} \\ ((1,\widehat{\gamma},\widehat{\sigma}_2,\square,\square,0) \qquad \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,0))\end{array}}$$

Multiparty If Else False

$$\frac{\begin{array}{c}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}_1} ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\hat{n}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\hat{n})) \qquad \hat{n} = 0 \\ ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{s}_1) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{s}_1)) \Downarrow'_{\widehat{\mathcal{D}}_2} ((1,\widehat{\gamma}_1,\widehat{\sigma}_2,\square,\square,\mathrm{skip}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma}_1,\widehat{\sigma}_2,\square,\square,\mathrm{skip})) \\ ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{s}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{s}_2)) \Downarrow'_{\widehat{\mathcal{D}}_3} ((1,\widehat{\gamma}_2,\widehat{\sigma}_3,\square,\square,\mathrm{skip}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma}_2,\widehat{\sigma}_3,\square,\square,\mathrm{skip}))\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\ \square,\square,\mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\ \square,\square,\mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::\widehat{\mathcal{D}}_3::(\mathrm{ALL},[\widehat{mpief}])} \\ ((1,\widehat{\gamma},\widehat{\sigma}_3,\square,\square,\mathrm{skip}) \qquad \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_3,\square,\square,\mathrm{skip}))\end{array}}$$

Multiparty If Else True

$$\frac{\begin{array}{c}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}_1} ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\hat{n}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\hat{n})) \qquad \hat{n} \neq 0 \\ ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{s}_1) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{s}_1)) \Downarrow'_{\widehat{\mathcal{D}}_2} ((1,\widehat{\gamma}_1,\widehat{\sigma}_2,\square,\square,\mathrm{skip}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma}_1,\widehat{\sigma}_2,\square,\square,\mathrm{skip})) \\ ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{s}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{s}_2)) \Downarrow'_{\widehat{\mathcal{D}}_3} ((1,\widehat{\gamma}_2,\widehat{\sigma}_3,\square,\square,\mathrm{skip}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma}_2,\widehat{\sigma}_3,\square,\square,\mathrm{skip}))\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\ \square,\square,\mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\ \square,\square,\mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::\widehat{\mathcal{D}}_3::(\mathrm{ALL},[\widehat{mpiet}])} \\ ((1,\widehat{\gamma},\widehat{\sigma}_2,\square,\square,\mathrm{skip}) \qquad \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,\mathrm{skip}))\end{array}}$$

Multiparty Pre-Increment Variable

$$\frac{\begin{array}{lll}\widehat{\gamma}(\widehat{x}) = (\widehat{l},\widehat{bty}) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\widehat{bty},1,\mathrm{PermL}(\mathrm{Freeable},\widehat{bty},\mathrm{public},1)) & \\ \mathrm{DecodeVal}(\widehat{bty},\widehat{\omega}) = \widehat{n}_1 & \widehat{n}_2 = \widehat{n}_1 + 1 & \mathrm{UpdateVal}(\widehat{\sigma},\widehat{l},\widehat{n}_2,\widehat{bty}) = \widehat{\sigma}_1\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\ \square,\square,++\ \widehat{x}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\ \square,\square,++\ \widehat{x})) \Downarrow'_{(\mathrm{ALL},[\widehat{mppin}])} \\ ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{n}_2) \quad \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{n}_2))\end{array}}$$

Figure 5.2: Multiparty Vanilla C semantics for multiparty evaluation of binary operations, if else statements, and pre-incrementing floats.

Multiparty Pointer Dereference Write Value

$$\frac{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}} ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{n}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{n})) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l},\widehat{bty*}) \qquad \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\widehat{bty*},1,\mathrm{PermL}(\mathrm{Freeable},\widehat{bty*},\mathrm{public},1)) \\ \mathrm{DecodePtr}(\widehat{bty*},1,\widehat{\omega}) = [1,[(\widehat{l}_1,\widehat{\mu}_1)],[1],1] \qquad \mathrm{UpdateOffset}(\widehat{\sigma}_1,(\widehat{l}_1,\widehat{\mu}_1),\widehat{n},\widehat{bty}) = (\widehat{\sigma}_2,1)\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,*\widehat{x}=\widehat{e}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,*\widehat{x}=\widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{ALL},[\widehat{mpwdp}])} \\ ((1,\widehat{\gamma},\widehat{\sigma}_2,\square,\square,\mathrm{skip}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,\mathrm{skip}))\end{array}}$$

Multiparty Pointer Dereference Write Value Higher Level Indirection

$$\frac{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}} ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,(\widehat{l}_e,\widehat{\mu}_e)) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,(\widehat{l}_e,\widehat{\mu}_e))) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l},\widehat{bty*}) \qquad \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\widehat{bty*},1,\mathrm{PermL}(\mathrm{Freeable},\widehat{bty*},\mathrm{public},1)) \\ \mathrm{DecodePtr}(\widehat{bty*},1,\widehat{\omega}) = [1,[(\widehat{l}_1,\widehat{\mu}_1)],[1],\widehat{i}] \\ \widehat{i} > 1 \qquad \mathrm{UpdatePtr}(\widehat{\sigma}_1,(\widehat{l}_1,\widehat{\mu}_1),[1,[(\widehat{l}_e,\widehat{\mu}_e)],[1],\widehat{i}-1],\widehat{bty*}) = (\widehat{\sigma}_2,1)\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,*\widehat{x}=\widehat{e}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,*\widehat{x}=\widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{ALL},[\widehat{mpwdp1}])} \\ ((1,\widehat{\gamma},\widehat{\sigma}_2,\square,\square,\mathrm{skip}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,\mathrm{skip}))\end{array}}$$

Multiparty Pointer Dereference

$$\frac{\begin{array}{l}\widehat{\gamma}(\widehat{x}) = (\widehat{l},\widehat{bty*}) \qquad \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\widehat{bty*},1,\mathrm{PermL}(\mathrm{Freeable},\widehat{bty*},\mathrm{public},1)) \\ \mathrm{DecodePtr}(\widehat{bty*},1,\widehat{\omega}) = [1,[(\widehat{l}_1,\widehat{\mu}_1)],[1],1] \qquad \mathrm{DerefPtr}(\widehat{\sigma},\widehat{bty},(\widehat{l}_1,\widehat{\mu}_1)) = (\widehat{n},1)\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,*\widehat{x}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,*\widehat{x})) \Downarrow'_{(\mathrm{ALL},[\widehat{mprdp}])} \\ ((1,\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{n}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{n}))\end{array}}$$

Multiparty Pointer Dereference Higher Level Indirection

$$\frac{\begin{array}{l}\widehat{\gamma}(\widehat{x}) = (\widehat{l},\widehat{bty*}) \qquad \widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\widehat{bty*},1,\mathrm{PermL}(\mathrm{Freeable},\widehat{bty*},\mathrm{public},1)) \qquad \widehat{i} > 1 \\ \mathrm{DecodePtr}(\widehat{bty*},1,\widehat{\omega}) = [1,[(\widehat{l}_1,\widehat{\mu}_1)],[1],\widehat{i}] \qquad \mathrm{DerefPtrHLI}(\widehat{\sigma},\widehat{bty*},(\widehat{l}_1,\widehat{\mu}_1)) = ([1,[(\widehat{l}_2,\widehat{\mu}_2)],[1],\widehat{i}-1],1)\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,*\widehat{x}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,*\widehat{x})) \Downarrow'_{(\mathrm{ALL},[\widehat{mprdp1}])} \\ ((1,\widehat{\gamma},\widehat{\sigma},\square,\square,(\widehat{l}_2,\widehat{\mu}_2)) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,(\widehat{l}_2,\widehat{\mu}_2)))\end{array}}$$

Multiparty Free

$$\frac{\begin{array}{l}\widehat{\gamma}(\widehat{x}) = (\widehat{l},\widehat{bty*}) \qquad \sigma(\widehat{l}) = (\widehat{\omega},\widehat{bty*},1,\mathrm{PermL}(\mathrm{Freeable},\widehat{bty*},\mathrm{public},1)) \\ \mathrm{DecodePtr}(\widehat{bty*},1,\widehat{\omega}) = [1,[(\widehat{l}_1,0)],[1],\widehat{i}] \qquad \mathrm{CheckFreeable}(\widehat{\gamma},[(\widehat{l}_1,0)],[1],\widehat{\sigma}) = 1 \qquad \mathrm{Free}(\widehat{\sigma},\widehat{l}_1) = \widehat{\sigma}_1\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,\mathrm{free}(\widehat{x})) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,\mathrm{free}(\widehat{x}))) \Downarrow'_{(\mathrm{ALL},[\widehat{mpfre}])} \\ ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\mathrm{skip}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\mathrm{skip}))\end{array}}$$

Multiparty Array Read

$$\frac{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}_1} ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{i}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{i})) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l},\mathrm{const}\ \widehat{bty*}) \qquad \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega},\mathrm{const}\ \widehat{bty*},1,\mathrm{PermL}(\mathrm{Freeable},\mathrm{const}\ \widehat{bty*},\mathrm{public},1)) \\ 0 \leq \widehat{i} \leq \widehat{\alpha}-1 \qquad \mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty*},1,\widehat{\omega}) = [1,[(\widehat{l}_1,0)],[1],1] \\ \widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1,\widehat{bty},\widehat{\alpha},\mathrm{PermL}(\mathrm{Freeable},\widehat{bty},\mathrm{public},\widehat{\alpha})) \qquad \mathrm{DecodeArr}(\widehat{bty},\ \widehat{i},\ \widehat{\omega}_1) = \widehat{n}_{\widehat{i}}\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{x}[\widehat{e}]) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{x}[\widehat{e}])) \Downarrow'_{\widehat{\mathcal{D}}_1::(\mathrm{ALL},[\widehat{mpra}])} \\ ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{n}_{\widehat{i}}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{n}_{\widehat{i}}))\end{array}}$$

Multiparty Array Write

$$\frac{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}_1) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}_1)) \Downarrow'_{\widehat{\mathcal{D}}_1} ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{i}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{i})) \\ ((1,\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{e}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_1,\square,\square,\widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_2} ((1,\widehat{\gamma},\widehat{\sigma}_2,\square,\square,\widehat{n}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,\widehat{n})) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l},\mathrm{const}\ \widehat{bty*}) \qquad \widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega},\mathrm{const}\ \widehat{bty*},1,\mathrm{PermL}(\mathrm{Freeable},\mathrm{const}\ \widehat{bty*},\mathrm{public},1)) \\ \mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty*},1,\widehat{\omega}) = [1,[(\widehat{l}_1,0)],[1],1] \qquad 0 \leq \widehat{i} \leq \widehat{\alpha}-1 \\ \widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1,\widehat{bty},\widehat{\alpha},\mathrm{PermL}(\mathrm{Freeable},\widehat{bty},\mathrm{public},\widehat{\alpha})) \qquad \mathrm{UpdateArr}(\widehat{\sigma}_2,\ (\widehat{l}_1,\widehat{i}),\ \widehat{n},\ \widehat{bty}) = \widehat{\sigma}_3\end{array}}{\begin{array}{l}((1,\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{x}[\widehat{e}_1]=\widehat{e}_2) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{x}[\widehat{e}_1]=\widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{ALL},[\widehat{mpwa}])} \\ ((1,\widehat{\gamma},\widehat{\sigma}_3,\square,\square,\mathrm{skip}) \parallel ... \parallel (\mathrm{q},\widehat{\gamma},\widehat{\sigma}_3,\square,\square,\mathrm{skip}))\end{array}}$$

Figure 5.3: Multiparty Vanilla C semantics for multiparty evaluation of pointers, deallocation, and arrays.

Equal To False

$$\frac{\begin{array}{l}((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel \widehat{C})\ \Downarrow'_{\widehat{\mathcal{D}}_1}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel \widehat{C}_1) \\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\square,\square,\ \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel \widehat{C}_2) \qquad (\widehat{n}_1 = \widehat{n}_2) = 0\end{array}}{((\mathrm{p},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}_1 == \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{eqf}])}\ ((\mathrm{p},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,0) \parallel \widehat{C}_2)}$$

Equal To True

$$\frac{\begin{array}{l}((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel \widehat{C})\ \Downarrow'_{\widehat{\mathcal{D}}_1}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel \widehat{C}_1) \\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\square,\square,\ \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel \widehat{C}_2) \qquad (\widehat{n}_1 = \widehat{n}_2) = 1\end{array}}{((\mathrm{p},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}_1 == \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{eqt}])}\ ((\mathrm{p},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,1) \parallel \widehat{C}_2)}$$

Not Equal To True

$$\frac{\begin{array}{l}((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel \widehat{C})\ \Downarrow'_{\widehat{\mathcal{D}}_1}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel \widehat{C}_1) \\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\square,\square,\ \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel \widehat{C}_2) \qquad (\widehat{n}_1 \neq \widehat{n}_2) = 1\end{array}}{((\mathrm{p},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}_1 != \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{net}])}\ ((\mathrm{p},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,\ 1) \parallel \widehat{C}_2)}$$

Not Equal To False

$$\frac{\begin{array}{l}((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel \widehat{C})\ \Downarrow'_{\widehat{\mathcal{D}}_1}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel \widehat{C}_1) \\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\square,\square,\ \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel \widehat{C}_2) \qquad (\widehat{n}_1 \neq \widehat{n}_2) = 0\end{array}}{((\mathrm{p},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}_1 != \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{nef}])}\ ((\mathrm{p},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,0) \parallel \widehat{C}_2)}$$

Less Than False

$$\frac{\begin{array}{l}((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel \widehat{C})\ \Downarrow'_{\widehat{\mathcal{D}}_1}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel \widehat{C}_1) \\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\square,\square,\ \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel \widehat{C}_2) \qquad (\widehat{n}_1 < \widehat{n}_2) = 0\end{array}}{((\mathrm{p},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}_1 < \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{ltf}])}\ ((\mathrm{p},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,0) \parallel \widehat{C}_2)}$$

Less Than True

$$\frac{\begin{array}{l}((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel \widehat{C})\ \Downarrow'_{\widehat{\mathcal{D}}_1}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel \widehat{C}_1) \\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\square,\square,\ \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel \widehat{C}_2) \qquad (\widehat{n}_1 < \widehat{n}_2) = 1\end{array}}{((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1 < \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{ltt}])}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ 1) \parallel \widehat{C}_2)}$$

Subtraction

$$\frac{\begin{array}{l}((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel \widehat{C})\ \Downarrow'_{\widehat{\mathcal{D}}_1}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel \widehat{C}_1) \\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\square,\square,\ \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel \widehat{C}_2) \qquad \widehat{n}_1 - \widehat{n}_2 = \widehat{n}_3\end{array}}{((\mathrm{p},\widehat{\gamma},\widehat{\sigma},\square,\square,\widehat{e}_1 - \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{bs}])}\ ((\mathrm{p},\widehat{\gamma},\widehat{\sigma}_2,\square,\square,\widehat{n}_3) \parallel \widehat{C}_2)}$$

Addition

$$\frac{\begin{array}{l}((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel \widehat{C})\ \Downarrow'_{\widehat{\mathcal{D}}_1}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel \widehat{C}_1) \\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\square,\square,\ \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel \widehat{C}_2) \qquad \widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3\end{array}}{((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1 + \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{bp}])}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_3) \parallel \widehat{C}_2)}$$

Multiplication

$$\frac{\begin{array}{l}((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel \widehat{C})\ \Downarrow'_{\widehat{\mathcal{D}}_1}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel \widehat{C}_1) \\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\square,\ \square,\widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel \widehat{C}_2) \qquad \widehat{n}_1 \cdot \widehat{n}_2 = \widehat{n}_3\end{array}}{((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1 \cdot \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{bm}])}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_3) \parallel \widehat{C}_2)}$$

Division

$$\frac{\begin{array}{l}((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1) \parallel \widehat{C})\ \Downarrow'_{\widehat{\mathcal{D}}_1}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\ \square,\square,\ \widehat{n}_1) \parallel \widehat{C}_1) \\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\square,\ \square,\widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_2) \parallel \widehat{C}_2) \qquad \widehat{n}_1 \div \widehat{n}_2 = \widehat{n}_3\end{array}}{((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \square,\square,\ \widehat{e}_1 \div \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{bd}])}\ ((\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \square,\square,\ \widehat{n}_3) \parallel \widehat{C}_2)}$$

Figure 5.4: Multiparty Vanilla C semantic rules for binary operations.

Declaration Assignment

$$\frac{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x}) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \ \|\ \widehat{C}_1) \\ ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \widehat{x} = \widehat{e}) \ \|\ \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \ \|\ \widehat{C}_2)\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x} = \widehat{e}) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{ds}])} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \ \|\ \widehat{C}_2)}$$

Address Of

$$\frac{\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{ty})}{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \&\widehat{x}) \ \|\ \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{loc}])} \\ ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}, 0)) \ \|\ \widehat{C})\end{array}}$$

Write

$$\frac{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \ \|\ \widehat{C}_1) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}) \qquad \mathrm{UpdateVal}(\widehat{\sigma}_1, \widehat{l}, \widehat{n}, \widehat{bty}) = \widehat{\sigma}_2\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x} = \widehat{e}) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::(\mathrm{p},[\widehat{w}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \ \|\ \widehat{C}_1)}$$

Size of type

$$\frac{\widehat{n} = \tau(\widehat{ty})}{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{sizeof}(\widehat{ty})) \ \|\ \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{ty}])} \\ ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}) \ \|\ \widehat{C})\end{array}}$$

Declaration

$$\frac{\begin{array}{ll}\widehat{l} = \phi() & \widehat{\omega} = \mathrm{EncodeVal}(\widehat{bty}, \mathrm{NULL}) \\ \widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \to (\widehat{l}, \widehat{bty})] & \widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \to (\widehat{\omega}, \widehat{bty}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, 1))]\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{bty}\ \widehat{x}) \ \|\ \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{dv}])} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \ \|\ \widehat{C})}$$

Statement Sequencing

$$\frac{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}_1) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \widehat{v}_1) \ \|\ \widehat{C}_1) \\ ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \widehat{s}_2) \ \|\ \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}_2, \widehat{\sigma}_2, \square, \square, \widehat{v}_2) \ \|\ \widehat{C}_2)\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}_1; \widehat{s}_2) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{ss}])} ((\mathrm{p}, \widehat{\gamma}_2, \widehat{\sigma}_2, \square, \square, \widehat{v}_2) \ \|\ \widehat{C}_2)}$$

Parentheses

$$\frac{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{v}) \ \|\ \widehat{C}_1)}{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{e})) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{ep}])} \\ ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{v}) \ \|\ \widehat{C}_1)\end{array}}$$

Statement Block

$$\frac{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \ \|\ \widehat{C}_1)}{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \{\widehat{s}\}) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{sb}])} \\ ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \ \|\ \widehat{C}_1)\end{array}}$$

Read

$$\frac{\begin{array}{l}\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}) \\ \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, 1)) \\ \mathrm{DecodeVal}(\widehat{bty}, \widehat{\omega}) = \widehat{n}\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \ \|\ \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{r}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}) \ \|\ \widehat{C})}$$

Pointer Read Location

$$\frac{\begin{array}{l}\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*) \\ \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}*, \mathrm{public}, 1)) \\ \mathrm{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \ \|\ \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{rp}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}_1, \widehat{\mu}_1)) \ \|\ \widehat{C})}$$

Pre-increment Variable

$$\frac{\begin{array}{lll}\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, 1)) & \\ \mathrm{DecodeVal}(\widehat{bty}, \widehat{\omega}) = \widehat{n}_1 & \widehat{n}_2 = \widehat{n}_1 + 1 & \mathrm{UpdateVal}(\widehat{\sigma}, \widehat{l}, \widehat{n}_2, \widehat{bty}) = \widehat{\sigma}_1\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, ++\ \widehat{x}) \ \|\ \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{pin}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}_2) \ \|\ \widehat{C})}$$

While End

$$\frac{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \ \|\ \widehat{C}_1) \qquad \widehat{n} = 0}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{while}(\widehat{e})\ \widehat{s}) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wle}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \ \|\ \widehat{C}_1)}$$

While Continue

$$\frac{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \ \|\ \widehat{C}_1) \qquad \widehat{n} \neq 0 \\ ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}) \ \|\ \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \ \|\ \widehat{C}_2)\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{while}(\widehat{e})\ \widehat{s}) \ \|\ \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{wlc}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{while}(\widehat{e})\ \widehat{s}) \ \|\ \widehat{C}_2)}$$

Figure 5.5: Additional Multiparty Vanilla C semantic rules within the scope of the grammar shown in Figure 3.1.

If Else False

$$\frac{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \hat{n}) \parallel \widehat{C}_1) \qquad \hat{n} = 0 \\ ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{ief}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)}$$

If Else True

$$\frac{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \hat{n}) \parallel \widehat{C}_1) \qquad \hat{n} \neq 0 \\ ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_1) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{iet}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)}$$

Function Call

$$\frac{\begin{array}{lll}\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{\overline{ty}} \to \widehat{ty}) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{\overline{ty}} \to \widehat{ty}, 1, \mathrm{PermL_Fun(public)}) & \mathrm{DecodeFun}(\widehat{\omega}) = (\widehat{s}, \square, \widehat{\overline{p}}) \\ \mathrm{GetFunParamAssign}(\widehat{\overline{p}}, \widehat{\overline{e}}) = \widehat{s}_1 & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}_1) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \parallel \widehat{C}_1) & \\ & ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \widehat{s}) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}_2, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2) &\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}(\widehat{\overline{e}})) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{fc}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)}$$

Pre-Declared Function Definition

$$\frac{\begin{array}{l}\widehat{x} \in \widehat{\gamma} \qquad \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{\overline{ty}} \to \widehat{ty}) \\ \qquad\qquad \mathrm{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}}) = \widehat{\omega} \\ \widehat{\sigma} = \widehat{\sigma}_1[\widehat{l} \to (\mathrm{NULL}, \widehat{\overline{ty}} \to \widehat{ty}, 1, \mathrm{PermL_Fun(public)})] \\ \widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \to (\widehat{\omega}, \widehat{\overline{ty}} \to \widehat{ty}, 1, \mathrm{PermL_Fun(public)})]\end{array}}{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p}, [\widehat{fpd}])} \\ ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \qquad \parallel \widehat{C})\end{array}}$$

Function Definition

$$\frac{\begin{array}{l}\widehat{x} \notin \widehat{\gamma} \qquad \mathrm{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}} \\ \widehat{l} = \phi() \qquad \widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \to (\widehat{l}, \widehat{\overline{ty}} \to \widehat{ty})] \\ \qquad\qquad \mathrm{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}}) = \widehat{\omega} \\ \widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \to (\widehat{\omega}, \widehat{\overline{ty}} \to \widehat{ty}, 1, \mathrm{PermL_Fun(public)})]\end{array}}{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p}, [\widehat{fd}])} \\ ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \qquad \parallel \widehat{C})\end{array}}$$

Function Declaration

$$\frac{\begin{array}{l}\widehat{l} = \phi() \qquad \mathrm{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}} \\ \qquad\qquad \widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \to (\widehat{l}, \widehat{\overline{ty}} \to \widehat{ty})] \\ \widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \to (\mathrm{NULL}, \widehat{\overline{ty}} \to \widehat{ty}, 1, \mathrm{PermL_Fun(public)})]\end{array}}{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p}, [\widehat{df}])} \\ ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \qquad \parallel \widehat{C})\end{array}}$$

Cast Value

$$\frac{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \hat{n}) \parallel \widehat{C}_1) \\ \hat{n}_1 = \mathrm{Cast}(\mathrm{public}, \widehat{ty}, \hat{n})\end{array}}{\begin{array}{l}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{ty})\ \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{cv}])} \\ ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \hat{n}_1) \qquad \parallel \widehat{C}_1)\end{array}}$$

Figure 5.6: Multiparty Vanilla C semantic rules for branching, functions, and casting values.

Input Value

$$\frac{\begin{array}{ll}\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}) & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \quad \| \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \quad \| \widehat{C}_1) \\ \mathrm{InputValue}(\widehat{x}, \widehat{n}) = \widehat{n}_1 & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{x} = \widehat{n}_1) \| \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \| \widehat{C}_2)\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{mcinput}(\widehat{x}, \widehat{e})) \| \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{inp}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \| \widehat{C}_2)}$$

Output Value

$$\frac{\begin{array}{lll}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \| \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \| \widehat{C}_1) & \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}) & \\ \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, 1)) & \mathrm{DecodeVal}(\widehat{bty}, \widehat{\omega}) = \widehat{n}_1 & \mathrm{OutputValue}(\widehat{x}, \widehat{n}, \widehat{n}_1)\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{mcoutput}(\widehat{x}, \widehat{e})) \| \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{out}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \| \widehat{C}_1)}$$

Input Array

$$\frac{\begin{array}{ll}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}_1) \| \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \| \widehat{C}_1) & \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \mathrm{const}\ \widehat{bty}*) \\ ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{e}_2) \| \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \widehat{\alpha}) \| \widehat{C}_2) & \mathrm{InputArray}(\widehat{x}, \widehat{n}, \widehat{\alpha}) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{\alpha}}] \\ ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \widehat{x} = [\widehat{m}_0, ..., \widehat{m}_{\widehat{\alpha}}]) \| \widehat{C}_2) \Downarrow'_{\widehat{\mathcal{D}}_3} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \| \widehat{C}_3) & \end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{mcinput}(\widehat{x}, \widehat{e}_1, \widehat{e}_2)) \| \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: \widehat{\mathcal{D}}_3 :: (\mathrm{p}, [\widehat{inp1}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \| \widehat{C}_3)}$$

Output Array

$$\frac{\begin{array}{ll}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}_1) \| \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \| \widehat{C}_1) & \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \mathrm{const}\ \widehat{bty}*) \\ ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{e}_2) \| \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \widehat{\alpha}) \| \widehat{C}_2) & \\ \widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty}*, \mathrm{public}, 1)) & \\ \mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1] & \widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, \widehat{\alpha})) \\ \forall i \in \{0, ..., \widehat{\alpha} - 1\} \quad \mathrm{DecodeArr}(\widehat{bty}, i, \widehat{\omega}_1) = \widehat{m}_i & \mathrm{OutputArray}(\widehat{x}, \widehat{n}, \widehat{\alpha}) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{\alpha}-1}]\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{mcoutput}(\widehat{x}, \widehat{e}_1, \widehat{e}_2)) \| \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{out1}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \| \widehat{C}_2)}$$

Free

$$\frac{\begin{array}{lll}\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}*, \mathrm{public}, 1)) & \\ \mathrm{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1] & \mathrm{CheckFreeable}(\widehat{\gamma}, [(\widehat{l}_1, 0)], [1], \widehat{\sigma}) = 1 & \mathrm{Free}(\widehat{\sigma}, \widehat{l}_1) = \widehat{\sigma}_1\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{free}(\widehat{x})) \| \widehat{C}) \Downarrow'_{(\mathrm{p}, [\widehat{fre}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \| \widehat{C})}$$

Malloc

$$\frac{\begin{array}{ll} & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \| \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \hat{n}) \| \widehat{C}_1) \\ \widehat{l} = \phi() & \widehat{\sigma}_2 = \widehat{\sigma}_1 \big[\widehat{l} \to \big(\mathrm{NULL}, \mathrm{void}*, \hat{n}, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{void}*, \mathrm{public}, \widehat{n})\big)\big]\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{malloc}(\widehat{e})) \| \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{mal}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, (\widehat{l}, 0)) \| \widehat{C}_1)}$$

Cast Location

$$\frac{\begin{array}{ll} (\widehat{ty} = \widehat{bty}*) & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \| \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}, 0)) \| \widehat{C}_1) \\ & \widehat{\sigma}_1 = \widehat{\sigma}_2 \big[\widehat{l} \to \big(\widehat{\omega}, \mathrm{void}*, \hat{n}, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{void}*, \mathrm{public}, \widehat{n})\big)\big] \\ & \widehat{\sigma}_3 = \widehat{\sigma}_2 \Big[\widehat{l} \to \Big(\widehat{\omega}, \widehat{ty}, \dfrac{\widehat{n}}{\tau(\widehat{ty})}, \mathrm{PermL}\Big(\mathrm{Freeable}, \widehat{ty}, \mathrm{public}, \dfrac{\widehat{n}}{\tau(\widehat{ty})}\Big)\Big)\Big]\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{ty})\ \widehat{e}) \| \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{cl}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, (\widehat{l}, 0)) \| \widehat{C}_1)}$$

Figure 5.7: Multiparty Vanilla C semantic rules for input and output, memory allocation and deallocation, and casting locations.

Pointer Declaration

$$\frac{\begin{array}{lll}(\widehat{ty} = \widehat{bty*}) & \text{GetIndirection}(*) = \widehat{i} & \widehat{l} = \phi() \quad \widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \to (\widehat{l}, \widehat{ty})] \\ \widehat{\omega} = \text{EncodePtr}(\widehat{ty*}, [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}]) & & \widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \to (\widehat{\omega}, \widehat{ty}, 0, \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, 0))]\end{array}}{((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\text{p}, [\widehat{dp}])} ((\text{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \text{skip}) \parallel \widehat{C})}$$

Pointer Write Location

$$\frac{\begin{array}{ll} & ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_e, \widehat{\mu}_e)) \parallel \widehat{C}_1) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*}) & \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \\ \text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}] & \text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}, 0), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}], \widehat{bty*}) = (\widehat{\sigma}_2, 1)\end{array}}{((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x} = \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{wp}])} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \text{skip}) \parallel \widehat{C}_1)}$$

Pre-Increment Pointer

$$\frac{\begin{array}{ll} \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*}) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \\ & \text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1] \\ ((\widehat{l}_2, \widehat{\mu}_2), 1) = \text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty}), \widehat{\sigma}) & \text{UpdatePtr}(\widehat{\sigma}, (\widehat{l}, 0), [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1], \widehat{bty*}) = (\widehat{\sigma}_1, 1)\end{array}}{((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, ++\ \widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\text{p}, [\widehat{pin1}])} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_2, \widehat{\mu}_2)) \parallel \widehat{C})}$$

Pre-Increment Pointer Higher Level Indirection

$$\frac{\begin{array}{ll} \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*}) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \\ \widehat{i} > 1 & \text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}] \\ ((\widehat{l}_2, \widehat{\mu}_2), 1) = \text{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty*}), \widehat{\sigma}) & \text{UpdatePtr}(\widehat{\sigma}, (\widehat{l}, 0), [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}], \widehat{bty*}) = (\widehat{\sigma}_1, 1)\end{array}}{((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, ++\ \widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\text{p}, [\widehat{pin2}])} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_2, \widehat{\mu}_2)) \parallel \widehat{C})}$$

Pointer Dereference Write Value

$$\frac{\begin{array}{ll} & ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*}) & \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \\ \text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1] & \text{UpdateOffset}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), \widehat{n}, \widehat{bty}) = (\widehat{\sigma}_2, 1)\end{array}}{((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{wdp}])} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \text{skip}) \parallel \widehat{C}_1)}$$

Pointer Dereference Write Higher Level Indirection

$$\frac{\begin{array}{ll} \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*}) & ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_e, \widehat{\mu}_e)) \parallel \widehat{C}_1) \\ \widehat{i} > 1 & \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \\ \text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}] & \text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1], \widehat{bty*}) = (\widehat{\sigma}_2, 1)\end{array}}{((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{wdp1}])} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \text{skip}) \parallel \widehat{C}_1)}$$

Pointer Dereference

$$\frac{\begin{array}{ll} \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*}) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \\ \text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1] & \text{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{n}, 1)\end{array}}{((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\text{p}, [\widehat{rdp}])} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}) \parallel \widehat{C})}$$

Pointer Dereference Higher Level Indirection

$$\frac{\begin{array}{lll} \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*}) & \widehat{i} > 1 & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1)) \\ \text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}] & & \text{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty*}, (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i} - 1], 1)\end{array}}{((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\text{p}, [\widehat{rdp1}])} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}_2, \widehat{\mu}_2)) \parallel \widehat{C})}$$

Figure 5.8: Additional Multiparty Vanilla C semantic rules for pointers.

Array Declaration Assignment

$$\frac{\begin{array}{c}((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x}[\widehat{e}]) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \parallel \widehat{C}_1) \\ ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \widehat{x} = \widehat{e}) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x}[\widehat{e}] = \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p}, [\widehat{das}])} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)}$$

Read Entire Array

$$\frac{\begin{array}{ll}\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \mathrm{const}\ \widehat{bty}*) & \widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty}*, \mathrm{public}, 1)) \\ & \mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1] \\ \widehat{\sigma}(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, bty, \mathrm{public}, \widehat{\alpha})) & \forall \widehat{i} \in \{0...\widehat{\alpha} - 1\}.\quad \mathrm{DecodeArr}(\widehat{bty}, \widehat{i}, \widehat{\omega}_1) = \widehat{n}_{\widehat{i}}\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p}, [\widehat{rea}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, [\widehat{n}_0, ..., \widehat{n}_{\widehat{\alpha}-1}]) \parallel \widehat{C})}$$

Write Entire Array

$$\frac{\begin{array}{ll} & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, [\widehat{n}_0, ..., \widehat{n}_{\widehat{\alpha}_e - 1}]) \parallel \widehat{C}_1) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \mathrm{const}\ \widehat{bty}*) & \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty}*, \mathrm{public}, 1)) \\ \mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1] & \widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, bty, \mathrm{public}, \widehat{\alpha})) \\ \widehat{\alpha}_e = \widehat{\alpha} & \forall \widehat{i} \in \{0...\widehat{\alpha} - 1\}\quad \mathrm{UpdateArr}(\widehat{\sigma}_{1+\widehat{i}},\ (\widehat{l}_1, \widehat{i}),\ \widehat{n}_{\widehat{i}},\ \widehat{bty}) = \sigma_{2+\widehat{i}}\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x} = \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p}, [\widehat{wea}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_{2+\widehat{\alpha}-1}, \square, \square, \mathrm{skip}) \parallel \widehat{C}_1)}$$

Array Declaration

$$\frac{\begin{array}{ll}\widehat{l} = \phi() \qquad \widehat{l}_1 = \phi() & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{\alpha}) \parallel \widehat{C}_1) \\ \widehat{\alpha} > 0 & \widehat{\omega} = \mathrm{EncodePtr}(\mathrm{const}\ \widehat{bty}*, [1, [(\widehat{l}_1, 0)], [1], 1]) \\ \widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \to (\widehat{l}, \mathrm{const}\ \widehat{bty}*)] & \widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \to (\widehat{\omega}, \mathrm{const}\ \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty}*, \mathrm{public}, 1))] \\ \widehat{\omega}_1 = \mathrm{EncodeArr}(\widehat{bty}, \widehat{\alpha}, \mathrm{NULL}) & \widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l}_1 \to (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, \widehat{\alpha}))]\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{bty}\ \widehat{x}[\widehat{e}]) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::(\mathrm{p}, [\widehat{da}])} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \parallel \widehat{C}_1)}$$

Array Read

$$\frac{\begin{array}{ll} & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{i}) \parallel \widehat{C}_1) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \mathrm{const}\ \widehat{bty}*) & \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty}*, \mathrm{public}, 1)) \\ & \mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1] \\ & \widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, \widehat{\alpha})) \\ 0 \leq \widehat{i} \leq \widehat{\alpha} - 1 & \mathrm{DecodeArr}(\widehat{bty},\ \widehat{i},\ \widehat{\omega}_1) = \widehat{n}_{\widehat{i}}\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}]) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::(\mathrm{p}, [\widehat{ra}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}_{\widehat{i}}) \parallel \widehat{C}_1)}$$

Array Write

$$\frac{\begin{array}{ll} & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma},\ \square, \square, \widehat{e}_1) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{i}) \parallel \widehat{C}_1) \\ & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \widehat{n}) \parallel \widehat{C}_2) \\ \widehat{\gamma}(\widehat{x}) = (\widehat{l}, \mathrm{const}\ \widehat{bty}*) & \widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty}*, \mathrm{public}, 1)) \\ & \mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1] \\ & \widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, \widehat{\alpha})) \\ 0 \leq \hat{i} \leq \widehat{\alpha} - 1 & \mathrm{UpdateArr}(\widehat{\sigma}_2,\ (\widehat{l}_1, \widehat{i}),\ \widehat{n},\ \widehat{bty}) = \widehat{\sigma}_3\end{array}}{((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p}, [\widehat{wa}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)}$$

Figure 5.9: Multiparty Vanilla C semantic rules for arrays.

Array Read Out of Bounds

$$
\begin{array}{ll}
 & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{i}) \parallel \widehat{C}_1) \\
\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \mathrm{const}\ \widehat{bty}*) & \widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty}*, \mathrm{public}, 1)) \\
 & \mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1] \\
 & \widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, \widehat{\alpha})) \\
(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{\alpha}) & \mathrm{ReadOOB}(\widehat{i}, \widehat{\alpha}, \widehat{l}_1, \widehat{bty}, \widehat{\sigma}_1) = (\widehat{n}, 1)
\end{array}
$$

$$
((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}]) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{rao}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)
$$

Array Write Out of Bounds

$$
\begin{array}{ll}
 & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \ \square, \square, \widehat{e}_1) \parallel \widehat{C}) \ \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \hat{i}) \parallel \widehat{C}_1) \\
 & ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \widehat{n}) \parallel \widehat{C}_2) \\
\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \mathrm{const}\ \widehat{bty}*) & \widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty}*, \mathrm{public}, 1)) \\
 & \mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1] \\
 & \widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, \widehat{\alpha})) \\
(\hat{i} < 0) \vee (\hat{i} \geq \widehat{\alpha}) & \mathrm{WriteOOB}(\widehat{n}, \hat{i}, \widehat{\alpha}, \widehat{l}_1, \widehat{bty}, \widehat{\sigma}_2) = (\widehat{\sigma}_3, 1)
\end{array}
$$

$$
((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{wao}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)
$$

Figure 5.10: Multiparty Vanilla C semantic rules for array out of bounds.

5.1.2 Multiparty SMC^2

In this section, we show the Multiparty SMC^2 semantics with respect to the grammar (Figure 3.1). The Multiparty SMC^2 semantics are defined over multiple *interacting* parties, with the semantic judgements defined over a four-tuple configuration $C = (\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s)$ where each party has its own configuration and each rule is a reduction from a set of party-wise configurations $\widehat{C}^1 \parallel ... \parallel \widehat{C}^{\mathrm{q}}$, and each rule is a reduction from one configuration to a subsequent. We denote the party's identifier as p; the environment as γ; memory as σ; the location map for tracking changes within private-conditioned branches as Δ; the level of nesting of private-conditioned branches as acc; and a big-step evaluation of a statement s to a value v using $\Downarrow$. We annotate each evaluation with party-wise lists of the evaluation codes $\mathcal{D}$ of all rules that were used during the execution of the rule (i.e., $\Downarrow^{\mathcal{L}}_{\mathcal{D}}$) in order to keep an accurate evaluation tree, and party-wise lists of locations accessed $\mathcal{L}$ in order to show data-obliviousness (i.e., that given the same program and public data, we will always access the same set of locations).

The Multiparty SMC^2 semantics used to define the behavior of parties are mostly standard, with *non-interactive* semantic rules identical to those of Vanilla C semantics aside from additional assertions over the privacy labels of data and properly managing the private data. A few notable exceptions are *interactive* SMC operations (and in general operations over private values) and the private-conditioned **`if else`** statement, discussed in later in this section. To prevent leakage from within private-conditioned branches, we restrict all public side effects (i.e., the use of functions with public side effects, allocation and deallocation of memory, and any modifications to public variables). Additionally, in the case of pointer dereference write and array write statements, we have an additional check for when this occurs within a private-conditioned branch, as we need to perform additional analysis to ensure the location being written to is tracked properly due to the potential for the pointer's location being modified or an out-of-bounds array write. To enforce these restrictions, we use the assertion $\mathrm{acc} = 0$ within each restricted rule – as the accumulator acc is incremented at each level of nesting of a private-conditioned branch, this will result in a runtime failure. We do not show the semantic rules that are not *well-aligned* in this chapter, as they are nearly identical to their corresponding rules and the proof of noninterference over these rules are handled similarly to the cases of the corresponding rules. Such rules are shown in the Basic SMC^2 semantics for the interested reader. The assertions in each semantic rule can be read in sequential order, from left to right and top to bottom.

It is worthwhile to note here, before discussing the semantic rules, that the number of locations that a

pointer will refer to and the level of indirection of a pointer is based on the program itself, and therefore must be the same across all parties. Proving that the level of indirection is consistent across all parties is done by induction over all rules, showing that it is assigned when a pointer is declared and never changed in any other rules. Proving that the number of locations a pointer will refer to can be done by evaluating any of the following: (1) Private If Else rules change the number of locations based on the statements from both branches, or (2) Private Free changes the number of locations based on how many locations the pointer that is being freed had, or (3) Private Pointer Write and Dereference Write assign a new number of locations to a pointer based on the pointer that is being read from. All other rules do not modify the number of locations that a pointer refers to.

Figure 5.11 gives the semantic rules for binary operations involving private data. Figure 5.12 gives the semantic rules for reading from or writing to a private index of an array. Figure 5.13 gives the semantic rules for private free with multiple locations and the pre-increment operation over private float values. Figure 5.14 gives the semantic rules for dereference writing multiple location to a private pointer of a higher level of indirection. Figure 5.15 gives the semantic rules for private pointer dereference read with multiple locations and dereference write with a public value. Figure 5.16 gives the semantic rules for the multiparty execution of private-conditioned if else statements. Figure 5.17 gives the semantic rules for public branches and loops. Figure 5.18 gives the semantic rules for pointer declarations and writing to a pointer. Figure 5.19 gives the semantic rules for reading and dereferencing a pointer. Figure 5.20 gives the semantic rules for dereference writing to a pointer.

Figure 5.21 gives the semantic rules for array declarations and reading from a public index of an array. Figure 5.22 gives the semantic rules for writing to a public index of an array. Figure 5.23 gives the semantic rules for reading from and writing to an entire array. Figure 5.24 gives the semantic rules for reading and writing out of bounds for arrays. Figure 5.25 gives the semantic rules for pre-incrementing private int variables and for private pointers. Figure 5.26 gives the semantic rules for pre-incrementing public variables. Figure 5.27 gives the semantic rules for memory allocation and deallocation, casting, and finding the address of a variable. Figure 5.28 gives the semantic rules for functions and finding the size of a type. Figure 5.29 gives the semantic rules for public binary operations. Figure 5.30 gives the semantic rules for declarations, reading, and writing for regular (non-pointer, non-array) variables, as well as general sequencing rules. Figures 5.31 and 5.32 give the semantic rules for inputting and outputting data, respectively.

First, we will consider the *interactive* semantic rules (i.e., those that require communication between

Multiparty Binary Operation

$$\frac{\begin{array}{l}\{(e_1, e_2) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}} \qquad\qquad bop \in \{\cdot, +, -, \div\} \\ ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1)) \Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n_1^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n_1^{\mathrm{q}})) \\ ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, e_2)) \Downarrow_{\mathcal{D}_2}^{\mathcal{L}_2} ((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n_2^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n_2^{\mathrm{q}})) \\ \mathrm{MPC}_b(bop, [n_1^1, ..., n_1^{\mathrm{q}}], [n_2^1, ..., n_2^{\mathrm{q}}]) = (n_3^1, ..., n_3^{\mathrm{q}})\end{array}}{\begin{array}{l}((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1 \ bop \ e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1 \ bop \ e_2)) \Downarrow_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpb])}^{\mathcal{L}_1::\mathcal{L}_2} \\ ((1, \gamma_2^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n_3^1) \qquad \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n_3^{\mathrm{q}}))\end{array}}$$

Multiparty Comparison Operation

$$\frac{\begin{array}{l}\{(e_1, e_2) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}} \qquad\qquad bop \in \{==, !=, <\} \\ ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1)) \Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((1, \gamma_1^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n_1^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n_1^{\mathrm{q}})) \\ ((1, \gamma_1^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, e_2)) \Downarrow_{\mathcal{D}_2}^{\mathcal{L}_2} ((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n_2^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n_2^{\mathrm{q}})) \\ \mathrm{MPC}_{cmp}(bop, [n_1^1, ..., n_1^{\mathrm{q}}], [n_2^1, ..., n_2^{\mathrm{q}}]) = (n_3^1, ..., n_3^{\mathrm{q}})\end{array}}{\begin{array}{l}((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1 \ bop \ e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1 \ bop \ e_2)) \Downarrow_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpcmp])}^{\mathcal{L}_1::\mathcal{L}_2} \\ ((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n_3^1) \qquad \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n_3^{\mathrm{q}}))\end{array}}$$

Figure 5.11: Multiparty SMC^2 semantics for reading from or writing to a private index of an array and binary operations involving private data.

parties). To better illustrate the correspondence between Multiparty SMC^2 and Multiparty Vanilla C, let us first consider the Multiparty Binary Operation rule. Multiparty SMC^2 rule Multiparty Binary Operation asserts that one of the given binary operators $(\cdot, +, -, \div)$ is used and additionally that either expression contains private data with relation to the environment. We use the notation $(e_1, e_2) \vdash \gamma$ to show this relation, and notation $\{...\}_{\mathrm{p}=1}^{\mathrm{q}}$ to show that all parties will ensure that property holds locally. We then use the multiparty protocol MPC_b, passing the given binary operator and the current values of n_1^{p} and n_2^{p} for each party p. This protocol will dictate how communication occurs and what data is exchanged between parties. We receive n_3^{p} as the result for each party, which we then return appropriately. We assume that the protocol is implemented correctly (i.e. provided by the underlying SMC cryptographic library) and define this assumption formally, its impact on our noninterferences proof, and how to reason if a library adheres to our assumption later in Section 5.3.

Within the multiparty rules, each party maintains control of their own data, only sharing it with other parties in the ways dictated by the multiparty protocols. We choose to show the execution of the entire computational process here in order to emphasize what data is involved, and that each of the parties will take part in this computation. The Multiparty Comparison Operation rule is nearly identical to the Multiparty Binary Operation rule, only differing in which binary operations are accepted by the rule. We separate these two rules because, when proving correctness of the rules, comparison operations and other binary operations have differing behavior. Other binary operations will compute over the two values, returning the result of that computation, and there is only one possible outcome for the rule. With comparison operations, we have two

possible outcomes: either the comparison holds true, and we return 1, or the comparison is false, and we return 0. In Multiparty Vanilla C, we have two rules: one for each possible result of the comparison (i.e., true or false), and to prove correctness of Multiparty Comparison Operation, we must prove both subcases are also correct.

Multiparty Array Read Private Index

$$\frac{\begin{array}{l}\{(e) \vdash \gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1} \qquad \{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \text{const } a\ bty*)\}^{\mathrm{q}}_{\mathrm{p}=1} \\ ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_l} ((1, \gamma^1, \sigma^1_1, \Delta^1_1, \mathrm{acc}, i^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_1, \Delta^{\mathrm{q}}_1, \mathrm{acc}, i^{\mathrm{q}})) \\ \{\sigma^{\mathrm{p}}_1(l^{\mathrm{p}}) = (\omega^{\mathrm{p}},\ a \text{ const } bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, a \text{ const } bty*, a, 1))\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\mathrm{DecodePtr}(a \text{ const } bty*,\ 1,\ \omega^{\mathrm{p}}) = [1,\ [(l^{\mathrm{p}}_1, 0)],\ [1],\ 1]\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\sigma^{\mathrm{p}}_1(l^{\mathrm{p}}_1) = (\omega^{\mathrm{p}}_1, a\ bty, \alpha, \mathrm{PermL}(\mathrm{Freeable}, a\ bty, a, \alpha))\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\forall j \in \{0...\alpha-1\} \quad \mathrm{DecodeArr}(a\ bty, j, \omega^{\mathrm{p}}_1) = n^{\mathrm{p}}_j\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \mathrm{MPC}_{ar}((i^1, [n^1_0, ..., n^1_{\alpha-1}]), ..., (i^{\mathrm{q}}, [n^{\mathrm{q}}_0, ..., n^{\mathrm{q}}_{\alpha-1}])) = (n^1, ..., n^{\mathrm{q}}) \qquad \{(n^{\mathrm{p}}) \vdash \gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \mathcal{L}_2 = (1, [(l^1, 0), (l^1_1, 0), ..., (l^1_1, \alpha-1)]) \parallel ... \parallel (\mathrm{q}, [(l^{\mathrm{q}}, 0), (l^{\mathrm{q}}_1, 0), ..., (l^{\mathrm{q}}_1, \alpha-1)])\end{array}}{\begin{array}{l}((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, x[e]) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, x[e])) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_l::(\mathrm{ALL}, [mpra])} \\ ((1, \gamma^1, \sigma^1_1, \Delta^1_1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_1, \Delta^{\mathrm{q}}_1, \mathrm{acc}, n^{\mathrm{q}}))\end{array}}$$

Multiparty Array Write Private Index

$$\frac{\begin{array}{l}\{(e_1) \vdash \gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1} \qquad \{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \text{private const } bty*)\}^{\mathrm{q}}_{\mathrm{p}=1} \\ ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_l} ((1, \gamma^1, \sigma^1_1, \Delta^1_1, \mathrm{acc}, i^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_1, \Delta^{\mathrm{q}}_1, \mathrm{acc}, i^{\mathrm{q}})) \\ ((1, \gamma^1, \sigma^1_1, \Delta^1_1, \mathrm{acc}, e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_1, \Delta^{\mathrm{q}}_1 \mathrm{acc}, e_2)) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((1, \gamma^1, \sigma^1_2, \Delta^1_2, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_2, \Delta^{\mathrm{q}}_2, \mathrm{acc}, n^{\mathrm{q}})) \\ \{\sigma^{\mathrm{p}}_2(l^{\mathrm{p}}) = (\omega^{\mathrm{p}},\ \text{private const } bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, \text{private const } bty*, \text{private}, 1))\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\mathrm{DecodePtr}(\text{private const } bty*,\ 1,\ \omega^{\mathrm{p}}) = [1,\ [(l^{\mathrm{p}}_1, 0)],\ [1],\ 1]\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\sigma^{\mathrm{p}}_2(l^{\mathrm{p}}_1) = (\omega^{\mathrm{p}}_1, \text{private } bty, \alpha, \mathrm{PermL}(\mathrm{Freeable}, \text{private } bty, \text{private}, \alpha))\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\forall j \in \{0...\alpha-1\} \quad \mathrm{DecodeArr}(\text{private } bty, j, \omega^{\mathrm{p}}_1) = n^{\mathrm{p}}_j\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \mathrm{MPC}_{aw}((i^1, n^1, [n^1_0, ..., n^1_{\alpha-1}]), ..., (i^{\mathrm{q}}, n^{\mathrm{q}}, [n^{\mathrm{q}}_0, ..., n^{\mathrm{q}}_{\alpha-1}])) = ([n'^1_0, ..., n'^1_{\alpha-1}], ..., [n'^{\mathrm{q}}_0, ..., n'^{\mathrm{q}}_{\alpha-1}]) \\ \{\forall j \in \{0...\alpha-1\} \quad \mathrm{UpdateArr}(\sigma^{\mathrm{p}}_{2+j},\ (l^{\mathrm{p}}_1, j),\ n'^{\mathrm{p}}_j,\ \text{private } bty) = \sigma^{\mathrm{p}}_{3+j}\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \mathcal{L}_3 = (1, [(l^1, 0), (l^1_1, 0), ..., (l^1_1, \alpha-1)]) \parallel ... \parallel (\mathrm{q}, [(l^{\mathrm{q}}, 0), (l^{\mathrm{q}}_1, 0), ..., (l^{\mathrm{q}}_1, \alpha-1)])\end{array}}{\begin{array}{l}((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, x[e_1] = e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, x[e_1] = e_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3}_{\mathcal{D}_l::\mathcal{D}_2::(\mathrm{ALL}, [mpwa])} \\ ((1, \gamma^1, \sigma^1_{3+\alpha-1}, \Delta^1_2, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_{3+\alpha-1}, \Delta^{\mathrm{q}}_2, \mathrm{acc}, \mathrm{skip}))\end{array}}$$

Figure 5.12: Multiparty SMC^2 semantics for reading from or writing to a private index of an array and binary operations involving private data.

In rule Multiparty Array Read Private Index, we are handling the case where we are reading from private index in a public or private array; because we have a private index, we must obtain the value without revealing which location we are taking the value from. We first assert that the expression must contain private information in order to be a private index, and that this must hold for every party. Then, each party will look up the variable to find that it is an array type, with the same basic type and privacy label for all parties. All parties will then evaluate the expression to obtain their version of the private index (e.g., their share of the private value when using a Shamir Secret Sharing implementation). Each party then proceeds to look up the array pointer and then the array data in memory, decoding the byte representation for all indices to obtain the entire array. We then use multiparty protocol MPC_{ar}, discussed later in Algorithm 108, to privately obtain the value at the private index. We make the assertion that the values returned from multiparty protocol

MPC_{ar} must be private, as they were based on a private index.

It is important to note here that even if the private index is beyond the bounds of the array, we do not access beyond the elements within the array, as that would reveal information about the private index. An example of how this protocol can be implemented is to iterate over all values stored in the array; at each value, we encrypt the current index number m, privately compare it to i, and perform a bitwise `and` operation over this and the encrypted value n_m stored at index m. We perform a bitwise `or` operation over each such value obtained from the array to attain our final encrypted value n, which is returned. The final assertion is over what locations were accessed within this rule, to assist us in reasoning about data obliviousness within our proof of noninterference. In this rule, for every party we have accessed the location of the array pointer (i.e., location $(l^{\mathrm{p}}, 0)$) by looking it up in memory σ_1^{p} and then decoding what was stored there. We have also accessed the location of each of the elements within the array (i.e., every element of the array data), by looking the array data block up in memory and then proceeding to decode the value found at every index within that block. We pass along the party-wise location lists obtained from the evaluation of the expression and the evaluation of the rule concatenated together in the order that they were accessed.

The Multiparty SMC^2 rule Multiparty Array Write Private Index is quite similar to In rule Multiparty Array Read Private Index. We must also evaluate the second expression to find what we are storing at the private index, which can be public or private. After we have looked up the array and obtained all values stored within the array, we use multiparty protocol MPC_{aw}, discussed later in Algorithm 109, to privately obtain the new array data with the new value stored at the private index. Each party then places their new array data back into memory for the array. Our final assertion is over which locations have been accessed within this rule, which is, again, very similar to the previous rule. We will access every index within the array block twice within this rule, as we are first reading it from memory and decoding it and then updating it in memory at the end. It is important to note here that for both of these rules, we cannot go out-of-bounds of the array data, as that would leak information about what the private index was.

When deallocating private memory, we provide the `pfree` builtin function to handle private pointers potentially having multiple locations. In the case of a single location, we use rule Private Free Single Location, and it behaves identically to rule Public Free; however, with multiple locations, we need to deterministically free a single location (which may or may not be the true location that was intended to be freed) to maintain data-obliviousness. We describe this case in more detail here. In rule Private Free Multiple Locations, we assert that x is a private pointer of type int or float, we are not inside a private-conditioned branch (acc is 0, as

Private Free Multiple Locations

$$\frac{\begin{array}{l}\{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}},\ \text{private } bty*)\}_{\mathrm{p}=1}^{\mathrm{q}} \qquad \mathrm{acc} = 0 \qquad (bty = \mathrm{int}) \vee (bty = \mathrm{float}) \\ \{\sigma^{\mathrm{p}}(l^{\mathrm{p}}) = (\omega^{\mathrm{p}}, \text{private } bty*, \alpha, \mathrm{PermL}(\mathrm{Freeable}, \text{private } bty*, \mathrm{private}, \alpha))\}_{\mathrm{p}=1}^{\mathrm{q}} \qquad \{\alpha > 1\}_{\mathrm{p}=1}^{\mathrm{q}} \\ \{[\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}},\ i] = \mathrm{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}} \\ \mathrm{if}(i > 1)\{ty = \text{private } bty*\} \text{ else } \{ty = \text{private } bty\} \\ \{\mathrm{CheckFreeable}(\gamma^{\mathrm{p}}, \bar{l}^{\mathrm{p}}, \bar{j}^{\mathrm{p}}, \sigma^{\mathrm{p}}) = 1\}_{\mathrm{p}=1}^{\mathrm{q}} \\ \{\forall (l_m^{\mathrm{p}}, 0) \in \bar{l}^{\mathrm{p}}.\quad \sigma^{\mathrm{p}}(l_m^{\mathrm{p}}) = (\omega_m^{\mathrm{p}}, ty, \alpha_m, \mathrm{PermL}(\mathrm{Freeable}, ty, \mathrm{private}, \alpha_m))\}_{\mathrm{p}=1}^{\mathrm{q}} \\ \mathrm{MPC}_{free}([[\omega_0^1, ..., \omega_{\alpha-1}^1], ..., [\omega_0^{\mathrm{q}}, ..., \omega_{\alpha-1}^{\mathrm{q}}]], [\bar{j}^1, ...\bar{j}^{\mathrm{q}}]) = ([[\omega_0'^1, ..., \omega_{\alpha-1}'^1], ..., [\omega_0'^{\mathrm{q}}, ..., \omega_{\alpha-1}'^{\mathrm{q}}]], [\bar{j}'^1, ..., \bar{j}'^{\mathrm{q}}]) \\ \{\mathrm{UpdateBytesFree}(\sigma^{\mathrm{p}}, \bar{l}^{\mathrm{p}}, [\omega_0'^{\mathrm{p}}, ..., \omega_{\alpha-1}'^{\mathrm{p}}]) = \sigma_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}} \\ \{(\sigma_2^{\mathrm{p}}, \bar{l}_1^{\mathrm{p}}) = \mathrm{UpdatePointerLocations}(\sigma_1^{\mathrm{p}}, \bar{l}^{\mathrm{p}}[1:\alpha-1],\ \bar{j}^{\mathrm{p}}[1:\alpha-1], \bar{l}^{\mathrm{p}}[0], \bar{j}^{\mathrm{p}}[0])\}_{\mathrm{p}=1}^{\mathrm{q}}\end{array}}{\begin{array}{l}((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, \mathrm{pfree}(x)) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, \mathrm{pfree}(x))) \Downarrow_{(\mathrm{ALL}, [mpfre])}^{(1, [(l^1, 0)]::\bar{l}^1::\bar{l}_1^1) \parallel ... \parallel (\mathrm{q}, [(l^{\mathrm{q}}, 0)]::\bar{l}^{\mathrm{q}}::\bar{l}_1^{\mathrm{q}})} \\ ((1, \gamma^1, \sigma_2^1, \Delta^1, \mathrm{acc}, \mathrm{skip}) \quad \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip}))\end{array}}$$

Multiparty Pre-Increment Private Float Variable

$$\frac{\begin{array}{ll}\{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \text{private float})\}_{\mathrm{p}=1}^{\mathrm{q}} & \\ \{\sigma^{\mathrm{p}}(l^{\mathrm{p}}) = (\omega^{\mathrm{p}}, \text{private float}, 1, \mathrm{PermL}(\mathrm{Freeable}, \text{private float}, \mathrm{private}, 1))\}_{\mathrm{p}=1}^{\mathrm{q}} & \\ \{(x) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}} & \{\mathrm{DecodeVal}(\text{private float}, \omega^{\mathrm{p}}) = n_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}} \\ \mathrm{MPC}_u(++, n_1^1, ..., n_1^{\mathrm{q}}) = (n_2^1, ..., n_2^{\mathrm{q}}) & \{\mathrm{UpdateVal}(\sigma^{\mathrm{p}}, l^{\mathrm{p}}, n_2^{\mathrm{p}}, \text{private float}) = \sigma_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}\end{array}}{\begin{array}{l}((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, ++\ x) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, ++\ x)) \Downarrow_{(\mathrm{ALL}, [mppin])}^{(1, [(l^1, 0)]) \parallel ... \parallel (\mathrm{q}, [(l^{\mathrm{q}}, 0)])} \\ ((1, \gamma^1, \sigma_1^1, \Delta^1, \mathrm{acc}, n_2^1) \quad \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, n_2^{\mathrm{q}}))\end{array}}$$

Figure 5.13: Multiparty SMC^2 semantics for private free with multiple locations and the pre-increment operation over private float values.

this rule causes public side effects), and that the number of locations the pointer refers to (α) is greater than 1 for all parties. We then assert that *all* locations referred to by x are freeable (i.e., they are all memory blocks that were allocated via malloc) and proceed to retrieve the data that is stored for each of these locations. This data and the tag lists are then passed to MPC_{free}, as this is what we will need in order to privately free a location without revealing if it was the true location.

To accomplish this, we must free one location based on publicly available information, regardless of the true location of the pointer. For that reason, and without loss of generality, we free the first location, l_0. Since l_0 may not be the true location and may be in use by other pointers, we need to do additional computation to maintain correctness without disclosing whether or not this was the true location. In particular, if l_0 is not the true location, we preserve the content of l_0 by obliviously copying it to the pointer's true location prior to freeing. This behavior is defined in function MPC_{free}, and follows the strategy suggested in [22]. MPC_{free} returns the modified bytes and tag lists. $\mathrm{UpdateBytesFree}$ then updates these in their corresponding locations in memory and marks the permissions of l_0 as None (i.e., this block has been freed). The remaining step is to update other pointers that stored l_0 on their lists to point to the updated location instead of l_0, which is accomplished by $\mathrm{UpdatePointerLocations}$.

Multiparty Pre-Increment Private Float Variable handles the pre-increment operation over private float

variables. Due to the way private float data is stored and distributed, it is necessary for all float operations, including the pre-increment operation, to be performed in the *interactive* setting, whereas pre-incrementing a private integer can be handled locally. This rule, however, is fairly straightforward – we first must look up the value stored for the float variable at each party, then we use the multiparty protocol to perform the increment operation, and finally update the incremented value in memory and return it.

Multiparty Private Pointer Dereference Write Value Higher Level Indirection

$$\frac{\begin{array}{l}
\{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \text{private } bty*)\}_{\mathrm{p}=1}^{\mathrm{q}} \\
\quad ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc},\ e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \text{acc}, e)) \\
\Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((1, \gamma^1,\ \sigma_1^1,\ \Delta_1^1,\ \text{acc},\ (l_e^1, \mu_e^1)) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \text{acc}, (l_e^{\mathrm{q}}, \mu_e^{\mathrm{q}}))) \\
\{\sigma_1^{\mathrm{p}}(l^{\mathrm{p}}) = (\omega^{\mathrm{p}},\ \text{private } bty*,\ \alpha,\ \mathrm{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))\}_{\mathrm{p}=1}^{\mathrm{q}} \qquad \alpha > 1 \\
\{\mathrm{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega^{\mathrm{p}}) = [\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}}, i]\}_{\mathrm{p}=1}^{\mathrm{q}} \qquad i > 1 \\
\{\mathrm{DynamicUpdate}(\Delta_1^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, \bar{l}^{\mathrm{p}},\ \text{acc}, \text{private } bty*) = (\Delta_2^{\mathrm{p}}, \bar{l}_1^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}} \\
\{\mathrm{Retrieve_Values}(\alpha,\ \bar{l}^{\mathrm{p}},\ \text{private } bty*,\ \sigma_1^{\mathrm{p}}) = ([[\alpha_0,\ \bar{l}_0^{\mathrm{p}},\ \bar{j}_0^{\mathrm{p}}, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^{\mathrm{p}},\ \bar{j}_{\alpha-1}^{\mathrm{p}}, i-1]], 1)\}_{\mathrm{p}=1}^{\mathrm{q}} \\
\mathrm{MPC}_{wdp}([[[1, [(l_e^1, \mu_e^1)], [1], i-1], [\alpha_0,\ \bar{l}_0^1,\ \bar{j}_0^1, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^1,\ \bar{j}_{\alpha-1}^1, i-1]], ..., \\
\qquad [[1, [(l_e^{\mathrm{q}}, \mu_e^{\mathrm{q}})], [1], i-1], [\alpha_0,\ \bar{l}_0^{\mathrm{q}},\ \bar{j}_0^{\mathrm{q}}, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^{\mathrm{q}},\ \bar{j}_{\alpha-1}^{\mathrm{q}}, i-1]]], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) \\
\quad = [[[\alpha'_0,\ \bar{l}_0'^1,\ \bar{j}_0'^1, i-1], ..., [\alpha'_{\alpha-1},\ \bar{l}_{\alpha-1}'^1,\ \bar{j}_{\alpha-1}'^1, i-1]], ..., [[\alpha'_0,\ \bar{l}_0'^{\mathrm{q}},\ \bar{j}_0'^{\mathrm{q}}, i-1], ..., [\alpha'_{\alpha-1},\ \bar{l}_{\alpha-1}'^{\mathrm{q}},\ \bar{j}_{\alpha-1}'^{\mathrm{q}}, i-1]]] \\
\{\mathrm{UpdateDerefVals}(\alpha, \bar{l}^{\mathrm{p}}, [[\alpha'_0, \bar{l}_0'^{\mathrm{p}}, \bar{j}_0'^{\mathrm{p}}, i-1], ..., [\alpha'_{\alpha-1}, \bar{l}_{\alpha-1}'^{\mathrm{p}}, \bar{j}_{\alpha-1}'^{\mathrm{p}}, i-1]], \text{private } bty*, \sigma_1^{\mathrm{p}}) = \sigma_2^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}
\end{array}}{\begin{array}{l}
((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc},\ *x = e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \text{acc}, *x = e)) \ \Downarrow_{\mathcal{D}_1::(\mathrm{ALL}, [mpwdp2])}^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}_1^1::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^{\mathrm{q}},0)::\bar{l}_1^{\mathrm{q}}::\bar{l}^{\mathrm{q}})} \\
((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \text{acc},\ \text{skip}) \quad \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \text{acc}, \text{skip}))
\end{array}}$$

Multiparty Private Pointer Dereference Write Multiple Locations Higher Level Indirection

$$\frac{\begin{array}{l}
\{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \text{private } bty*)\}_{\mathrm{p}=1}^{\mathrm{q}} \\
\quad ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc},\ e) \parallel .. \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \text{acc}, e)) \\
\Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((1, \gamma^1,\ \sigma_1^1,\ \Delta_1^1,\ \text{acc},\ [\alpha_e, \bar{l}_e^1, \bar{j}_e^1, i-1]) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \text{acc}, [\alpha_e, \bar{l}_e^{\mathrm{q}}, \bar{j}_e^{\mathrm{q}}, i-1])) \qquad \alpha_e > 1 \\
\{\sigma_1^{\mathrm{p}}(l^{\mathrm{p}}) = (\omega^{\mathrm{p}},\ \text{private } bty*,\ \alpha,\ \mathrm{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))\}_{\mathrm{p}=1}^{\mathrm{q}} \qquad \alpha > 1 \\
\{\mathrm{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega^{\mathrm{p}}) = [\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}}, i]\}_{\mathrm{p}=1}^{\mathrm{q}} \qquad i > 1 \\
\{\mathrm{DynamicUpdate}(\Delta_1^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, \bar{l}^{\mathrm{p}},\ \text{acc}, \text{private } bty*) = (\Delta_2^{\mathrm{p}}, \bar{l}_1^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}} \\
\{\mathrm{Retrieve_Values}(\alpha,\ \bar{l}^{\mathrm{p}},\ \text{private } bty*,\ \sigma_1^{\mathrm{p}}) = ([[\alpha_0,\ \bar{l}_0^{\mathrm{p}},\ \bar{j}_0^{\mathrm{p}}, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^{\mathrm{p}},\ \bar{j}_{\alpha-1}^{\mathrm{p}}, i-1]], 1)\}_{\mathrm{p}=1}^{\mathrm{q}} \\
\mathrm{MPC}_{wdp}([[[\alpha_e, \bar{l}_e^1, \bar{j}_e^1, i-1], [\alpha_0,\ \bar{l}_0^1,\ \bar{j}_0^1, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^1,\ \bar{j}_{\alpha-1}^1, i-1]], ..., \\
\qquad [[\alpha_e, \bar{l}_e^{\mathrm{q}}, \bar{j}_e^{\mathrm{q}}, i-1], [\alpha_0,\ \bar{l}_0^{\mathrm{q}},\ \bar{j}_0^{\mathrm{q}}, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^{\mathrm{q}},\ \bar{j}_{\alpha-1}^{\mathrm{q}}, i-1]]], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) \\
\quad = [[[\alpha'_0,\ \bar{l}_0'^1,\ \bar{j}_0'^1, i-1], ..., [\alpha'_{\alpha-1},\ \bar{l}_{\alpha-1}'^1,\ \bar{j}_{\alpha-1}'^1, i-1]], ..., [[\alpha'_0,\ \bar{l}_0'^{\mathrm{q}},\ \bar{j}_0'^{\mathrm{q}}, i-1], ..., [\alpha'_{\alpha-1},\ \bar{l}_{\alpha-1}'^{\mathrm{q}},\ \bar{j}_{\alpha-1}'^{\mathrm{q}}, i-1]]] \\
\{\mathrm{UpdateDerefVals}(\alpha, \bar{l}^{\mathrm{p}}, [[\alpha'_0, \bar{l}_0'^{\mathrm{p}}, \bar{j}_0'^{\mathrm{p}}, i-1], ..., [\alpha'_{\alpha-1}, \bar{l}_{\alpha-1}'^{\mathrm{p}}, \bar{j}_{\alpha-1}'^{\mathrm{p}}, i-1]], \text{private } bty*, \sigma_1^{\mathrm{p}}) = \sigma_2^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}
\end{array}}{\begin{array}{l}
((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc},\ *x = e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \text{acc}, *x = e)) \ \Downarrow_{\mathcal{D}_1::(\mathrm{ALL}, [mpwdp1])}^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}_1^1::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^{\mathrm{q}},0)::\bar{l}_1^{\mathrm{q}}::\bar{l}^{\mathrm{q}})} \\
((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \text{acc},\ \text{skip}) \quad \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \text{acc}, \text{skip}))
\end{array}}$$

Figure 5.14: Multiparty SMC^2 semantic rules for dereference writing multiple location to a private pointer of a higher level of indirection.

The difference between rules Multiparty Private Pointer Dereference Write Value Higher Level Indirection and Multiparty Private Pointer Dereference Write Multiple Locations Higher Level Indirection is the number of locations that we are assigning to the dereferenced pointer (i.e., a single location and multiple locations, respectively). In both rules, we assert that, for every party, the pointer is private, and then all parties evaluate the expression to obtain the location(s) to assign to the dereferenced pointer. Each party can then look up the pointer in memory. Given that we are assigning locations to the dereferenced pointer, we also must assert that

this pointer has a level of indirection greater than 1.

We add in a call to Algorithm 120 here, which will ensure that the location we are modifying is properly tracked when we are inside a private-conditioned branch. We then retrieve all of the pointer data structures stored at each of the possible locations for this pointer we are dereference writing to (i.e., the top-level pointer). These pointer data structures, along with the location (or pointer data structure) we are assigning to the location and the tag list from the top-level pointer, are passed to the multiparty protocol MPC_{wdp}, discussed in Algorithm 113. This protocol will handle (separately) merging the location lists and tag lists for each of the locations with the location(s) obtained from the evaluation of the expression, and return a new pointer data structure to be written into memory for each of the possible locations of the top-level pointer. Additionally, we pass along all locations that were accessed within the evaluation of the expression and the evaluation of this rule when we return from this rule, as shown by the annotation on the evaluation arrow $\Downarrow$.

The public **`if else`** rules, shown in appendix Figure 5.17, are nearly identical to the Multiparty Vanilla C rules, with the added assertion that the guard of the conditional is public (i.e., does not contain private data): $(e) \nvdash \gamma$. The private **`if else`** rules, shown in Figure 5.16, are more interesting. Our strategy for dealing with private-conditioned branches involves executing both branches as a sequence of statements (with some additional helper algorithms to aid in storing changes, restoration between branches, and resolution of true values). We chose to use big-step semantics to facilitate the comparison of the Multiparty SMC^2 semantics with the Multiparty Vanilla C semantics, and for its proof of correctness that we will discuss in the next Section. We use coloring throughout Figure 5.16 to highlight the corresponding sections of rule execution.

The starting and ending states of the Multiparty SMC^2 Private If Else rules are essentially the same as the starting and ending states of the corresponding Multiparty Vanilla C If Else rule; however, there are several additional assertions that guarantee that both of the private-conditioned branches are executed. The assertions of these semantic rules are listed sequentially, from top to bottom. We have two different styles of tracking modifications within conditional code blocks that are used within these rules: variable tracking and location tracking. Variable tracking is used when there are only single-level changes within the private-conditioned branches, whereas location tracking is used when we have multi-level changes (i.e., a branch contains a pointer dereference write) or potential out-of-bounds changes (i.e., array write at a public index).

The main idea of both styles is to first store the original value of each variable that is modified within either branch; execute the **`then`** branch; save the resulting values from the **`then`** branch and restore all modified variables to their original values; execute the **`else`** branch; and finally, to securely resolve which values

should be kept – those from the **`then`** branch or those from the **`else`** branch. In the variable style of tracking, we utilize temporary variables to keep track of all modifications made during either branch – initializing the **`else`** temporary with the original value, storing the result of the **`then`** branch in the **`then`** temporary and using the **`else`** temporary to restore the original value, and finally using the result of the private-conditional and what is stored in each variable at the end of the **`else`** branch as well as it's corresponding **`then`** temporary to securely resolve what values to continue evaluating the program with.

This style of tracking is robust enough for many uses, however, there are two notable exceptions where we run into issues, both involving the potential of the location we track not being the location that is actually modified. The first exception involves pointer dereference writes – these alone are not an issue, but when location the pointer refers to is modified and we also perform pointer dereference writes, it becomes clear that variable tracking cannot easily find and handle these cases. The second exception involves array writes at public indices – these become problematic due to the potential for writing out-of-bounds. As most array indices are not hard-coded, it isn't obvious that the write will be within bounds until execution, and to ensure we catch all of these cases we must use a more robust style of tracking to catch out-of-bounds writes. We stress here that array writes at private indices do not fall within this exception, as this operation will securely update the array within its bounds (as updating beyond the bounds of the array would leak that this private value is larger than the size of the array), and as such we can simply track the entire array properly using variable tracking. It is possible to ensure that we find all of the locations that are modified in both of these cases by dynamically adding these types of modifications as they are evaluated, which is the goal of the location tracking. In the location style of tracking, we still follow a similar evaluation pattern as with variable tracking, storing the original values for locations we know will be modified first, then restoring between branches, and resolving at the end. As we evaluate each branch and come upon one of these special cases, we will check to see if we have already marked that location for tracking, and if not we add that location and its original value before the modification occurs. It is worthwhile to stress again the role of the accumulator here with respect to other statements. We increment it when we evaluate the **`then`** and **`else`** statements, so that if we attempt to evaluate a (sub)statement with public side effects or restricted operations, we have an (oblivious) runtime failure. It also facilitates scoping of temporary variables within nested private-conditioned **`if else`** statements. We proceed to further describe the different assertions and specifics of both styles next.

Conditional Code Block Variable Tracking For this style of tracking, we first evaluate expression e over environment γ, memory σ and accumulator acc to obtain some number n; the same environment, and a potentially updated memory (e.g. in the case $e = x++$). We then extract the non-local variables that are modified within either branch, and check whether multi-level modifications or array writes at a public index occur. This is achieved with Algorithm $\mathrm{Extract}$ by iterating through both statement s_1 and s_2 and storing the variable names in list $list_{\mathrm{acc}+1}$, as well as updating and returning a tag to indicate whether we have found multi-level modifications (0 for false, 1 for true). Next we call Algorithm $\mathrm{InitializeVariables}$, which stores n as the value of a temporary variable $res_{\mathrm{acc}+1}$, using $\mathrm{acc} + 1$ to denote the current level of nesting in the upcoming **`then`** and **`else`** statements. The variable $res_{\mathrm{acc}+1}$ is later used in the resolution phase, to select the result according to the branching condition. It then iterates through the list of variables, creating two temporary versions of each variable, named x_then_acc and x_else_acc, and storing each in memory with the initial value of what x has in the memory σ_1. Next is the evaluation of the **`then`** statement, and afterwards we must restore the original memory. To do this, we call $\mathrm{RestoreVariables}$, which iterates through each of the variables x contained within $list_{\mathrm{acc}+1}$, storing their current value into their **`then`** temporary (i.e., $x_then_{\mathrm{acc}+1} = x$) and restoring their original value from their **`else`** temporary (i.e., $x = x_else_{\mathrm{acc}+1}$). Once we have completed this, the evaluation of the **`else`** statement can occur.

Finally, we need to perform the resolution of all changes made to variables in either branch. To enable this, we call Algorithm $\mathrm{ResolveVariables_Retrieve}$ to iterate through each of the variables within $list_{\mathrm{acc}+1}$ and grab their values accordingly, as well as retrieving the result of the private condition (whose value we stored in $res_{\mathrm{acc}+1}$). We then use multiparty protocol $\mathrm{MPC}_{resolve}$ to facilitate the resolution of the true values, as these computations require communication between parties. For variables that are not array or pointer variables, we perform a series of binary operations over the byte values of the private variables (e.g., `c=(res·c_t)+((1-res)·c_e)`). The process is similar for arrays, with some addition bookkeeping due to their structure as a const pointer referring to the location with the array data. For pointers, we must handle the different locations referred to from each branch, merging the two location lists and finding what the true location is. The resolved values are then returned, and Algorithm $\mathrm{ResolveVariables_Store}$ stores all each back into memory for its respective variable. Notice that, in the conclusion, we revert to the original environment γ. In this way, all the temporary variables we used become out of scope and are discarded - in particular, this prevents reusing the same temporary variable mapping if we have multiple (not nested) private if else statements.

Conditional Code Block Location Tracking Here we track modifications during private-conditioned branches at the level of memory blocks and offsets, which ensures that we do not update any data in memory inaccurately, as is shown in Figure 4.1a using variable tracking SMC techniques. To facilitate this, we use the mapping structure Δ to track changes to each location at each level of nesting. This structure maps locations to a four-tuple of the original value, the **`then`** branch value, a tag to notate whether the **`then`** branch value was updated during the restoration phase, and the type of value stored (i.e., $(l, \mu) \rightarrow (v_1, v_2, j, ty)$). The tag is used to allow us to add to Δ as we encounter pointer dereference writes and array writes at public indices without needing to track which branch we are in. It is always initialized as 0, and updated to 1 when we enter the restoration phase and store a value into the **`then`** position. This way, if a location was added in the **`else`** branch (i.e., was not modified in the **`then`** branch), we know to use the original value as the **`then`** value when we resolve the true value of that location at the end.

The overall structure of the location tracking rule is similar to the variable tracking rule. We first evaluate e to n, then call $\mathrm{Extract}$ to find variables that are modified during the execution of either branch and that there are multi-level modifications within at least one branch. We then call $\mathrm{Initialize}$, which stores the result of the private conditional and uses the variables we found to create the initial mapping Δ. Next, we proceed to evaluate the **`then`** branch, and call $\mathrm{Restore}$ to update Δ with the ending **`then`** values for all locations that are tracked and restore the original values back into memory. After, we evaluate the **`else`** branch and, once complete, call $\mathrm{Resolve_Retrieve}$ to retrieve the result of the conditional and the **`then`** and **`else`** values for each location. As with variable tracking, we use multiparty protocol $\mathrm{MPC}_{resolve}$ to obtain the true values, and then store them back into their respective locations using Algorithm $\mathrm{Resolve_Store}$. It is important to note that when we evaluate a pointer dereference write or array write at a public index inside a branch, we check to see if the given location is in $\Delta[\mathrm{acc}]$. If it is not, we add a mapping to store the original data (i.e., $(l, \mu) \rightarrow (\mathtt{orig}, \mathrm{NULL}, 0, ty)$). Notice that the data can either be a regular value (i.e., for a memory block storing a private int) or a pointer data structure representing a private pointer (i.e., for a memory block storing a private int*).

The remaining semantics rules are all *non-interactive*. These are mostly standard, and do not have major differences from prior versions. In rules for pointer dereference write and writing to an array at a public index, we insert calls to Algorithm 120 in order to ensure that the locations modified by these rules are properly tracked within private-conditioned branches. When this algorithm is called from outside a private-conditioned branch, it will return without having modified location map Δ.

Reading from a private pointer that has multiple locations and assigning multiple locations to a pointer are local operations. This is because we are simply reading from or writing to memory - we do not need to know the true location for the pointer in these operations. All dereference operations over private pointers with single locations are executed locally, as we easily read and write at the publicly known location that the private pointer refers to. These operations have multiparty counterparts for when the private pointers refer to multiple locations, as when we execute those versions we must have communication between parties to privately evaluate what location's data we are truly reading from or writing to.

Multiparty Private Pointer Dereference Single Level Indirection

$$\frac{\begin{array}{l}\{(x) \vdash \gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1} \qquad \{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \text{private } bty*)\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\sigma^{\mathrm{p}}(l^{\mathrm{p}}) = (\omega^{\mathrm{p}},\ \text{private } bty*,\ \alpha,\ \mathrm{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))\}^{\mathrm{q}}_{\mathrm{p}=1} \qquad \alpha > 1 \\ \{\mathrm{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega^{\mathrm{p}}) = [\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}}, 1]\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\mathrm{Retrieve_Values}(\alpha, \bar{l}^{\mathrm{p}}, \text{private } bty, \sigma^{\mathrm{p}}) = ([n^{\mathrm{p}}_0, ...n^{\mathrm{p}}_{\alpha-1}], 1)\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \mathrm{MPC}_{dv}([[n^1_0, ..., n^1_{\alpha-1}], ..., [n^{\mathrm{q}}_0, ..., n^{\mathrm{q}}_{\alpha-1}]], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) = (n^1, ..., n^{\mathrm{q}})\end{array}}{\begin{array}{l}((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, *x)) \Downarrow^{(1,(l^1,0)::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^{\mathrm{q}},0)::\bar{l}^{\mathrm{q}})}_{(\mathrm{ALL},[mprdp])} \\ ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}}))\end{array}}$$

Multiparty Private Pointer Dereference Higher Level Indirection

$$\frac{\begin{array}{l}\{(x) \vdash \gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1} \qquad \{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \text{private } bty*)\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\sigma^{\mathrm{p}}(l^{\mathrm{p}}) = (\omega^{\mathrm{p}},\ \text{private } bty*,\ \alpha,\ \mathrm{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))\}^{\mathrm{q}}_{\mathrm{p}=1} \qquad \alpha > 1 \\ \{\mathrm{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega^{\mathrm{p}}) = [\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}}, i]\}^{\mathrm{q}}_{\mathrm{p}=1} \qquad i > 1 \\ \{\mathrm{Retrieve_Values}(\alpha,\ \bar{l}^{\mathrm{p}},\ \text{private } bty*,\ \sigma^{\mathrm{p}}) = ([[\alpha_0,\ \bar{l}^{\mathrm{p}}_0,\ \bar{j}^{\mathrm{p}}_0, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}^{\mathrm{p}}_{\alpha-1},\ \bar{j}^{\mathrm{p}}_{\alpha-1}, i-1]], 1)\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \mathrm{MPC}_{dp}([[[\alpha_0,\ \bar{l}^1_0,\ \bar{j}^1_0], ..., [\alpha_{\alpha-1},\ \bar{l}^1_{\alpha-1},\ \bar{j}^1_{\alpha-1}]], ..., [[\alpha_0,\ \bar{l}^{\mathrm{q}}_0,\ \bar{j}^{\mathrm{q}}_0], ..., [\alpha_{\alpha-1},\ \bar{l}^{\mathrm{q}}_{\alpha-1},\ \bar{j}^{\mathrm{q}}_{\alpha-1}]]], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) \\ \qquad = ([[\alpha_\alpha,\ \bar{l}^1_\alpha,\ \bar{j}^1_\alpha], ..., [\alpha_\alpha,\ \bar{l}^{\mathrm{q}}_\alpha,\ \bar{j}^{\mathrm{q}}_\alpha]])\end{array}}{\begin{array}{l}((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, *x)) \Downarrow^{(1,(l^1,0)::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^{\mathrm{q}},0)::\bar{l}^{\mathrm{q}})}_{(\mathrm{ALL},[mprdp1])} \\ ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, [\alpha_\alpha, \bar{l}^1_\alpha, \bar{j}^1_\alpha, i-1]) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, [\alpha_\alpha, \bar{l}^{\mathrm{q}}_\alpha, \bar{j}^{\mathrm{q}}_\alpha, i-1]))\end{array}}$$

Multiparty Private Pointer Dereference Write Private Value

$$\frac{\begin{array}{l}\{(e) \vdash \gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1} \qquad \{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \text{private } bty*)\}^{\mathrm{q}}_{\mathrm{p}=1} \\ ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_I} ((1, \gamma^1, \sigma^1_1, \Delta^1_1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_1, \Delta^{\mathrm{q}}_1, \mathrm{acc}, n^{\mathrm{q}})) \\ \{\sigma^{\mathrm{p}}_1(l^{\mathrm{p}}) = (\omega^{\mathrm{p}},\ \text{private } bty*,\ \alpha,\ \mathrm{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))\}^{\mathrm{q}}_{\mathrm{p}=1} \qquad \alpha > 1 \\ \{\mathrm{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega^{\mathrm{p}}) = [\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}}, 1]\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\mathrm{DynamicUpdate}(\Delta^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_1, \bar{l}^{\mathrm{p}},\ \mathrm{acc}, \text{private } bty) = (\Delta^{\mathrm{p}}_2, \bar{l}^{\mathrm{p}}_1)\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\mathrm{Retrieve_Values}(\alpha, \bar{l}^{\mathrm{p}}, \text{private } bty, \sigma^{\mathrm{p}}_1) = ([n^{\mathrm{p}}_0, ...n^{\mathrm{p}}_{\alpha-1}], 1)\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \mathrm{MPC}_{wdv}([[n^1_0, ..., n^1_{\alpha-1}], ..., [n^{\mathrm{q}}_0, ..., n^{\mathrm{q}}_{\alpha-1}]], [n^1, ..., n^{\mathrm{q}}], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) = ([n'^1_0, ..., n'^1_{\alpha-1}], ..., [n'^{\mathrm{q}}_0, ..., n'^{\mathrm{q}}_{\alpha-1}]) \\ \{\mathrm{UpdateDerefVals}(\alpha, \bar{l}^{\mathrm{p}}, [n'^{\mathrm{p}}_0, ..., n'^{\mathrm{p}}_{\alpha-1}], \text{private } bty, \sigma^{\mathrm{p}}_1) = \sigma^{\mathrm{p}}_2\}^{\mathrm{q}}_{\mathrm{p}=1}\end{array}}{\begin{array}{l}((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \mathrm{acc},\ *x = e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}},\ \sigma^{\mathrm{q}},\ \Delta^{\mathrm{q}},\ \mathrm{acc},\ *x = e)) \Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}^1_1::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^{\mathrm{q}},0)::\bar{l}^{\mathrm{q}}_1::\bar{l}^{\mathrm{q}})}_{\mathcal{D}_I::(\mathrm{ALL},[mpwdp3])} \\ ((1, \gamma^1,\ \sigma^1_2,\ \Delta^1_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}},\ \sigma^{\mathrm{q}}_2,\ \Delta^{\mathrm{q}}_2,\ \mathrm{acc},\ \mathrm{skip}))\end{array}}$$

Multiparty Private Pointer Dereference Write Public Value

$$\frac{\begin{array}{l}\{(e) \nvdash \gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1} \qquad \{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \text{private } bty*)\}^{\mathrm{q}}_{\mathrm{p}=1} \\ ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_I} ((1, \gamma^1, \sigma^1_1, \Delta^1_1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_1, \Delta^{\mathrm{q}}_1, \mathrm{acc}, n^{\mathrm{q}})) \\ \{\sigma^{\mathrm{p}}_1(l^{\mathrm{p}}) = (\omega^{\mathrm{p}},\ \text{private } bty*,\ \alpha,\ \mathrm{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))\}^{\mathrm{q}}_{\mathrm{p}=1} \qquad \alpha > 1 \\ \{\mathrm{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega^{\mathrm{p}}) = [\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}}, 1]\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\mathrm{DynamicUpdate}(\Delta^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_1, \bar{l}^{\mathrm{p}},\ \mathrm{acc}, \text{private } bty) = (\Delta^{\mathrm{p}}_2, \bar{l}^{\mathrm{p}}_1)\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \{\mathrm{Retrieve_Values}(\alpha, \bar{l}^{\mathrm{p}}, \text{private } bty, \sigma^{\mathrm{p}}_1) = ([n^{\mathrm{p}}_0, ...n^{\mathrm{p}}_{\alpha-1}], 1)\}^{\mathrm{q}}_{\mathrm{p}=1} \\ \mathrm{MPC}_{wdv}([[n^1_0, ..., n^1_{\alpha-1}], ..., [n^{\mathrm{q}}_0, ..., n^{\mathrm{q}}_{\alpha-1}]], [\mathrm{encrypt}(n^1), ..., \mathrm{encrypt}(n^{\mathrm{q}})], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) \\ \qquad = ([n'^1_0, ..., n'^1_{\alpha-1}], ..., [n'^{\mathrm{q}}_0, ..., n'^{\mathrm{q}}_{\alpha-1}]) \\ \{\mathrm{UpdateDerefVals}(\alpha, \bar{l}^{\mathrm{p}}, [n'^{\mathrm{p}}_0, ..., n'^{\mathrm{p}}_{\alpha-1}], \text{private } bty, \sigma^{\mathrm{p}}_1) = \sigma^{\mathrm{p}}_2\}^{\mathrm{q}}_{\mathrm{p}=1}\end{array}}{\begin{array}{l}((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \mathrm{acc}, *x = e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, *x = e)) \Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}^1_1::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^{\mathrm{q}},0)::\bar{l}^{\mathrm{q}}_1::\bar{l}^{\mathrm{q}})}_{\mathcal{D}_I::(\mathrm{ALL},[mpwdp])} \\ ((1, \gamma^1,\ \sigma^1_2,\ \Delta^1_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_2, \Delta^{\mathrm{q}}_2, \mathrm{acc}, \mathrm{skip}))\end{array}}$$

Figure 5.15: Multiparty SMC^2 semantic rules for private pointer dereference read with multiple locations and dereference write with a public value.

Private If Else (Variable Tracking)

$$
\begin{array}{l}
\quad ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, e) \;\; \| \ldots \| \; (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, e)) \\
\Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} \; ((1, \gamma^1, \sigma^1_1, \Delta^1_1, \text{acc}, n^1) \| \ldots \| \; (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}_1, \Delta^{\text{q}}_1, \text{acc}, n^{\text{q}})) \qquad \{(e) \vdash \gamma^{\text{p}}\}^{\text{q}}_{\text{p}=1} \\
\{\text{Extract}(s_1, s_2, \gamma^{\text{p}}) = (\overline{x}, 0)\}^{\text{q}}_{\text{p}=1} \\
\{\text{InitializeVariables}(\overline{x}, \; \gamma^{\text{p}}, \sigma^{\text{p}}_1, n^{\text{p}}, \; \text{acc}+1) = (\gamma^{\text{p}}_1, \sigma^{\text{p}}_2, \bar{l}^{\text{p}}_2)\}^{\text{q}}_{\text{p}=1} \\
\quad ((1, \gamma^1_1, \sigma^1_2, \Delta^1_1, \text{acc}+1, s_1) \;\; \| \ldots \| \; (\text{q}, \gamma^{\text{q}}_1, \sigma^{\text{q}}_2, \Delta^{\text{q}}_1, \text{acc}+1, s_1)) \\
\Downarrow^{\mathcal{L}_3}_{\mathcal{D}_2} \; ((1, \gamma^1_2, \sigma^1_3, \Delta^1_2, \text{acc}+1, \text{skip}) \| \ldots \| \; (\text{q}, \gamma^{\text{q}}_2, \sigma^{\text{q}}_3, \Delta^{\text{q}}_2, \text{acc}+1, \text{skip})) \\
\{\text{RestoreVariables}(\overline{x}, \; \gamma^{\text{p}}_1, \; \sigma^{\text{p}}_3, \; \text{acc}+1) = (\sigma^{\text{p}}_4, \bar{l}^{\text{p}}_4)\}^{\text{q}}_{\text{p}=1} \\
\quad ((1, \gamma^1_1, \sigma^1_4, \Delta^1_2, \text{acc}+1, s_2) \;\; \| \ldots \| \; (\text{q}, \gamma^{\text{q}}_1, \sigma^{\text{q}}_4, \Delta^{\text{q}}_2, \text{acc}+1, s_2)) \\
\Downarrow^{\mathcal{L}_5}_{\mathcal{D}_3} \; ((1, \gamma^1_3, \sigma^1_5, \Delta^1_3, \text{acc}+1, \text{skip}) \| \ldots \| \; (\text{q}, \gamma^{\text{q}}_3, \sigma^{\text{q}}_5, \Delta^{\text{q}}_3, \text{acc}+1, \text{skip})) \\
\{\text{ResolveVariables_Retrieve}(\overline{x}, \; \text{acc}+1, \gamma^{\text{p}}_1, \sigma^{\text{p}}_5) = ([(v^{\text{p}}_{t1}, v^{\text{p}}_{e1}), \ldots, (v^{\text{p}}_{tm}, v^{\text{p}}_{em})], n^{\text{p}}, \bar{l}^{\text{p}}_6)\}^{\text{q}}_{\text{p}=1} \\
\text{MPC}_{resolve}([n^1, \ldots, n^{\text{q}}], [[(v^1_{t1}, v^1_{e1}), \ldots, (v^1_{tm}, v^1_{em})], \ldots, [(v^{\text{q}}_{t1}, v^{\text{q}}_{e1}), \ldots, (v^{\text{q}}_{tm}, v^{\text{q}}_{em})]]) = [[v^1_1, \ldots, v^1_m], \ldots, [v^{\text{q}}_1, \ldots, v^{\text{q}}_m]] \\
\{\text{ResolveVariables_Store}(\overline{x}, \; \text{acc}+1, \gamma^{\text{p}}_1, \sigma^{\text{p}}_5, [v^{\text{p}}_1, \ldots, v^{\text{p}}_m]) = (\sigma^{\text{p}}_6, \bar{l}^{\text{p}}_7)\}^{\text{q}}_{\text{p}=1} \\
\mathcal{L}_2 = (1, \bar{l}^1_2) \| \ldots \| \; (\text{q}, \bar{l}^{\text{q}}_2) \qquad \mathcal{L}_4 = (1, \bar{l}^1_4) \| \ldots \| \; (\text{q}, \bar{l}^{\text{q}}_4) \\
\mathcal{L}_6 = (1, \bar{l}^1_6) \| \ldots \| \; (\text{q}, \bar{l}^{\text{q}}_6) \qquad \mathcal{L}_7 = (1, \bar{l}^1_7) \| \ldots \| \; (\text{q}, \bar{l}^{\text{q}}_7) \\
\hline
\quad ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, \text{if } (e) \; s_1 \text{ else } s_2) \| \ldots \| \; (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{if } (e) \; s_1 \text{ else } s_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3::\mathcal{L}_4::\mathcal{L}_5::\mathcal{L}_6::\mathcal{L}_7}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\text{p},[iep])} \\
\quad ((1, \gamma^1, \sigma^1_6, \Delta^1_3, \text{acc}, \text{skip}) \qquad \| \ldots \| \; (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}_6, \Delta^{\text{q}}_3, \text{acc}, \text{skip}))
\end{array}
$$

Private If Else (Location Tracking)

$$
\begin{array}{l}
\quad ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, e) \;\; \| \ldots \| \; (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, e)) \\
\Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} \; ((1, \gamma^1, \sigma^1_1, \Delta^1_1, \text{acc}, n^1) \| \ldots \| \; (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}_1, \Delta^{\text{q}}_1, \text{acc}, n^{\text{q}})) \qquad \{(e) \vdash \gamma^{\text{p}}\}^{\text{q}}_{\text{p}=1} \\
\{\text{Extract}(s_1, s_2, \gamma^{\text{p}}) = (\overline{x}, 1)\}^{\text{q}}_{\text{p}=1} \\
\{\text{Initialize}(\Delta^{\text{p}}_1, \overline{x}, \; \gamma^{\text{p}}, \; \sigma^{\text{p}}_1, \; n^{\text{p}}, \text{acc}+1) = (\gamma^{\text{p}}_1, \sigma^{\text{p}}_2, \Delta^{\text{p}}_2, \bar{l}^{\text{p}}_2)\}^{\text{q}}_{\text{p}=1} \\
\quad ((1, \gamma^1_1, \sigma^1_2, \Delta^1_2, \text{acc}+1, s_1) \;\; \| \ldots \| \; (\text{q}, \gamma^{\text{q}}_1, \sigma^{\text{q}}_2, \Delta^{\text{q}}_2, \text{acc}+1, s_1)) \\
\Downarrow^{\mathcal{L}_3}_{\mathcal{D}_2} \; ((1, \gamma^1_2, \sigma^1_3, \Delta^1_3, \text{acc}+1, \text{skip}) \| \ldots \| \; (\text{q}, \gamma^{\text{q}}_2, \sigma^{\text{q}}_3, \Delta^{\text{q}}_3, \text{acc}+1, \text{skip})) \\
\{\text{Restore}(\sigma^{\text{p}}_3, \Delta^{\text{p}}_3, \text{acc}+1) = (\sigma^{\text{p}}_4, \Delta^{\text{p}}_4, \bar{l}^{\text{p}}_4)\}^{\text{q}}_{\text{p}=1} \\
\quad ((1, \gamma^1_1, \sigma^1_4, \Delta^1_4, \text{acc}+1, s_2) \;\; \| \ldots \| \; (\text{q}, \gamma^{\text{q}}_1, \sigma^{\text{q}}_4, \Delta^{\text{q}}_4, \text{acc}+1, s_2)) \\
\Downarrow^{\mathcal{L}_5}_{\mathcal{D}_3} \; ((1, \gamma^1_3, \sigma^1_5, \Delta^1_5, \text{acc}+1, \text{skip}) \| \ldots \| \; (\text{q}, \gamma^{\text{q}}_3, \sigma^{\text{q}}_5, \Delta^{\text{q}}_5, \text{acc}+1, \text{skip})) \\
\{\text{Resolve_Retrieve}(\gamma^{\text{p}}_1, \sigma^{\text{p}}_5, \Delta^{\text{p}}_5, \text{acc}+1) = ([(v^{\text{p}}_{t1}, v^{\text{p}}_{e1}), \ldots, (v^{\text{p}}_{tm}, v^{\text{p}}_{em})], n^{\text{p}}, \bar{l}^{\text{p}}_6)\}^{\text{q}}_{\text{p}=1} \\
\text{MPC}_{resolve}([n^1, \ldots, n^{\text{q}}], [[(v^1_{t1}, v^1_{e1}), \ldots, (v^1_{tm}, v^1_{em})], \ldots, [(v^{\text{q}}_{t1}, v^{\text{q}}_{e1}), \ldots, (v^{\text{q}}_{tm}, v^{\text{q}}_{em})]]) = [[v^1_1, \ldots, v^1_m], \ldots [v^{\text{q}}_1, \ldots, v^{\text{q}}_m]] \\
\{\text{Resolve_Store}(\Delta^{\text{p}}_5, \sigma^{\text{p}}_5, \text{acc}+1, [v^{\text{p}}_1, \ldots, v^{\text{p}}_m]) = (\sigma^{\text{p}}_6, \Delta^{\text{p}}_6, \bar{l}^{\text{p}}_7)\}^{\text{q}}_{\text{p}=1} \\
\mathcal{L}_2 = (1, \bar{l}^1_2) \| \ldots \| \; (\text{q}, \bar{l}^{\text{q}}_2) \qquad \mathcal{L}_4 = (1, \bar{l}^1_4) \| \ldots \| \; (\text{q}, \bar{l}^{\text{q}}_4) \\
\mathcal{L}_6 = (1, \bar{l}^1_6) \| \ldots \| \; (\text{q}, \bar{l}^{\text{q}}_6) \qquad \mathcal{L}_7 = (1, \bar{l}^1_7) \| \ldots \| \; (\text{q}, \bar{l}^{\text{q}}_7) \\
\hline
\quad ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, \text{if } (e) \; s_1 \text{ else } s_2) \| \ldots \| \; (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{if } (e) \; s_1 \text{ else } s_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3::\mathcal{L}_4::\mathcal{L}_5::\mathcal{L}_6::\mathcal{L}_7}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\text{ALL},[iepd])} \\
\quad ((1, \gamma^1, \sigma^1_6, \Delta^1_6, \text{acc}, \text{skip}) \qquad \| \ldots \| \; (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}_6, \Delta^{\text{q}}_6, \text{acc}, \text{skip}))
\end{array}
$$

Figure 5.16: Multiparty SMC^2 semantic rules for the multiparty execution of private-conditioned if else statements.

Public If Else True

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e)\ \ \| C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p}, \gamma,\ \sigma_1, \Delta_1, \mathrm{acc}, n)\ \ \| C_1) \\ n\ \neq\ 0 & ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, s_1) \| C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma_1, \sigma_2, \Delta_2, \mathrm{acc}, \mathrm{skip}) \| C_2)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \text{if}\ (e)\ s_1\ \text{else}\ s_2) \| C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[iet])}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \| C_2)}$$

Public If Else False

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e)\ \ \| C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p},\ \gamma,\ \sigma_1, \Delta_1, \mathrm{acc}, n)\ \ \| C_1) \\ n\ =\ 0 & ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, s_2) \| C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma_1, \sigma_2, \Delta_2, \mathrm{acc}, \mathrm{skip}) \| C_2)\end{array}}{((\mathrm{p},\ \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \text{if}\ (e)\ s_1\ \text{else}\ s_2) \| C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ief])}\ ((\mathrm{p},\ \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \| C_2)}$$

While End

$$\frac{(e) \nvdash \gamma \qquad ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \| C) \Downarrow^{\mathcal{L}}_{\mathcal{D}} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \| C_1) \qquad n\ =\ 0}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \text{while}\ (e)\ s) \| C)\ \Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wle])}\ ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \| C_1)}$$

While Continue

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e) \| C)\ \ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p}, \gamma,\ \sigma_1, \Delta_1, \mathrm{acc}, n)\ \ \| C_1) \\ n\ \neq\ 0 & ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, s) \| C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma_1, \sigma_2, \Delta_2, \mathrm{acc}, \mathrm{skip}) \| C_2)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \text{while}\ (e)\ s) \| C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wlc])}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \text{while}\ (e)\ s) \| C_2)}$$

Figure 5.17: Multiparty SMC^2 semantic rules for public branches, loops, and sequencing.

Public Pointer Declaration

$$\frac{\begin{array}{ll}(ty = \text{public } bty*) & \text{acc} = 0 \qquad l = \phi() \\ \text{GetIndirection}(*) = i & \omega = \text{EncodePtr}(\text{public } bty*, [1, [(l_{default}, 0)], [1], i]) \\ \gamma_1 = \gamma[x \rightarrow (l, \text{ public } bty*)] & \sigma_1 = \sigma[l \rightarrow (\omega, \text{ public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))]\end{array}}{((\text{p}, \gamma, \sigma, \Delta, \text{acc}, ty\ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[dp])} ((\text{p}, \gamma_1, \sigma_1, \Delta, \text{acc}, \text{skip}) \parallel C)}$$

Private Pointer Declaration

$$\frac{\begin{array}{ll}l = \phi() & ((ty = bty*) \vee (ty = \text{private } bty*)) \wedge ((bty = \text{int}) \vee (bty = \text{float})) \\ \text{GetIndirection}(*) = i & \omega = \text{EncodePtr}(\text{private } bty*, [1, [(l_{default}, 0)], [1], i]) \\ \gamma_1 = \gamma[x \rightarrow (l, \text{private } bty*)] & \sigma_1 = \sigma[l \rightarrow (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1))]\end{array}}{((\text{p}, \gamma, \sigma, \Delta, \text{acc}, ty\ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[dp1])} ((\text{p}, \gamma_1, \sigma_1, \Delta, \text{acc}, \text{skip}) \parallel C)}$$

Public Pointer Write

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & ((\text{p}, \gamma, \sigma, \Delta, \text{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, (l_e, \mu_e)) \parallel C_1) \\ \gamma(x) = (l, \text{ public } bty*) & \sigma_1(l) = (\omega, \text{ public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{acc} = 0 & \text{DecodePtr}(\text{public } bty*, 1, \omega) = [1, [(l_1, \mu_1)], [1], i] \\ & \text{UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \text{public } bty*) = (\sigma_2, 1)\end{array}}{((\text{p}, \gamma, \sigma, \Delta, \text{acc}, x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)])}_{\mathcal{D}_1::(\text{p},[wp])} ((\text{p}, \gamma, \sigma_2, \Delta_1, \text{acc}, \text{skip}) \parallel C_1)}$$

Private Pointer Write

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & ((\text{p}, \gamma, \sigma, \Delta, \text{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, (l_e, \mu_e)) \parallel C_1) \\ \gamma(x) = (l, \text{ private } bty*) & \sigma_1(l) = (\omega, \text{ private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ & \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i] \\ & \text{UpdatePtr}(\sigma_1, (l, 0), [1, [(l_e, \mu_e)], [1], i], \text{private } bty*) = (\sigma_2, 1)\end{array}}{((\text{p}, \gamma, \sigma, \Delta, \text{acc}, x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)])}_{\mathcal{D}_1::(\text{p},[wp1])} ((\text{p}, \gamma, \sigma_2, \Delta_1, \text{acc}, \text{skip}) \parallel C_1)}$$

Private Pointer Write Multiple Locations

$$\frac{\begin{array}{ll}(bty = \text{int}) \vee (bty = \text{float}) & ((\text{p}, \gamma, \sigma, \Delta, \text{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, [\alpha_e, \bar{l}_e, \bar{j}_e, i]) \parallel C_1) \\ \gamma(x) = (l, \text{private } bty*) & \sigma_1(l) = (\omega, \text{ private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ & \text{DecodePtr}(\text{private } bty*, \alpha, \omega) = [\alpha, \bar{l}, \bar{j}, i] \\ & \text{UpdatePtr}(\sigma_1, (l, 0), [\alpha_e, \bar{l}_e, \bar{j}_e, i], \text{private } bty*) = (\sigma_2, 1)\end{array}}{((\text{p}, \gamma, \sigma, \Delta, \text{acc}, x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)])}_{\mathcal{D}_1::(\text{p},[wp2])} ((\text{p}, \gamma, \sigma_2, \Delta_1, \text{acc}, \text{skip}) \parallel C_1)}$$

Figure 5.18: Multiparty SMC^2 semantic rules for pointer declarations and writing.

Pointer Read Single Location

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ a\ bty*) & \sigma(l) = (\omega,\ a\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, a\ bty*, a, 1)) \\ & \mathrm{DecodePtr}(a\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i]\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C)\ \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[rp])}\ ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_1, \mu_1)) \parallel C)}$$

Private Pointer Read Multiple Locations

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ \mathrm{private}\ bty*) & \sigma(l) = (\omega,\ \mathrm{private}\ bty*,\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty*, \mathrm{private}, \alpha)) \\ (bty = \mathrm{int}) \vee (bty = \mathrm{float}) & \mathrm{DecodePtr}(\mathrm{private}\ bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C)\ \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[rp1])}\ ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [\alpha,\ \bar{l},\ \bar{j},\ i]) \parallel C)}$$

Pointer Dereference Single Location

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ a\ bty*) & \sigma(l) = (\omega,\ a\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, a\ bty*, a, 1)) \\ \mathrm{DecodePtr}(a\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1] & \mathrm{DerefPtr}(\sigma, a\ bty, (l_1, \mu_1)) = (n, 1)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C)\ \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp])}\ ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)}$$

Pointer Dereference Single Location Higher Level Indirection

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ a\ bty*) & \sigma(l) = (\omega_1,\ a\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, a\ bty*, a, 1)) \\ i > 1 & \mathrm{DecodePtr}(a\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i] \\ & \mathrm{DerefPtrHLI}(\sigma, a\ bty*, (l_1, \mu_1)) = ([1,\ [(l_2, \mu_2)],\ [1],\ i-1], 1)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C)\ \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp1])}\ ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)}$$

Private Pointer Dereference Single Location Higher Level Indirection

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ \mathrm{private}\ bty*) & \sigma(l) = (\omega_1,\ \mathrm{private}\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty*, \mathrm{private}, 1)) \\ i > 1 & \mathrm{DecodePtr}(\mathrm{private}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i] \\ & \mathrm{DerefPtrHLI}(\sigma, \mathrm{private}\ bty*, (l_1, \mu_1)) = ([\alpha,\ \bar{l},\ \bar{j},\ i-1], 1)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C)\ \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp2])}\ ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [\alpha,\ \bar{l},\ \bar{j},\ i-1]) \parallel C)}$$

Figure 5.19: Multiparty SMC^2 semantic rules for reading from a pointer and dereferencing a pointer at a single location.

Public Pointer Dereference Write Public Value

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1) \\ \gamma(x) = (l,\ \text{public } bty*) & \sigma_1(l) = (\omega,\ \text{public } bty*,\ 1, \mathrm{PermL}(\mathrm{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \mathrm{acc} = 0 & \mathrm{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ \text{public } bty,\ 1] \\ & \mathrm{UpdateOffset}(\sigma_1,\ (l_1, \mu_1),\ n, \text{public } bty) = (\sigma_2, 1)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x =\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)}$$

Private Pointer Dereference Write Single Location Private Value

$$\frac{\begin{array}{ll}(e) \vdash \gamma & ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1) \\ \gamma(x) = (l,\ \text{private } bty*) & \sigma_1(l) = (\omega,\ \text{private } bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \text{private } bty*, \text{private}, 1)) \\ (bty = \mathrm{int}) \vee (bty = \mathrm{float}) & \mathrm{DecodePtr}(\text{private } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1] \\ & \mathrm{DynamicUpdate}(\Delta_1, \sigma_1, [(l_1, \mu_1)],\ \mathrm{acc}, \text{private } bty) = (\Delta_2, \bar{l}_1) \\ & \mathrm{UpdateOffset}(\sigma_1,\ (l_1, \mu_1),\ n, \text{private } bty) = (\sigma_2, 1)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp3])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)}$$

Private Pointer Dereference Write Single Location Public Value

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1) \\ \gamma(x) = (l,\ \text{private } bty*) & \sigma_1(l) = (\omega,\ \text{private } bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \text{private } bty*, \text{private}, 1)) \\ (bty = \mathrm{int}) \vee (bty = \mathrm{float}) & \mathrm{DecodePtr}(\text{private } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1] \\ & \mathrm{DynamicUpdate}(\Delta_1, \sigma_1, [(l_1, \mu_1)],\ \mathrm{acc}, \text{private } bty) = (\Delta_2, \bar{l}_1) \\ & \mathrm{UpdateOffset}(\sigma_1,\ (l_1, \mu_1),\ \mathrm{encrypt}(n), \text{private } bty) = (\sigma_2, 1)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp4])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)}$$

Public Pointer Dereference Write Higher Level Indirection

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ (l_e, \mu_e)) \parallel C_1) \\ \gamma(x) = (l,\ \text{public } bty*) & \sigma_1(l) = (\omega,\ \text{public } bty*,\ 1, \mathrm{PermL}(\mathrm{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \mathrm{acc} = 0 & \mathrm{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i] \\ i > 1 & \mathrm{UpdatePtr}(\sigma_1,\ (l_1, \mu_1),\ [1,\ [(l_e, \mu_e)],\ [1],\ i-1], \text{public } bty*) = (\sigma_2, 1)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)}$$

Private Pointer Dereference Write to Single Location Higher Level Indirection

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ (l_e, \mu_e)) \parallel C_1) \\ \gamma(x) = (l,\ \text{private } bty*) & \sigma_1(l) = (\omega,\ \text{private } bty*,\ 1, \mathrm{PermL}(\mathrm{Freeable}, \text{private } bty*, \text{private}, 1)) \\ & \mathrm{DecodePtr}(\text{private } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i] \\ i > 1 & \mathrm{DynamicUpdate}(\Delta_1, \sigma_1, [(l_1, \mu_1)],\ \mathrm{acc}, \text{private } bty*) = (\Delta_2, \bar{l}_1) \\ & \mathrm{UpdatePtr}(\sigma_1,\ (l_1, \mu_1),\ [1,\ [(l_e, \mu_e)],\ [1],\ i-1], \text{private } bty*) = (\sigma_2, 1)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp5])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)}$$

Private Pointer Dereference Write Multiple Locations to Single Location Higher Level Indirection

$$\frac{\begin{array}{ll} & ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ [\alpha, \bar{l}_e, \bar{j}_e, i-1]) \parallel C_1) \\ \gamma(x) = (l,\ \text{private } bty*) & \sigma_1(l) = (\omega,\ \text{private } bty*,\ 1, \mathrm{PermL}(\mathrm{Freeable}, \text{private } bty*, \text{private}, 1)) \\ & \mathrm{DecodePtr}(\text{private } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i] \\ i > 1 & \mathrm{DynamicUpdate}(\Delta_1, \sigma_1, [(l_1, \mu_1)],\ \mathrm{acc}, \text{private } bty*) = (\Delta_2, \bar{l}_1) \\ & \mathrm{UpdatePtr}(\sigma_1,\ (l_1, \mu_1),\ [\alpha, \bar{l}_e, \bar{j}_e, i-1], \text{private } bty*) = (\sigma_2, 1)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)}$$

Figure 5.20: Multiparty SMC^2 semantic rules for pointer dereference write.

Public Array Declaration

$$\frac{\begin{array}{ll} \text{acc} = 0 & (ty = \text{public } bty) \\ (e) \nvdash \gamma & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ \alpha) \parallel C_1) \\ \alpha > 0 & \omega = \text{EncodePtr}(\text{public const } bty*,\ [1,\ [(l_1, 0)],\ [1],\ 1]) \\ l = \phi() & \omega_1 = \text{EncodeArr}(\text{public } bty, \alpha, \text{NULL}) \\ l_1 = \phi() & \gamma_1 = \gamma[x\ \rightarrow\ (l,\ \text{public const } bty*)] \\ & \sigma_2 = \sigma_1[l\ \rightarrow\ (\omega,\ \text{public const } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))] \\ & \sigma_3 = \sigma_2[l_1\ \rightarrow\ (\omega_1,\ \text{public } bty,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, \alpha))] \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x[e]) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0),(l_1,0)])}_{\mathcal{D}_1::(\text{p},[da])} ((\text{p}, \gamma_1,\ \sigma_3,\ \Delta,\ \text{acc},\ \text{skip}) \parallel C_1)}$$

Private Array Declaration

$$\frac{\begin{array}{ll} (e) \nvdash \gamma & ((ty = \text{private } bty) \vee (ty = bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float})) \\ & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ \alpha) \parallel C_1) \\ \alpha > 0 & \omega = \text{EncodePtr}(\text{private const } bty*,\ [1,\ [(l_1, 0)],\ [1],\ 1]) \\ l = \phi() & \omega_1 = \text{EncodeArr}(\text{private } bty, \alpha, \text{NULL}) \\ l_1 = \phi() & \gamma_1 = \gamma[x\ \rightarrow\ (l,\ \text{private const } bty*)] \\ & \sigma_2 = \sigma_1[l\ \rightarrow\ (\omega,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))] \\ & \sigma_3 = \sigma_2[l_1\ \rightarrow\ (\omega_1,\ \text{private } bty,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha))] \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x[e]) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0),(l_1,0)])}_{\mathcal{D}_1::(\text{p},[da1])} ((\text{p}, \gamma_1,\ \sigma_3,\ \Delta,\ \text{acc},\ \text{skip}) \parallel C_1)}$$

Array Declaration Assignment

$$\frac{\begin{array}{l} ((\text{p},\ \gamma,\ \sigma,\ \Delta,\ \text{acc}, ty\ x[e_1]) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma_1, \sigma_1, \Delta_1, \text{acc}, \text{skip}) \parallel C_1) \\ ((\text{p}, \gamma_1, \sigma_1, \Delta_1, \text{acc}, x = e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\text{p}, \gamma_1, \sigma_2, \Delta_2, \text{acc}, \text{skip}) \parallel C_2) \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x[e_1] = e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[das])} ((\text{p}, \gamma_1,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{skip}) \parallel C_2)}$$

Public Array Read Public Index

$$\frac{\begin{array}{ll} (e) \nvdash \gamma & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ i) \parallel C_1) \\ \gamma(x) = (l,\ \text{public const } bty*) & \sigma_1(l) = (\omega,\ \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)) \\ & \text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma_1(l_1) = (\omega_1, \text{public } bty, \alpha, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, \alpha)) \\ 0 \leq i \leq \alpha - 1 & \text{DecodeArr}(\text{public } bty,\ i,\ \omega_1) = n_i \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e]) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::(\text{p},[ra])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ n_i) \parallel C_1)}$$

Private Array Read Public Index

$$\frac{\begin{array}{ll} \gamma(x) = (l,\ \text{private const } bty*) & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ i) \parallel C_1) \\ (e) \nvdash \gamma & \sigma_1(l) = (\omega,\ \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ & \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ 0 \leq i \leq \alpha - 1 & \sigma_1(l_1) = (\omega_1, \text{private } bty, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha)) \\ & \text{DecodeArr}(\text{private } bty,\ i,\ \omega_1) = n_i \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e]) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::(\text{p},[ra1])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ n_i) \parallel C_1)}$$

Figure 5.21: Multiparty SMC^2 semantic rules for array declarations and reading from a public index.

Aside from reading and writing to a private index, all array operations will occur locally. This is because we are simply accessing data at or copying data to a known position in our local memory - we do not need to know anything further about the data during these operations, therefore no communication between parties is needed in these rules. With array evaluations, all evaluations that use a public index behave nearly identically to those over public data. The difference is that we have an additional check within array writes at a public index to see if we are in a private-conditioned branch; if so, we must ensure we properly track the modification

Public Array Write Public Value Public Index

$$\frac{\begin{array}{ll} \text{acc} = 0 & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc}, e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, i) \parallel C_1) \\ (e_1, e_2) \nvdash \gamma & ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\text{p}, \gamma, \sigma_2, \Delta_2, \text{acc}, n) \parallel C_2) \\ \gamma(x) = (l,\ \text{public const}\ bty*) & \sigma_2(l) = (\omega, \text{public const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{public const}\ bty*, \text{public}, 1)) \\ & \text{DecodePtr}(\text{public const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma_2(l_1) = (\omega_1, \text{public}\ bty, \alpha, \text{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, \alpha)) \\ 0 \leq i \leq \alpha - 1 & \text{UpdateArr}(\sigma_2,\ (l_1, i),\ n,\ \text{public}\ bty) = \sigma_3 \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e_1]\ = e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\text{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[wa])}\ ((\text{p}, \gamma,\ \sigma_3,\ \Delta_2,\ \text{acc},\ \text{skip}) \parallel C_2)}$$

Private Array Write Private Value Public Index

$$\frac{\begin{array}{ll} \gamma(x) = (l,\ \text{private const}\ bty*) & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc}, e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, i) \parallel C_1) \\ (e_1) \nvdash \gamma & ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\text{p}, \gamma, \sigma_2, \Delta_2, \text{acc}, n) \parallel C_2) \\ (e_2) \vdash \gamma & \sigma_2(l) = (\omega, \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1)) \\ & \text{DecodePtr}(\text{private const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma_2(l_1) = (\omega_1, \text{private}\ bty, \alpha, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, \alpha)) \\ 0 \leq i \leq \alpha - 1 & \text{DynamicUpdate}(\Delta_2, \sigma_2, [(l_1, i)],\ \text{acc}, \text{private}\ bty) = \Delta_3 \\ & \text{UpdateArr}(\sigma_2,\ (l_1, i),\ n,\ \text{private}\ bty) = \sigma_3 \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e_1]\ = e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\text{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[wa2])}\ ((\text{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \text{acc},\ \text{skip}) \parallel C_2)}$$

Private Array Write Public Value Public Index

$$\frac{\begin{array}{ll} \gamma(x) = (l,\ \text{private const}\ bty*) & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ i) \parallel C_1) \\ (e_1, e_2) \nvdash \gamma & ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ n) \parallel C_2) \\ & \sigma_2(l) = (\omega, \text{private const}\ bty*, 1, \text{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1)) \\ & \text{DecodePtr}(\text{private const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma_2(l_1) = (\omega_1, \text{private}\ bty, \alpha, \text{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, \alpha)) \\ 0 \leq i \leq \alpha - 1 & \text{DynamicUpdate}(\Delta_2, \sigma_2, [(l_1, i)],\ \text{acc}, \text{private}\ bty) = \Delta_3 \\ & \text{UpdateArr}(\sigma_2,\ (l_1, i),\ \text{encrypt}(n),\ \text{private}\ bty) = \sigma_3 \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e_1]\ = e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\text{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[wa1])}\ ((\text{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \text{acc},\ \text{skip}) \parallel C_2)}$$

Figure 5.22: Multiparty SMC^2 semantic rules for writing to an array.

made (this is because a public index that is not hard-coded could have lead to an out-of-bounds array write).

Read Entire Array

$$\frac{\begin{array}{ll}\gamma(x) = (l, a \text{ const } bty*) & \sigma(l) = (\omega, a \text{ const } bty*, 1, \text{PermL}(\text{Freeable}, a \text{ const } bty*, a, 1)) \\ & \text{DecodePtr}(a \text{ const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma(l_1) = (\omega_1, a\ bty, \alpha, \text{PermL}(\text{Freeable}, a\ bty, a, \alpha)) \\ & \forall i \in \{0...\alpha - 1\} \quad \text{DecodeArr}(a\ bty, i, \omega_1) = n_i\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x) \parallel C)\ \Downarrow^{(\text{p},[(l,0),(l_1,0),\ldots,(l_1,\alpha-1)])}_{(\text{p},[rea])}\ ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ [n_0, ..., n_{\alpha-1}]) \parallel C)}$$

Write Entire Public Array

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ \text{public const } bty*) & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ [n_0, ..., n_{\alpha_e-1}]) \parallel C_1) \\ \text{acc} = 0 & \sigma_1(l) = (\omega,\ \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)) \\ (e) \nvdash \gamma & \text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma_1(l_1) = (\omega_1, \text{public } bty, \alpha, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, \alpha)) \\ \alpha_e = \alpha & \forall i \in \{0...\alpha - 1\} \quad \text{UpdateArr}(\sigma_{1+i},\ (l_1, i),\ n_i,\ \text{public } bty) = \sigma_{2+i}\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x = e) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0),(l_1,0),\ldots,(l_1,\alpha-1)])}_{\mathcal{D}_1::(\text{p},[wea1])}\ ((\text{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)}$$

Write Entire Private Array

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ \text{private const } bty*) & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ [n_0, ..., n_{\alpha_e-1}]) \parallel C_1) \\ (e) \vdash \gamma & \sigma_1(l) = (\omega,\ \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ & \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma_1(l_1) = (\omega_1, \text{private } bty, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha)) \\ \alpha_e = \alpha & \forall i \in \{0...\alpha - 1\} \quad \text{UpdateArr}(\sigma_{1+i},\ (l_1, i),\ n_i,\ \text{private } bty) = \sigma_{2+i}\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x = e) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0),(l_1,0),\ldots,(l_1,\alpha-1)])}_{\mathcal{D}_1::(\text{p},[wea1])}\ ((\text{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)}$$

Private Array Write Entire Public Array

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ \text{private const } bty*) & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ [n_0, ..., n_{\alpha_e-1}]) \parallel C_1) \\ (e) \nvdash \gamma & \sigma_1(l) = (\omega,\ \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ & \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma_1(l_1) = (\omega_1, \text{private } bty, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha)) \\ \alpha_e = \alpha & \forall i \in \{0...\alpha - 1\} \quad \text{UpdateArr}(\sigma_{1+i},\ (l_1, i),\ \text{encrypt}(n_i),\ \text{private } bty) = \sigma_{2+i}\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x = e) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0),(l_1,0),\ldots,(l_1,\alpha-1)])}_{\mathcal{D}_1::(\text{p},[wea2])}\ ((\text{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)}$$

Figure 5.23: Multiparty SMC^2 semantic rules for reading and writing an entire array.

Public Array Read Out of Bounds Public Index

$$\frac{\begin{array}{ll} \gamma(x) = (l,\ \text{public const } bty*) & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ i) \parallel C_1) \\ (e) \nvdash \gamma & \sigma_1(l) = (\omega,\ \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)) \\ & \text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma_1(l_1) = (\omega_1, \text{public } bty, \alpha, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, \alpha)) \\ (i < 0) \vee (i \geq \alpha) & \text{ReadOOB}(i, \alpha, l_1, \text{public } bty, \sigma_1) = (n, 1, (l_2, \mu)) \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e]) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::(\text{p},[rao])}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ n) \parallel C_1)}$$

Private Array Read Out of Bounds Public Index

$$\frac{\begin{array}{ll} \gamma(x) = (l,\ \text{private const } bty*) & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ i) \parallel C_1) \\ (e) \nvdash \gamma & \sigma_1(l) = (\omega,\ \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ & \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma_1(l_1) = (\omega_1, \text{private } bty, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha)) \\ (i < 0) \vee (i \geq \alpha) & \text{ReadOOB}(i, \alpha, l_1, \text{private } bty, \sigma_1) = (n, 1, (l_2, \mu)) \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e]) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::(\text{p},[rao1])}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ n) \parallel C_1)}$$

Public Array Write Out of Bounds Public Index Public Value

$$\frac{\begin{array}{ll} (e_1, e_2) \nvdash \gamma & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc}, e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ i) \parallel C_1) \\ \text{acc} = 0 & ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, e_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ n) \parallel C_2) \\ \gamma(x) = (l, \text{public const } bty*) & \sigma_2(l) = (\omega, \text{public const } bty*, 1, \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1)) \\ & \text{DecodePtr}(\text{public const } bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma_2(l_1) = (\omega_1,\ \text{public } bty,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, \alpha)) \\ (i < 0) \vee (i \geq \alpha) & \text{WriteOOB}(n,\ i,\ \alpha,\ l_1,\ \text{public } bty,\ \sigma_2,\ \text{acc}) = (\sigma_3,\ 1, (l_2, \mu)) \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e_1]\ = e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\text{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[wao])}\ ((\text{p}, \gamma,\ \sigma_3,\ \Delta_2,\ \text{acc},\ \text{skip}) \parallel C_2)}$$

Private Array Write Out of Bounds Public Index Private Value

$$\frac{\begin{array}{ll} \gamma(x) = (l, \text{private const } bty*) & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc}, e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, i) \parallel C_1) \\ (e_1) \nvdash \gamma & ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, e_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\text{p}, \gamma, \sigma_2, \Delta_2, \text{acc}, n) \parallel C_2) \\ (e_2) \vdash \gamma & \sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ & \text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma_2(l_1) = (\omega_1,\ \text{private } bty,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha)) \\ (i < 0) \vee (i \geq \alpha) & \text{WriteOOB}(n,\ i,\ \alpha,\ l_1,\ \text{private } bty,\ \sigma_2,\ \Delta_2, \text{acc}) = (\sigma_3,\ \Delta_3, 1, (l_2, \mu)) \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e_1]\ = e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\text{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[wao2])}\ ((\text{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \text{acc},\ \text{skip}) \parallel C_2)}$$

Private Array Write Public Value Out of Bounds Public Index

$$\frac{\begin{array}{ll} (e_1, e_2) \nvdash \gamma & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ i) \parallel C_1) \\ \gamma(x) = (l, \text{private const } bty*) & ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ e) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ n) \parallel C_2) \\ & \sigma_2(l) = (\omega, \text{private const } bty*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) \\ & \text{DecodePtr}(\text{private const } bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ & \sigma_2(l_1) = (\omega_1,\ \text{private } bty,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha)) \\ (i < 0) \vee (i \geq \alpha) & \text{WriteOOB}(\text{encrypt}(n),\ i,\ \alpha,\ l_1,\ \text{private } bty,\ \sigma_2,\ \Delta_2, \text{acc}) = (\sigma_3, \Delta_3, 1, (l_2, \mu)) \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e_1]\ = e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\text{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[wao1])}\ ((\text{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \text{acc},\ \text{skip}) \parallel C_2)}$$

Figure 5.24: Multiparty SMC^2 semantic rules for reading and writing out of bounds for arrays.

Pre-Increment Private Int Variable

$$\frac{\begin{array}{ll}\gamma(x) = (l, \text{private int}) & \sigma(l) = (\omega, \text{private int}, 1, \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1)) \\ & \text{DecodeVal}(\text{private int}, \omega) = n_1 \\ n_2 = n_1 + \text{encrypt}(1) & \text{UpdateVal}(\sigma, l, n_2, \text{private int}) = \sigma_1\end{array}}{((\text{p}, \gamma, \sigma, \Delta, \text{acc}, ++\ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin3])} ((\text{p}, \gamma, \sigma_1, \Delta, \text{acc}, n_2) \parallel C)}$$

Pre-Increment Private Pointer Multiple Locations

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ \text{private } bty*) & \sigma(l) = (\omega,\ \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ & \text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ 1] \\ & \text{IncrementList}(\bar{l},\ \tau(\text{private } bty), \sigma) = (\bar{l}_1, 1) \\ & \text{UpdatePtr}(\sigma,\ (l,0),\ [\alpha,\ \bar{l}_1,\ \bar{j},\ 1],\ \text{private } bty*) = (\sigma_1,\ 1)\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C)\ \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin4])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ [n,\ \bar{l}_1,\ \bar{j},\ 1]) \parallel C)}$$

Pre-Increment Private Pointer Higher Level Indirection Multiple Locations

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ \text{private } bty*) & \sigma(l) = (\omega,\ \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) \\ & \text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i] \\ & \text{IncrementList}(\bar{l},\ \tau(\text{private } bty*), \sigma) = (\bar{l}_1, 1) \\ & \text{UpdatePtr}(\sigma,\ (l,0),\ [\alpha,\ \bar{l}_1,\ \bar{j},\ i],\ \text{private } bty*) = (\sigma_1,\ 1)\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C)\ \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin5])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ [\alpha,\ \bar{l}_1,\ \bar{j},\ i]) \parallel C)}$$

Pre-Increment Private Pointer Single Location

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ \text{private } bty*) & \sigma(l) = (\omega,\ \text{private } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1)) \\ & \text{DecodePtr}(\text{private } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1] \\ & ((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{private } bty), \sigma) \\ & \text{UpdatePtr}(\sigma,\ (l,0),\ [1,\ [(l_2, \mu_2)],\ [1],\ 1],\ \text{private } bty*) = (\sigma_1,\ 1)\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C)\ \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin6])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ (l_2, \mu_2)) \parallel C)}$$

Pre-Increment Private Pointer Higher Level Indirection Single Location

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ \text{private } bty*) & \sigma(l) = (\omega,\ \text{private } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1)) \\ & \text{DecodePtr}(\text{private } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i] \\ i > 1 & ((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{private } bty*), \sigma) \\ & \text{UpdatePtr}(\sigma,\ (l,0),\ [1,\ [(l_2, \mu_2)],\ [1],\ i],\ \text{private } bty) = (\sigma_1,\ 1)\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C)\ \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin7])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ (l_2, \mu_2)) \parallel C)}$$

Figure 5.25: Multiparty SMC^2 pre-increment rules for private int variables and for private pointers.

Incrementing a private int value occurs locally. Incrementing the locations of pointers (public and private) will always be local, as all locations pointed to by the pointer will be incremented by the appropriate amount, regardless of which is the true location. This does not modify which is the true location, nor require knowing which is the true location.

Memory allocation (public or private) occurs locally. When allocating private memory, we provide the `pmalloc` builtin function to internally handle obtaining the size of the private type; the programmer to only needs to know how many elements of the given type they desire to allocate. In rule Private Malloc, we assert that the given type is either private int or private float, as this function only handles those types, and that the accumulator acc is 0 (i.e., we are not inside an **`if else`** statement branching on private data, as this function causes public side effects). Then we evaluate e to n and obtain the next open memory location l from ϕ. We

Pre-Increment Public Variable

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ \text{public}\ bty) & \sigma(l) = (\omega,\ \text{public}\ bty,\ 1,\ \text{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, 1)) \\ \text{acc} = 0 & \text{DecodeVal}(\text{public}\ bty,\ \omega) = n \\ n_1 = n + 1 & \text{UpdateVal}(\sigma,\ l,\ n_1,\ \text{public}\ bty) = \sigma_1\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C)\ \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin])}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ n_1) \parallel C)}$$

Pre-Increment Public Pointer Single Location

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ \text{public}\ bty*) & \sigma(l) = (\omega,\ \text{public}\ bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1)) \\ & \text{DecodePtr}(\text{public}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1] \\ & ((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public}\ bty), \sigma) \\ & \text{UpdatePtr}(\sigma,\ (l, 0),\ [1,\ [(l_2, \mu_2)],\ [1],\ 1],\ \text{public}\ bty*) = (\sigma_1,\ 1)\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C)\ \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin1])}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ (l_2, \mu_2)) \parallel C)}$$

Pre-Increment Public Pointer Higher Level Indirection Single Location

$$\frac{\begin{array}{ll}\gamma(x) = (l,\ \text{public}\ bty*) & \sigma(l) = (\omega,\ \text{public}\ bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{public}\ bty*, \text{public}, 1)) \\ & \text{DecodePtr}(\text{public}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i] \\ i > 1 & ((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public}\ bty*), \sigma) \\ & \text{UpdatePtr}(\sigma,\ (l, 0),\ [1,\ [(l_2, \mu_2)],\ [1],\ i],\ \text{public}\ bty) = (\sigma_1,\ 1)\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C)\ \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin2])}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ (l_2, \mu_2)) \parallel C)}$$

Figure 5.26: Multiparty SMC^2 semantic rules for the public pre-increment operator.

add to σ_1 the new mapping from l to the tuple of a NULL set of bytes; the type ty; the size n; and a list of private, Freeable permissions. As with public `malloc`, we return the new location, $(l, 0)$. Freeing allocated memory from a pointer with a single location occurs locally, regardless of if the pointer is public or private. This is because the true location of the pointer is publicly known.

Public Malloc

$$\frac{\begin{array}{lll} \text{acc} = 0 & (e) \nvdash \gamma & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ n) \parallel C_1) \\ l = \phi() & & \sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, n)\big)\big] \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{malloc}(e)) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)])}_{\mathcal{D}_1::(\text{p},[mal])}\ ((\text{p}, \gamma,\ \sigma_2,\ \Delta,\ \text{acc},\ (l,0)) \parallel C_1)}$$

Private Malloc

$$\frac{\begin{array}{ll} (e) \nvdash \gamma & (ty = \text{private } bty*) \vee (ty = \text{private } bty) \\ \text{acc} = 0 & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ n) \parallel C_1) \\ l = \phi() & \sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL},\ \text{void}*,\ n \cdot \tau(ty),\ \text{PermL}(\text{Freeable}, \text{void}*, \text{private}, n \cdot \tau(ty))\big)\big] \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{pmalloc}(e,\ ty)) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)])}_{\mathcal{D}_1::(\text{p},[malp])}\ ((\text{p}, \gamma,\ \sigma_2,\ \Delta,\ \text{acc},\ (l,0)) \parallel C_1)}$$

Public Free

$$\frac{\begin{array}{ll} \gamma(x) = (l,\ \text{public } bty*) & \sigma(l) = (\omega, \text{public } bty*, 1, \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1)) \\ \text{acc} = 0 & \text{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1] \\ \text{CheckFreeable}(\gamma, [(l_1, 0)], [1], \sigma) = 1 & \text{Free}(\sigma,\ l_1) = (\sigma_1, (l_1, 0)) \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{free}(x)) \parallel C)\ \Downarrow^{(\text{p},[(l,0),(l_1,0)])}_{(\text{p},[fre])}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ \text{skip}) \parallel C)}$$

Private Free Single Location

$$\frac{\begin{array}{ll} \gamma(x) = (l,\ \text{private } bty*) & \sigma(l) = (\omega, \text{private } bty*, 1, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, 1)) \\ \text{acc} = 0 & \text{DecodePtr}(\text{private } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [j],\ 1] \\ \text{CheckFreeable}(\gamma, [(l_1, 0)], [j], \sigma) = 1 & \text{Free}(\sigma,\ l_1) = (\sigma_1, (l_1, 0)) \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{pfree}(x)) \parallel C)\ \Downarrow^{(\text{p},[(l,0),(l_1,0)])}_{(\text{p},[pfre])}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ \text{skip}) \parallel C)}$$

Cast Private Location

$$\frac{\begin{array}{ll} (ty = \text{private } bty*) & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ (l,0)) \parallel C_1) \\ & \sigma_1 = \sigma_2\big[l \rightarrow \big(\omega,\ \text{void}*,\ n,\ \text{PermL}(\text{Freeable}, \text{void}*, \text{private}, n)\big)\big] \\ & \sigma_3 = \sigma_2\big[l \rightarrow \big(\omega,\ ty,\ \frac{n}{\tau(ty)},\ \text{PermL}(\text{Freeable}, ty, \text{private}, \frac{n}{\tau(ty)})\big)\big] \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ (ty)\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)])}_{\mathcal{D}_1::(\text{p},[cl1])}\ ((\text{p}, \gamma,\ \sigma_3,\ \Delta_1,\ \text{acc},\ (l,0)) \parallel C_1)}$$

Cast Public Location

$$\frac{\begin{array}{ll} (ty = \text{public } bty*) & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ (l,0)) \parallel C_1) \\ \text{acc} = 0 & \sigma_1 = \sigma_2\big[l \rightarrow \big(\omega,\ \text{void}*,\ n,\ \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, n)\big)\big] \\ & \sigma_3 = \sigma_2\big[l \rightarrow \big(\omega,\ ty,\ \frac{n}{\tau(ty)},\ \text{PermL}(\text{Freeable}, ty, \text{public}, \frac{n}{\tau(ty)})\big)\big] \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ (ty)\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)])}_{\mathcal{D}_1::(\text{p},[cl])}\ ((\text{p}, \gamma,\ \sigma_3,\ \Delta_1,\ \text{acc},\ (l,0)) \parallel C_1)}$$

Cast Public Value

$$\frac{\begin{array}{ll} (e) \nvdash \gamma & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ n) \parallel C_1) \\ (ty = \text{public } bty) & n_1 = \text{Cast}(\text{public},\ ty,\ n) \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ (ty)\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\text{p},[cv])}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ n_1) \parallel C_1)}$$

Cast Private Value

$$\frac{\begin{array}{ll} (e) \vdash \gamma & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ n) \parallel C_1) \\ (ty = \text{private } bty) & n_1 = \text{Cast}(\text{private},\ ty,\ n) \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ (ty)\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\text{p},[cv1])}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ n_1) \parallel C_1)}$$

Address Of

$$\frac{\gamma(x) = (l,\ ty)}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \&x) \parallel C)\ \Downarrow^{\epsilon}_{(\text{p},[loc])}\ ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ (l,0)) \parallel C)}$$

Figure 5.27: Multiparty SMC^2 semantic rules for memory allocation and deallocation, casting, and finding the address of a variable.

Function Declaration

$$\frac{\begin{array}{lll} \text{acc} = 0 & l = \phi() & \text{GetFunTypeList}(\overline{p}) = \overline{ty} \\ \gamma_1 = \gamma[x \rightarrow (l,\ \overline{ty} \rightarrow ty)] & & \omega = \text{EncodeFun}(\text{NULL}, -1, \overline{p}) \\ & & \sigma_1 = \sigma[l \rightarrow (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)})] \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x(\overline{p}))\ \|\ C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[df])} ((\text{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \text{acc},\ \text{skip})\ \|\ C)}$$

Function Definition

$$\frac{\begin{array}{lll} \text{acc} = 0 & \text{GetFunTypeList}(\overline{p}) = \overline{ty} & \text{CheckPublicEffects}(s,\ x,\ \gamma, \sigma) = n \\ x \notin \gamma & \text{EncodeFun}(s,\ n,\ \overline{p}) = \omega & \\ l = \phi() & \gamma_1 = \gamma[x \rightarrow (l,\ \overline{ty} \rightarrow ty)] & \sigma_1 = \sigma[l \rightarrow (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)})] \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x(\overline{p})\{s\})\ \|\ C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[fd])} ((\text{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \text{acc},\ \text{skip})\ \|\ C)}$$

Pre-Declared Function Definition

$$\frac{\begin{array}{lll} \text{acc} = 0 & x \in \gamma & \text{CheckPublicEffects}(s,\ x,\ \gamma, \sigma) = n \\ \gamma(x) = (l,\ \overline{ty} \rightarrow ty) & & \sigma = \sigma_1[l \rightarrow (\omega_1,\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)})] \\ \text{DecodeFun}(\omega) = (\text{NULL}, -1, \overline{p}) & & \\ \text{EncodeFun}(s,\ n,\ \overline{p}) = \omega & & \sigma_2 = \sigma_1[l \rightarrow (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)})] \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x(\overline{p})\{s\})\ \|\ C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[fpd])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta,\ \text{acc},\ \text{skip})\ \|\ C)}$$

Function Call Without Public Side Effects

$$\frac{\begin{array}{ll} \gamma(x) = (l,\ \overline{ty} \rightarrow ty) & \sigma(l) = (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)}) \\ \text{DecodeFun}(\omega) = (s,\ n,\ \overline{p}) & \text{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1 \\ n = 0 & ((\text{p},\ \gamma,\ \sigma,\ \Delta,\ \text{acc}, s_1)\ \|\ C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma_1, \sigma_1, \Delta_1, \text{acc}, \text{skip})\ \|\ C_1) \\ & ((\text{p}, \gamma_1, \sigma_1, \Delta_1, \text{acc}, s)\ \|\ C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\text{p}, \gamma_2, \sigma_2, \Delta_2, \text{acc}, \text{skip})\ \|\ C_2) \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x(\overline{e}))\ \|\ C) \Downarrow^{(\text{p},[(l,0)]) :: \mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\text{p},[fc1])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{skip})\ \|\ C_2)}$$

Function Call With Public Side Effects

$$\frac{\begin{array}{ll} \gamma(x) = (l,\ \overline{ty} \rightarrow ty) & \sigma(l) = (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun(public)}) \\ \text{DecodeFun}(\omega) = (s,\ n,\ \overline{p}) & \text{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1 \\ \text{acc} = 0 & ((\text{p},\ \gamma,\ \sigma,\ \Delta,\ \text{acc}, s_1)\ \|\ C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma_1, \sigma_1, \Delta_1, \text{acc}, \text{skip})\ \|\ C_1) \\ n = 1 & ((\text{p}, \gamma_1, \sigma_1, \Delta_1, \text{acc}, s)\ \|\ C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\text{p}, \gamma_2, \sigma_2, \Delta_2, \text{acc}, \text{skip})\ \|\ C_2) \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x(\overline{e}))\ \|\ C) \Downarrow^{(\text{p},[(l,0)]) :: \mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\text{p},[fc])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{skip})\ \|\ C_2)}$$

Size of Type

$$\frac{(ty) \nvdash \gamma \qquad n = \tau(ty)}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{sizeof}(ty))\ \|\ C) \Downarrow^{\epsilon}_{(\text{p},[ty])} ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ n)\ \|\ C)}$$

Figure 5.28: Multiparty SMC^2 semantic rules for functions and finding the size of a type.

At the top level (as shown within our function rules), functions do not need to be executed in a multiparty setting. Given our model uses big-step semantics, we show the overall results of executing the statement(s) for the function - any statements that require multiparty execution will be subsequently executed using their respective multiparty rules, without requiring the top-level rules to be executed in a multiparty setting. When functions are defined, we evaluate whether or not they have public side effects. This is necessary to know which functions should not be allowed to execute within either branch of a private-conditioned if else statement, as neither branch can have public side effects in order to prevent leakage of information about which branch was intended to be executed (and therefore leakage about the private condition itself).

Public Addition

$$\frac{(e_1, e_2) \nvdash \gamma \qquad ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \quad ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2) \qquad n_1 + n_2 = n_3}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 + e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bp])}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2)}$$

Public Subtraction

$$\frac{(e_1, e_2) \nvdash \gamma \qquad ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \quad ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2) \qquad n_1 - n_2 = n_3}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 - e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bs])}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2)}$$

Public Multiplication

$$\frac{(e_1, e_2) \nvdash \gamma \qquad ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \quad ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2) \qquad n_1 \cdot n_2 = n_3}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 \cdot e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bm])}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2)}$$

Public Division

$$\frac{(e_1, e_2) \nvdash \gamma \qquad ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \quad ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2) \qquad n_1 \div n_2 = n_3}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 \div e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bd])}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2)}$$

Public Less Than True

$$\frac{(e_1, e_2) \nvdash \gamma \qquad ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \quad ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2) \qquad (n_1 < n_2) = 1}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltt])}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)}$$

Public Less Than False

$$\frac{(e_1, e_2) \nvdash \gamma \qquad ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \quad ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2) \qquad (n_1 < n_2) = 0}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltf])}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)}$$

Public Equal To True

$$\frac{(e_1, e_2) \nvdash \gamma \qquad ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \quad ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2) \qquad (n_1 = n_2) = 1}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 == e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[eqt])}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)}$$

Public Equal To False

$$\frac{(e_1, e_2) \nvdash \gamma \qquad ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \quad ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2) \qquad (n_1 = n_2) = 0}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 == e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[eqf])}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)}$$

Public Not Equal To True

$$\frac{(e_1, e_2) \nvdash \gamma \qquad ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \quad ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2) \qquad (n_1 \neq n_2) = 1}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 ! = e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[net])}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)}$$

Public Not Equal To False

$$\frac{(e_1, e_2) \nvdash \gamma \qquad ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \quad ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2) \qquad (n_1 =\neq n_2) = 0}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 ! = e_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[nef])}\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)}$$

Figure 5.29: Multiparty SMC^2 semantics for public binary operations.

Public Declaration

$$\frac{(ty = \text{public } bty) \quad \text{acc} = 0 \quad l = \phi() \quad \gamma_1 = \gamma[x \rightarrow (l,\ ty)] \quad \omega = \text{EncodeVal}(ty, \text{NULL}) \quad \sigma_1 = \sigma[l \rightarrow (\omega,\ ty,\ 1,\ \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x) \parallel C)\ \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[dv])}\ ((\text{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \text{acc},\ \text{skip}) \parallel C)}$$

Private Declaration

$$\frac{((ty = bty) \vee (ty = \text{private } bty)) \wedge ((bty = \text{int}) \vee (bty = \text{float})) \quad l = \phi() \quad \omega = \text{EncodeVal}(ty, \text{NULL}) \quad \gamma_1 = \gamma[x \rightarrow (l, ty)] \quad \sigma_1 = \sigma[l \rightarrow (\omega, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{private}, 1))]}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x) \parallel C)\ \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[d1])}\ ((\text{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \text{acc},\ \text{skip}) \parallel C)}$$

Declaration Assignment

$$\frac{((\text{p},\ \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma_1, \sigma_1, \Delta_1, \text{acc}, \text{skip}) \parallel C_1) \quad ((\text{p}, \gamma_1, \sigma_1, \Delta_1, \text{acc}, x = e) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\text{p}, \gamma_1, \sigma_2, \Delta_2, \text{acc}, \text{skip}) \parallel C_2)}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x = e) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[ds])}\ ((\text{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_2)}$$

Read Public Variable

$$\frac{\gamma(x) = (l,\ \text{public } bty) \quad \sigma(l) = (\omega,\ \text{public } bty,\ 1,\ \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, 1)) \quad \text{DecodeVal}(\text{public } bty,\ \omega) = n}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x) \parallel C)\ \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[r])}\ ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ n) \parallel C)}$$

Read Private Variable

$$\frac{\gamma(x) = (l,\ \text{private } bty) \quad \sigma(l) = (\omega,\ \text{private } bty,\ 1,\ \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) \quad \text{DecodeVal}(\text{private } bty,\ \omega) = n}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x) \parallel C)\ \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[r1])}\ ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ n) \parallel C)}$$

Write Public Variable

$$\frac{(e) \nvdash \gamma \quad ((\text{p}, \gamma, \sigma, \Delta, \text{acc}, e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, n) \parallel C_1) \quad \gamma(x) = (l,\ \text{public } bty) \quad \text{UpdateVal}(\sigma_1,\ l,\ n,\ \text{public } bty) = \sigma_2}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x = e) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)])}_{\mathcal{D}_1::(\text{p},[w])}\ ((\text{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)}$$

Write Private Variable

$$\frac{(e) \vdash \gamma \quad ((\text{p}, \gamma, \sigma, \Delta, \text{acc}, e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, n) \parallel C_1) \quad \gamma(x) = (l,\ \text{private } bty) \quad \text{UpdateVal}(\sigma_1,\ l,\ n,\ \text{private } bty) = \sigma_2}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x = e) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)])}_{\mathcal{D}_1::(\text{p},[w1])}\ ((\text{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)}$$

Write Private Variable Public Value

$$\frac{(e) \nvdash \gamma \quad ((\text{p}, \gamma, \sigma, \Delta, \text{acc}, e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, n) \parallel C_1) \quad \gamma(x) = (l, \text{private } bty) \quad \text{UpdateVal}(\sigma_1, l, \text{encrypt}(n), \text{private } bty) = \sigma_2}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x = e) \parallel C)\ \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)])}_{\mathcal{D}_1::(\text{p},[w2])}\ ((\text{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)}$$

Parentheses

$$\frac{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ v) \parallel C_1)}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ (e)) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\text{p},[ep])}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ v) \parallel C_1)}$$

Statement Sequencing

$$\frac{((\text{p},\ \gamma,\ \sigma,\ \Delta,\ \text{acc},\ s_1) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma_1, \sigma_1, \Delta_1, \text{acc}, v_1) \parallel C_1) \quad ((\text{p}, \gamma_1, \sigma_1, \Delta_1, \text{acc}, s_2) \parallel C_1)\ \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}\ ((\text{p}, \gamma_2, \sigma_2, \Delta_2, \text{acc}, v_2) \parallel C_2)}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ s_1;\ s_2) \parallel C)\ \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[ss])}\ ((\text{p}, \gamma_2,\ \sigma_2,\ \Delta_2,\ \text{acc},\ v_2) \parallel C_2)}$$

Statement Block

$$\frac{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ s) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1}\ ((\text{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \text{acc},\ v) \parallel C_1)}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \{s\}) \parallel C)\ \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\text{p},[sb])}\ ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)}$$

Figure 5.30: Multiparty SMC^2 semantic rules for declarations, reading, writing, and sequencing.

SMC Input Public Value

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & \gamma(x) = (l, \text{public } bty) \\ \text{acc} = 0 & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, n) \parallel C_1) \\ \text{InputValue}(x, n) = n_1 & ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, x = n_1) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\text{p}, \gamma, \sigma_2, \Delta_2, \text{acc}, \text{skip}) \parallel C_2)\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{smcinput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[inp])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{skip}) \parallel C_2)}$$

SMC Input Private Value

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & \gamma(x) = (l, \text{private } bty) \\ \text{acc} = 0 & ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, n) \parallel C_1) \\ \text{InputValue}(x,\ n) = n_1 & ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, x = n_1) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\text{p}, \gamma, \sigma_2, \Delta_2, \text{acc}, \text{skip}) \parallel C_2)\end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{smcinput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[inp2])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{skip}) \parallel C_2)}$$

SMC Input Public 1D Array

$$\frac{\begin{array}{ll}(e_1, e_2) \nvdash \gamma & ((\text{p}, \gamma, \sigma, \Delta_1, \text{acc}, e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma, \sigma_1,\ \Delta_1,\ \text{acc},\ n) \parallel C_1) \\ \text{acc} = 0 & ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \alpha) \parallel C_2) \\ \gamma(x) = (l, \text{public const } bty*) & \text{InputArray}(x, n, \alpha) = [m_0, ..., m_\alpha] \\ ((\text{p}, \gamma, \sigma_2, \Delta_2, \text{acc}, x = [m_0, ..., m_\alpha]) \parallel C_2) \Downarrow^{\mathcal{L}_3}_{\mathcal{D}_3} ((\text{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \text{acc},\ \text{skip}) \parallel C_3) & \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{smcinput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\text{p},[inp1])} ((\text{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \text{acc},\ \text{skip}) \parallel C_3)}$$

SMC Input Private 1D Array

$$\frac{\begin{array}{ll}(e_1, e_2) \nvdash \gamma & ((\text{p}, \gamma, \sigma, \Delta, \text{acc}, e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, n) \parallel C_1) \\ \text{acc} = 0 & ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\text{p}, \gamma, \sigma_2, \Delta_2, \text{acc}, \alpha) \parallel C_2) \\ \gamma(x) = (l, \text{private const } bty*) & \text{InputArray}(x, n, \alpha) = [m_0, ..., m_\alpha] \\ ((\text{p}, \gamma, \sigma_2, \Delta_2, \text{acc}, x = [m_0, ..., m_\alpha]) \parallel C_2) \Downarrow^{\mathcal{L}_3}_{\mathcal{D}_3} ((\text{p}, \gamma, \sigma_3, \Delta_3, \text{acc}, \text{skip}) \parallel C_3) & \end{array}}{((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{smcinput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\text{p},[inp3])} ((\text{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \text{acc},\ \text{skip}) \parallel C_3)}$$

Figure 5.31: Multiparty SMC^2 semantic rules for input.

SMC Output Public Value

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1) \\ \gamma(x) = (l, \mathrm{public}\ bty) & \sigma_1(l) = (\omega,\ \mathrm{public}\ bty,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{public}\ bty, \mathrm{public}, 1)) \\ \mathrm{DecodeVal}(\mathrm{public}\ bty,\ \omega) = n_1 & \mathrm{OutputValue}(x, n, n_1)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[out])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)}$$

SMC Output Private Value

$$\frac{\begin{array}{ll}(e) \nvdash \gamma & ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1) \\ \gamma(x) = (l,\ \mathrm{private}\ bty) & \sigma_1(l) = (\omega,\ \mathrm{private}\ bty,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty, \mathrm{private}, 1)) \\ \mathrm{DecodeVal}(\mathrm{private}\ bty,\ \omega) = n_1 & \mathrm{OutputValue}(x,\ n, n_1)\end{array}}{((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[out2])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)}$$

SMC Output Public Array

$$\frac{\begin{array}{l}(e_1, e_2) \nvdash \gamma \qquad ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1) \\ \gamma(x) = (l, \mathrm{public\ const}\ bty*) \qquad ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, \alpha) \parallel C_2) \\ \sigma_2(l) = (\omega, \mathrm{public\ const}\ bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{public\ const}\ bty*, \mathrm{public}, 1)) \\ \mathrm{DecodePtr}(\mathrm{public\ const}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ \mathrm{public}\ bty,\ 1] \\ \sigma_2(l_1) = (\omega_1,\ \mathrm{public}\ bty,\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{public}\ bty, \mathrm{public}, \alpha)) \\ \forall i \in \{0, ..., \alpha - 1\} \quad \mathrm{DecodeArr}(\mathrm{public}\ bty,\ i,\ \omega_1) = m_i \\ \mathrm{OutputArray}(x,\ n,\ [m_0,\ ...,\ m_{\alpha-1}])\end{array}}{\begin{array}{l}((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[out1])} \\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)\end{array}}$$

SMC Output Private Array

$$\frac{\begin{array}{l}(e_1, e_2) \nvdash \gamma \qquad ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1) \\ \gamma(x) = (l, \mathrm{private\ const}\ bty*) \qquad ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, \alpha) \parallel C_2) \\ \sigma_2(l) = (\omega, \mathrm{private\ const}\ bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private\ const}\ bty*, \mathrm{private}, 1)) \\ \mathrm{DecodePtr}(\mathrm{private\ const}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ \mathrm{private}\ bty,\ 1] \\ \sigma_2(l_1) = (\omega_1,\ \mathrm{private}\ bty,\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty, \mathrm{private}, \alpha)) \\ \forall i \in \{0, ..., \alpha - 1\} \qquad \mathrm{DecodeArr}(\mathrm{private}\ bty,\ i,\ \omega_1) = m_i \\ \mathrm{OutputArray}(x,\ n,\ [m_0,\ ...,\ m_{\alpha-1}])\end{array}}{\begin{array}{l}((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[out3])} \\ ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)\end{array}}$$

Figure 5.32: Multiparty SMC^2 semantic rules for output.

5.1.3 Multiparty Vanilla C Algorithms

94 UpdateArr $::= f : (\widehat{\sigma},\ (\widehat{l}, \widehat{i}),\ \widehat{n},\ \widehat{bty}) \rightarrow \widehat{\sigma}$

95 EncodeArr $::= f : (\widehat{ty}, \widehat{n}, \widehat{n}) \rightarrow \widehat{\omega}$

96 DecodeArr $::= f : (\widehat{bty}, \widehat{i}, \widehat{\omega}) \rightarrow \widehat{n}$

Algorithm 94 $\widehat{\sigma}_2 \leftarrow \mathrm{UpdateArr}(\widehat{\sigma},\ (\widehat{l}, \widehat{i}),\ \widehat{v},\ \widehat{bty})$

1: $\widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{ty},\ \widehat{\alpha},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}, \mathrm{public}, \widehat{\alpha}))] = \widehat{\sigma}$
2: $\widehat{\mu} = \widehat{i} \cdot \mathrm{sizeof}(\widehat{bty})$
3: $\widehat{\omega}_1 = \widehat{\omega}[0 : \widehat{\mu}]$
4: $\widehat{\omega}_2 = \mathrm{EncodeVal}(\widehat{bty},\ \widehat{v})$
5: $\widehat{\omega}_3 = \widehat{\omega}[\widehat{\mu} + \widehat{\mu} :]$
6: $\widehat{\omega}_4 = \widehat{\omega}_1 :: \widehat{\omega}_2 :: \widehat{\omega}_3$
7: $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega}_4,\ \widehat{ty},\ \widehat{\alpha},\ \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}, \mathrm{public}, \widehat{\alpha}))]$
8: **return** $\widehat{\sigma}_2$

Algorithm 94 (UpdateArr) is used to update a value in memory at an index within an array. It takes as input memory $\widehat{\sigma}$, the location (memory block identifier and offset) and we will be updating, the value to store into memory, and the type to store the value as. Here, we first remove the mapping from memory (line 1), then find where the offset we will be updating will be within the array data (line 2). Next, we separate out the bytes before (line 3) and after (line 5) the data we will be replacing. We encode the new value based on the specified type (line 4), then combine these byte data to obtain the updated array byte data (line 6). We then place the new mapping with the updated data into memory (line 7) and return the updated memory. Here, we would like to highlight that we only update the portion of memory associated with the given offset (array index), which is public.

Algorithm 95 $\widehat{\omega} \leftarrow \mathrm{EncodeArr}(\widehat{ty}, \widehat{\alpha}, \widehat{v})$

$\widehat{\omega}_1 = \mathrm{EncodeVal}(\widehat{ty}, \widehat{v})$
$\widehat{\omega} = \widehat{\omega}_1$
for all $\widehat{i} \in \{1 ... \widehat{\alpha} - 1\}$ **do**
    $\widehat{\omega} = \widehat{\omega} + \widehat{\omega}_1$
end for
return $\widehat{\omega}$

Algorithm 95 (EncodeArr) takes an value and creates byte data for an array of length $\widehat{\alpha}$, with every element initialized to the value. It is currently only used in the semantics when declaring an array, to initialize the newly declared array as being filled with NULL elements. EncodeArr takes as input the type, number of elements, and the value to be used to initialize the array. It will encode the given value as byte data based on the type, and duplicate that $\widehat{\alpha}$ times to get the byte data for the entire array initialized with that value. This

full byte data is then returned.

Algorithm 96 $\widehat{v} \leftarrow \mathrm{DecodeArr}(\widehat{ty}, \widehat{i}, \widehat{\omega})$

1: $\widehat{\mu} = \widehat{i} \cdot \mathrm{sizeof}(\widehat{ty})$
2: $\widehat{\omega}_1 = \widehat{\omega}[\widehat{\mu} : \widehat{\mu} + \widehat{\mu}]$
3: $\widehat{v} = \mathrm{DecodeVal}(\widehat{ty}, \widehat{\omega}_1)$
4: **return** $\widehat{v}$

Algorithm 96 ($\mathrm{DecodeArr}$) takes byte data and returns the element of the given type at the specified index from the byte data. It takes as input a type, an index, and bytes of data for an array. It then finds the portion of bytes corresponding to that index, and calls Algorithm $\mathrm{DecodeVal}$ to obtain the value represented by those bytes. This value is then returned.

5.1.4 Multiparty Protocols for Multiparty SMC^2

97 $\mathrm{MPC}_{mult} ::= f : (n^{\mathrm{p}}, n^{\mathrm{p}}) \rightarrow n^{\mathrm{p}}$

98 $\mathrm{MPC}_{b} ::= f : (bop, [n^1, ..., n^{\mathrm{q}}], [n^1, ..., n^{\mathrm{q}}]) \rightarrow (n^1, ..., n^{\mathrm{q}})$

99 $\mathrm{MPC}_{cmp} ::= f : (bop, [n^1, ..., n^{\mathrm{q}}], [n^1, ..., n^{\mathrm{q}}]) = (n^1, ..., n^{\mathrm{q}})$

100 $\mathrm{MPC}_{u} ::= f : (++, n^1, ..., n^{\mathrm{q}}) \rightarrow (n^1, ..., n^{\mathrm{q}})$

101 $\mathrm{MPC}_{sub} ::= f : (n^{\mathrm{p}}, n^{\mathrm{p}}) \rightarrow n^{\mathrm{p}}$

102 $\mathrm{MPC}_{add} ::= f : (n^{\mathrm{p}}, n^{\mathrm{p}}) \rightarrow n^{\mathrm{p}}$

103 $\mathrm{MPC}_{div} ::= f : (n^{\mathrm{p}}, n^{\mathrm{p}}) \rightarrow n^{\mathrm{p}}$

104 $\mathrm{MPC}_{lt} ::= f : (n^{\mathrm{p}}, n^{\mathrm{p}}) \rightarrow n^{\mathrm{p}}$

105 $\mathrm{MPC}_{neq} ::= f : (n^{\mathrm{p}}, n^{\mathrm{p}}) \rightarrow n^{\mathrm{p}}$

106 $\mathrm{MPC}_{eq} ::= f : (n^{\mathrm{p}}, n^{\mathrm{p}}) \rightarrow n^{\mathrm{p}}$

107 $\mathrm{MPC}_{plpl} ::= f : (n^{\mathrm{p}}) \rightarrow n^{\mathrm{p}}$

108 $\mathrm{MPC}_{ar} ::= f : ((i^1, \overline{n}^1), ..., (i^{\mathrm{q}}, \overline{n}^{\mathrm{q}})) \rightarrow (n^1, ..., n^{\mathrm{q}})$

109 $\mathrm{MPC}_{aw} ::= f : ((i^1, n^1, \overline{n}^1), ..., (i^{\mathrm{q}}, n^{\mathrm{q}}, \overline{n}^{\mathrm{q}})) \rightarrow (\overline{n}^1, ..., \overline{n}^{\mathrm{q}})$

110 $\mathrm{MPC}_{dv} ::= f : ([\overline{n}^1, ..., \overline{n}^{\mathrm{q}}], [\overline{j}^1, ..., \overline{j}^{\mathrm{q}}]) \rightarrow (n^1, ..., n^{\mathrm{q}})$

111 $\mathrm{MPC}_{dp} ::= f : ([[[\alpha, \overline{l}^1, \overline{j}^1], ..., [\alpha, \overline{l}^1, \overline{j}^1]], ..., [[\alpha, \overline{l}^{\mathrm{q}}, \overline{j}^{\mathrm{q}}], ..., [\alpha, \overline{l}^{\mathrm{q}}, \overline{j}^{\mathrm{q}}]]], [\overline{j}^1, ..., \overline{j}^{\mathrm{q}}])$
$\rightarrow ([[\alpha, \overline{l}^1, \overline{j}^1], ..., [\alpha, \overline{l}^{\mathrm{q}}, \overline{j}^{\mathrm{q}}]])$

112 $\mathrm{MPC}_{wdv} ::= f : ([\overline{n}^1, ..., \overline{n}^{\mathrm{q}}], [n^1, ..., n^{\mathrm{q}}], [\overline{j}^1, ..., \overline{j}^{\mathrm{q}}]) \rightarrow (\overline{n}^1, ..., \overline{n}^{\mathrm{q}})$

113 $\mathrm{MPC}_{wdp} ::= f : ([\overline{ptr}^1, ..., \overline{ptr}^{\mathrm{q}}], [\overline{j}^1, ..., \overline{j}^{\mathrm{q}}]) \rightarrow [\overline{ptr}^1, ..., \overline{ptr}^{\mathrm{q}}]$

124 $\mathrm{MPC}_{resolve}$ $::= f : ([n^1, ..., n^q], [[(v^1, v^1), ..., (v^1, v^1)], ..., [(v^q, v^q), ..., (v^q, v^q)]]) \rightarrow [\overline{v}^1, ..., \overline{v}^q]$

128 MPC_{free} $::= f : ([\overline{\omega}^1, ..., \overline{\omega}^q], [\overline{j}^1, ...\overline{j}^q]) \rightarrow ([\overline{\omega}^1, ..., \overline{\omega}^q], [\overline{j}^1, ..., \overline{j}^q])$

Here, we present the multiparty protocols used in SMC^2. In our semantics, we leverage multiparty protocols to compartmentalize the complexity of handling private data. In the formal treatment this corresponds to using Axioms in our proofs to reason about protocols. These Axioms allow us to guarantee the desired properties of correctness and noninterference for the overall model, to provide easy integration with new, more efficient protocols as they become available, and to avoid re-proving the formal guarantees for the entire model when new protocols are added. Proving that these Axioms hold is a responsibility of the library implementor in order to have the system fully encompassed by our formal model. Secure multiparty computation protocols that already come with guarantees of correctness and security are the only ones worth considering, so the implementor would only need to ensure that these guarantees match our definitions of correctness and noninterference.

For example, if private values are represented using Shamir secret sharing [1], Algorithm 97, MPC_{mult}, represents a simple multiparty protocol for multiplying private values from [41]. In Algorithm 97, lines 2 and 3 define the protocol, while lines 1, 4, and 5 relate the protocol to our semantic representation.

Algorithm 97 $n_3^{\mathrm{p}} \leftarrow \mathrm{MPC}_{mult}(n_1^{\mathrm{p}}, n_2^{\mathrm{p}})$

1: Let $f_a(\mathrm{p}) = n_1^{\mathrm{p}}$ and $f_b(\mathrm{p}) = n_1^{\mathrm{p}}$.
2: Party p computes the value $f_a(\mathrm{p}) \cdot f_b(\mathrm{p})$ and creates its shares by choosing a random polynomial $h_{\mathrm{p}}(x)$ of degree t, such that $h_{\mathrm{p}}(0) = f_a(\mathrm{p}) \cdot f_b(\mathrm{p})$. Party p sends to each party i the value $h_{\mathrm{p}}(i)$.
3: After receiving shares from all other parties, party p computes their share of $a \cdot b$ as the linear combination $H(\mathrm{p}) = \sum_{i=1}^{\mathrm{q}} \lambda_i h_i(\mathrm{p})$.
4: Let $n_3^{\mathrm{p}} = H(\mathrm{p})$
5: **return** n_3^{p}

When computation is performed by q parties, at most t of whom may collude ($t < q/2$), Shamir secret sharing encodes a private integer a by choosing a polynomial $f(x)$ of degree t with random coefficients such that $f(0) = a$ (all computation takes place over a finite field). Each participant obtains evaluation of f on a unique non-zero point as their representation of private a; for example, party p obtains $f(\mathrm{p})$. This representation has the property that combining t or fewer shares reveals no information about a as all values of a are equally likely; however, possession of $t+1$ or more shares permits recovering of $f(x)$ via polynomial interpolation and thus learning $f(0) = a$.

In several of these Multiparty Algorithms, the outer loop (whose condition is $\mathrm{p} \in \{1...\mathrm{q}\}$) indicates that

the statements inside the loop would be run in parallel at each party. This notation facilitates showing that all parties are working together to compute the true value for each element that was modified within either branch.

Multiplication in Algorithm 97 corresponds to each party locally multiplying shares of inputs a and b, which computes the product, but raises the polynomial degree to $2t$. The parties consequently re-share their private intermediate results to lower the polynomial degree to t and re-randomize the shares. Values λ_{p} refer to interpolation coefficients which are derived from the computation setup and party p index.

In order to preserve the correctness and noninterference guarantees of our model when such an algorithm is added, a library developer will need to guarantee that the implementation of this algorithm is correct, meaning that it has the expected input output behavior, and it guarantees noninterference on what is observable.

Algorithm 98 $(n_3^1, ..., n_3^{\mathrm{q}}) \leftarrow \mathrm{MPC}_b(bop, [n_1^1, ..., n_1^{\mathrm{q}}], [n_2^1, ..., n_2^{\mathrm{q}}])$

```
for all p ∈ {1...q} do
    n_3^p = NULL
    if (bop = ·) then
        n_3^p = MPC_mult(n_1^p, n_2^p)
    else if (bop = ÷) then
        n_3^p = MPC_div(n_1^p, n_2^p)
    else if (bop = −) then
        n_3^p = MPC_sub(n_1^p, n_2^p)
    else if (bop = +) then
        n_3^p = MPC_add(n_1^p, n_2^p)
    end if
end for
return (n_3^1, ..., n_3^q)
```

Algorithm 99 $(n_3^1, ..., n_3^{\mathrm{q}}) \leftarrow \mathrm{MPC}_{cmp}([n_1^1, ..., n_1^{\mathrm{q}}], [n_2^1, ..., n_2^{\mathrm{q}}])$

```
for all p ∈ {1...q} do
    n_3^p = NULL
    if (bop = ==) then
        n_3^p = MPC_eq(n_1^p, n_2^p)
    else if (bop = ! =) then
        n_3^p = MPC_neq(n_1^p, n_2^p)
    else if (bop = <) then
        n_3^p = MPC_lt(n_1^p, n_2^p)
    end if
end for
return (n_3^1, ..., n_3^q)
```

Algorithm 98, MPC_b, is a selection control algorithm that directs the evaluation to the relevant multiparty computation algorithm based on the given binary operation $bop \in \{\cdot, \div, +, -\}$, and Algorithm 99, MPC_b, is a selection control algorithm that directs the evaluation to the relevant multiparty computation algorithm based on the given comparison operation $bop \in \{==, ! =, <\}$. Each of the given multiparty protocols

in Algorithm 98 (i.e., $\mathrm{MPC}_{mult}, \mathrm{MPC}_{sub}, \mathrm{MPC}_{add}, \mathrm{MPC}_{div}$) and each of the given multiparty protocols in Algorithm 99 (i.e., $\mathrm{MPC}_{eq}, \mathrm{MPC}_{neq}, \mathrm{MPC}_{lt}$) must be defined using protocols that have been proven to uphold the desired properties within our proofs (i.e., correctness and noninterference). We give an example definition for MPC_{mult} in Algorithm 97, but this definition can be swapped out with any protocol for the secure multiparty computation of multiplication that maintains the properties of correctness and noninterference. We defer the definition of all other SMC binary operations, rely on assertions that the protocols chosen to be used with this model will maintain both correctness and noninterference in our proofs. We chose this strategy as SMC implementations of such protocols will be proven to hold our desired properties on their own, and this allows us to not only leverage those proofs, but to also improve the versatility of our model by allowing such algorithms to be easily swapped out as newer, improved versions become available.

Algorithm 100 $(n_2^1, ..., n_2^{\mathrm{q}}) \leftarrow \mathrm{MPC}_u(uop, [n_1^1, ..., n_1^{\mathrm{q}}])$

```
1: if uop == ++ then
2:     for all p ∈ {1...q} do
3:         n_2^p = MPC_plpl(n_1^p)
4:     end for
5:     return (n_2^1, ..., n_2^q)
6: end if
```

Algorithm 100, MPC_u, is like MPC_b in that it is a selection control algorithm for multiparty unary operations. We only include the pre-increment operator here, as that is the only unary operation of this type that is within the scope of our current grammar (i.e., pointer dereferencing with $*$ is handled separately, and the address-of operator $\&$ is handled locally). Other types of operations that would be handled here are pre-decrement, post-increment and post-decrement of values, as well as negation. We chose not to include these elements in our grammar as they are trivial extensions of the current grammar.

The following Algorithms are given as placeholders for the SMC definitions of each function; for the model to be complete, these placeholder Algorithms would need to reflect the chosen implementations of each used within the system. Algorithm 101, MPC_{sub}, is for the SMC implementation of subtraction. This algorithm will securely compute whether $n_1^{\mathrm{p}} - n_2^{\mathrm{p}}$ for all parties p. Algorithm 102, MPC_{add}, is for the SMC implementation of addition. This algorithm will securely compute whether $n_1^{\mathrm{p}} + n_2^{\mathrm{p}}$ for all parties p. Algorithm 103, MPC_{div}, is for the SMC implementation of addition. This algorithm will securely compute whether $n_1^{\mathrm{p}} \div n_2^{\mathrm{p}}$ for all parties p. Algorithm 106, MPC_{eq}, is for the SMC implementation of equality. This algorithm will securely compute whether $n_1^{\mathrm{p}} == n_2^{\mathrm{p}}$ for all parties p. Algorithm 105, MPC_{neq}, is

for the SMC implementation of inequality. This algorithm will securely compute whether $n_1^{\mathrm{p}}! = n_2^{\mathrm{p}}$ for all parties p. Algorithm 104, MPC_{lt}, is for the SMC implementation of the less than operation. This algorithm will securely compute whether $n_1^{\mathrm{p}} < n_2^{\mathrm{p}}$ for all parties p. Algorithm 107, MPC_{plpl}, is for the SMC implementation of the pre-increment operation on a value. This algorithm will securely compute $n_1^{\mathrm{p}} + 1$ for all parties p.

Algorithm 101 $n_3^{\mathrm{p}} \leftarrow \mathrm{MPC}_{sub}(n_1^{\mathrm{p}}, n_2^{\mathrm{p}})$

Algorithm 102 $n_3^{\mathrm{p}} \leftarrow \mathrm{MPC}_{add}(n_1^{\mathrm{p}}, n_2^{\mathrm{p}})$

Algorithm 103 $n_3^{\mathrm{p}} \leftarrow \mathrm{MPC}_{div}(n_1^{\mathrm{p}}, n_2^{\mathrm{p}})$

Algorithm 104 $n_3^{\mathrm{p}} \leftarrow \mathrm{MPC}_{lt}(n_1^{\mathrm{p}}, n_2^{\mathrm{p}})$

Algorithm 105 $n_3^{\mathrm{p}} \leftarrow \mathrm{MPC}_{neq}(n_1^{\mathrm{p}}, n_2^{\mathrm{p}})$

Algorithm 106 $n_3^{\mathrm{p}} \leftarrow \mathrm{MPC}_{eq}(n_1^{\mathrm{p}}, n_2^{\mathrm{p}})$

Algorithm 107 $n_2^{\mathrm{p}} \leftarrow \mathrm{MPC}_{plpl}(n_1^{\mathrm{p}})$

The following Algorithms are given as placeholders for the SMC definitions of each function over arrays and pointers; for the model to be complete, these placeholder Algorithms would need to reflect the chosen implementations of each used within the system. We give a high level description for the idea behind each here.

Algorithm 108 $(n^1, ..., n^{\mathrm{q}}) \leftarrow \mathrm{MPC}_{ar}((i^1, [n_0^1, ..., n_{\alpha-1}^1]), ..., (i^{\mathrm{q}}, [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}]))$

Algorithm 108 (MPC_{ar}) is intended to privately read the value at the given private index within an array. It takes as input the array data and the private index from each party, and returns each party's resulting private value. One example of implementing this algorithm is as we had defined it in Basic SMC[2] ($v = \bigvee_{m=0}^{n-1}(i = \mathrm{encrypt}(m)) \wedge (v_m)$) – for each value in the array, we take the bitwise `and` of the value and the result of comparing the private index to the current index we are handling within the array, then taking the bitwise `or` of all resulting values.

Algorithm 109
$([n_0'^1, ..., n_{\alpha-1}'^1], ..., [n_0'^{\mathrm{q}}, ..., n_{\alpha-1}'^{\mathrm{q}}]) \leftarrow \mathrm{MPC}_{aw}((i^1, n^1, [n_0^1, ..., n_{\alpha-1}^1]), ..., (i^{\mathrm{q}}, n^{\mathrm{q}}, [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}]))$

Algorithm 109 (MPC_{aw}) is intended to privately replace the value at the given private index within the array with a new value. It takes as input the private index, the new value to place at the private

index, and the array data from each party, and returns each party's resulting array data. One example of implementing this algorithm is as we had defined it in Basic SMC^2 ($\forall n_m \in [n_0, ..., n_{\alpha-1}]$. $n'_m = ((i = \text{encrypt}(m)) \wedge n) \vee (\neg(n = \text{encrypt}(m)) \wedge n_m)$). For each value in the array, first find the private condition of whether the private index is equal to the current index we are evaluating. Then perform a bitwise `and` over the new value and this condition, and separately the logical negation of the private condition and the current value. Finally, we perform a bitwise `or` over the result of the two bitwise `and` operations, giving us the new value to store at this location.

Algorithm 110 $(n^1, ..., n^q) \leftarrow \text{MPC}_{dv}([[n_0^1, ..., n_{\alpha-1}^1], ..., [n_0^q, ..., n_{\alpha-1}^q]], [\bar{j}^1, ..., \bar{j}^q])$

Algorithm 110 (MPC_{dv}) is intended to privately evaluate which value is the true value when dereferencing a pointer with multiple locations. It takes as input the list of values obtained from dereferencing each possible location of the pointer and the pointer's tag list that indicates which location was the true location. One example of implementing this algorithm is how we defined it for Basic SMC^2, and is fairly similar to what we do when reading from a private index of an array . For each possible value, we take the bitwise `and` of the value and the tag for the location it came from, then taking the bitwise `or` of all resulting values ($n = \bigvee_{m=0}^{\alpha-1}(j_m) \wedge (n_m)$).

Algorithm 111
$([[\alpha, \bar{l}^1, \bar{j}^1], ..., [\alpha, \bar{l}^q, \bar{j}^q]]) \leftarrow \text{MPC}_{dp}([[[\alpha, \bar{l}^1, \bar{j}^1], ..., [\alpha, \bar{l}^1, \bar{j}^1]], ..., [[\alpha, \bar{l}^q, \bar{j}^q], ..., [\alpha, \bar{l}^q, \bar{j}^q]]], [\bar{j}^1, ..., \bar{j}^q])$

Algorithm 111 (MPC_{dp}) is intended to privately evaluate which location is the true value when dereferencing a pointer with multiple locations that is of a higher level of indirection. Here, we need to take all possible locations and condense them into a single location list and corresponding tag list. In the simplest case, where the higher level pointer has multiple locations but the dereference locations refer to only a single location, we can simply create a location list of all of these locations, and the tag list for the higher level pointer would correspond directly to this location list. However, it is possible that the lower level pointer locations can also refer to multiple locations, potentially with overlap between pointers. In this case, we need to take the location and tag lists for each of these pointers and privately combine them. The concept here is very similar to what occurs within the resolution of pointer locations at the end of a private-conditioned **`if else`** statement, and we can leverage the idea of Algorithm 78 here. This algorithm takes from each party a list of the number of locations, location list, and tag list for each of the possible locations of the original

pointer and the tag list of the original pointer as input, then returns each party's resulting number of locations, location list, and tag list.

Algorithm 112
$([n'^1_0, ..., n'^1_{\alpha-1}], ..., [n'^q_0, ..., n'^q_{\alpha-1}]) \leftarrow \mathrm{MPC}_{wdv}([[n^1_0, ..., n^1_{\alpha-1}], ..., [n^q_0, ..., n^q_{\alpha-1}]], [n^1, ..., n^q], [\bar{j}^1, ..., \bar{j}^q])$

Algorithm 112 (MPC_{wdv}) is intended to facilitate dereference writing a new value to a private pointer that has multiple locations. It takes as input from each party the list of dereferenced values from each possible location of the pointer, the new value to write into memory, and the list of tags for the pointer we are dereference writing to. One example of implementing this algorithm is how we defined it for Basic SMC^2, and is fairly similar to what we do when writing to a private index of an array ($\forall n_m \in [n_0, ..., n_{\alpha-1}].\ n'_m = (j_m \wedge n) \vee (\neg(j_m) \wedge n_m)$). For possible value, perform a bitwise `and` over the new value and the tag for the location it came from, and separately the logical negation of the the tag for the location it came from and the current value. Finally, we perform a bitwise `or` over the result of the two bitwise `and` operations, giving us the new value to store at this location.

Algorithm 113 $[\overline{ptr}^1, ..., \overline{ptr}^q] \leftarrow \mathrm{MPC}_{wdp}([\overline{ptr}^1, ..., \overline{ptr}^q], [\bar{j}^1, ..., \bar{j}^q])$

Algorithm 113 (MPC_{wdp}) is intended to facilitate dereference writing a new location list to a private pointer of a higher level of indirection that has multiple locations. It takes as input from each party the list of pointer data structures, the first being what needs to be written into memory and the rest from each possible location of the pointer, and also the list of tags for the pointer we are dereference writing to. The concept here is very similar to what occurs within the resolution of pointer locations at the end of a private-conditioned **`if else`** statement (i.e., separately combining the pointer data structure that needs to be stored into memory with the pointer data structures that are currently there), and we can leverage the idea of Algorithm 78 here. We return each party's updated pointer data structures to store back into memory for all possible locations of the pointer we are dereference writing to.

$\mathrm{MPC}_{resolve}$ and MPC_{free}, though multiparty algorithms, are diverted from this subsection to be shown in Algorithm 124 and 75, respectively, and discussed in conjunction with their corresponding algorithms.

5.1.5 Multiparty SMC^2 Algorithms

114	Extract	$::= f : (s, s, \gamma) = (\overline{x}, j)$
115	InitializeVariables	$::= f : (\overline{x}, \gamma, \sigma, n, \text{acc}) \rightarrow (\gamma, \sigma, \bar{l})$
116	RestoreVariables	$::= f : (\overline{x}, \gamma, \sigma, \text{acc}) \rightarrow (\sigma, \bar{l})$
117	ResolveVariables_Retrieve	$::= f : (\overline{x}, \text{acc}, \gamma, \sigma) \rightarrow ([(v, v), ..., (v, v)], n, \bar{l})$
118	ResolveVariables_Store	$::= f : (\overline{x}, \text{acc}, \gamma, \sigma, \overline{v}) \rightarrow (\sigma, \bar{l})$
119	Initialize	$::= f : (\Delta, \overline{x}, \gamma, \sigma, n, \text{acc}) \rightarrow (\gamma, \sigma, \Delta, \bar{l})$
120	DynamicUpdate	$::= f : (\Delta, \sigma, \bar{l},\ \text{acc}, ty) \rightarrow (\Delta, \bar{l})$
121	Restore	$::= f : (\sigma, \Delta, \text{acc}) \rightarrow (\sigma, \Delta, \bar{l})$
122	Resolve_Retrieve	$::= f : (\gamma, \sigma, \Delta, \text{acc}) \rightarrow ([(v, v), ..., (v, v)], n, \bar{l})$
123	Resolve_Store	$::= f : (\Delta, \sigma, \text{acc}, \overline{v}) \rightarrow (\sigma, \Delta, \bar{l})$
125	CondAssign	$::= f : ([\alpha, \bar{l}, \bar{j}], [\alpha, \bar{l}, \bar{j}], n) \rightarrow [\alpha, \bar{l}, \bar{j}]$
126	Free	$::= f : (\sigma,\ l) \rightarrow (\sigma, (l, \mu))$
127	CheckFreeable	$::= f : (\gamma, \bar{l}, \bar{j}, \sigma) \rightarrow j$
129	UpdatePointerLocations	$::= f : (\sigma, \bar{l},\ \bar{j}, (l, \mu), j) \rightarrow (\sigma, \bar{l})$
130	UpdateBytesFree	$::= f : (\sigma, \bar{l}, \overline{\omega}) \rightarrow \sigma$
131	UpdateArr	$::= f : (\sigma, (l, i), n, a\ bty) \rightarrow \sigma$
132	EncodeArr	$::= f : (a\ bty, \alpha, n) \rightarrow \omega$
133	DecodeArr	$::= f : (a\ bty, i, \omega) \rightarrow n$
134	UpdateDerefVals	$::= f : (\alpha, \bar{l}, \overline{v}, ty, \sigma) \rightarrow \sigma$
135	Retrieve_Values	$::= f : (\alpha,\ \bar{l},\ ty,\ \sigma) \rightarrow (\overline{v}, j)$
136	ReadOOB	$::= f : (i, \alpha, l, a\ bty, \sigma) \rightarrow (n, j, (l, \mu))$
137	WriteOOB	$::= f : (n, i, \alpha, l, a\ bty, \sigma, \Delta, \text{acc}) \rightarrow (\sigma, \Delta, j, (l, \mu))$
138	ϕ^{p}	$::= f : (\{temp\}) \rightarrow l$
139	$\mathcal{L} :: \mathcal{L}$	$::= f : \mathcal{L} :: \mathcal{L} \rightarrow \mathcal{L}$
140	$\mathcal{D} :: \mathcal{D}$	$::= f : \mathcal{D} :: \mathcal{D} \rightarrow \mathcal{D}$
141	$\mathcal{L}(\text{p})$	$::= f : \mathcal{L}(\text{p}) \rightarrow \mathcal{L}^{\text{p}}$
142	$\mathcal{D}(\text{p})$	$::= f : \mathcal{D}(\text{p}) \rightarrow \mathcal{D}^{\text{p}}$

143 $(\overline{e}) \vdash \gamma \quad ::= f : (\overline{e}) \vdash \gamma \rightarrow j$

144 $(\overline{e}) \nvdash \gamma \quad ::= f : (\overline{e}) \nvdash \gamma \rightarrow j$

Here, we will discuss the helper algorithms used when branching on private conditionals. First, we will discuss extraction of what variables are modified and how we choose which strategy to use. Second, we will discuss our variable-based tracking algorithms. Third, we will discuss our location-based tracking algorithms. Finally, we will discuss our multiparty resolution algorithms.

Algorithm 114 $(x_{mod}, j) \leftarrow \text{Extract}(s_1,\ s_2, \gamma)$

```
1: j = 0
2: x_local = [ ]
3: x_mod = [ ]
4: for all s ∈ {s_1; s_2} do
5:     if ((s = ty x) ∨ (s = ty x[e])) then
6:         x_local.append(x)
7:     else if ((s = x = e) ∧ (¬x_local.contains(x))) then
8:         x_mod = x_mod ∪ [x]
9:         for all e_1 ∈ e do
10:            if ((e_1 = ++ x_1) ∧ (¬x_local.contains(x_1))) then
11:                x_mod = x_mod ∪ [x_1]
12:            end if
13:        end for
14:    else if ((s = x[e_1] = e_2) ∧ (¬x_local.contains(x))) then
15:        if (e_1) ⊢ γ then
16:            x_mod = x_mod ∪ [x]
17:        else
18:            j = 1
19:        end if
20:        for all e ∈ {e_1, e_2} do
21:            if ((e = ++ x_1) ∧ (¬x_local.contains(x_1))) then
22:                x_mod = x_mod ∪ [x_1]
23:            end if
24:        end for
25:    else if ((s = ++ x) ∧ (¬x_local.contains(x))) then
26:        x_mod = x_mod ∪ [x]
27:    else if (s = *x = e) then
28:        j = 1
29:        for all e_1 ∈ e do
30:            if ((e_1 = ++ x_1) ∧ (¬x_local.contains(x_1))) then
31:                x_mod = x_mod ∪ [x_1]
32:            end if
33:        end for
34:    end if
35: end for
36: return (x_mod, j)
```

Algorithm 114, Extract, iterates over the statements contained in both branches, checking for which variables are modified (i.e., pre-increment operations, assignment statements) and whether either branch contains a pointer dereference write or array write at a public index. All variables that are modified through

pre-increment operations and regular assignment statements are added to the variable list that is returned. Pointer variables that are used in a pointer dereference write are not added to this list, as the data at the location that is referred to is being modified (and not the data stored at the pointer's location). This is also true for arrays that are only updated at public indices. When a pointer dereference write or array write at a public index is found, we update the tag to be 1 (i.e., true), otherwise the tag remains as 0. We later use this tag to decide whether we can proceed with the standard, flat basic block tracking using temporary variables (when no pointer dereference write operations or potential out of bounds writes occur), or whether we need to use the dynamic basic block tracking using locations.

It is important to note that if a pointer dereference write occurs inside a private-conditioned branch, we must proceed with location tracking at the level that it occurs as well as any outer levels of nesting of private-conditioned branches; however, if a lower level of nesting does not contain any pointer dereference writes, we can use variable tracking at that level. This algorithm will also filter out modifications made to any local variables, as we do not need to track and propagate those modifications outside of this local scope.

It is also important to note here that, currently, when we find an array has been modified as a whole, we simply add the array variable name to the list and track all locations. When an array has been modified at a private index, we add the entire array to be tracked, as we will be modifying all indices within the array to hide the true index that was updated. When an array has been modified at a specific public index, we trigger location tracking. We do this because we cannot easily tell what the value of the index will be at execution when we run $\mathrm{Extract}$ (unless the array index is hard-coded, but this is rare), and therefore we do not know if the array access will be in bounds or not.

Algorithms 115 ($\mathrm{InitializeVariables}$), 116 ($\mathrm{RestoreVariables}$), 117 ($\mathrm{ResolveVariables_Retrieve}$), and 118 ($\mathrm{ResolveVariables_Store}$) are specific to the variable tracking style of conditional code block tracking, as shown in rule Private If Else (Variable Tracking) in Figure 5.16.

First, Algorithm 115 stores the result of the conditional expression (n) in res_{acc} (lines 1:4). It grabs a new temporary variable location (line 1), adds the mapping to the environment (line 2), encodes the value n into its byte-representation (line 3), then adds the mapping into memory (line 4). It is important to note here that we pull new locations from the partition of memory designated for such temporary variables, as this simplifies the mapping of memory between SMC^2 and Vanilla C.

Then, for each variable x in x_{list}, we look up x in the environment and memory (line 7, 12), grab new temporary variable locations (lines 8-9), and create **`then`** and **`else`** temporary variables initialized with the

Algorithm 115 $(\gamma_1, \sigma_1, \bar{l}) \leftarrow \text{InitializeVariables}(x_{list},\ \gamma, \sigma, n, \text{acc})$

1: $l_{res} = \phi(temp)$
2: $\gamma_1 = \gamma[res_\text{acc} \rightarrow (l_{res}, \text{private int})]$
3: $\omega_{res} = \text{EncodeVal}(\text{private int}, n)$
4: $\sigma_1 = \sigma[l_{res} \rightarrow (\omega_{res}, \text{private int}, 1, \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$
5: $\bar{l} = [(l_{res}, 0)]$
6: **for all** $x \in x_{list}$ **do**
7: $\quad (l_x, ty) = \gamma(x)$
8: $\quad l_t = \phi(temp)$
9: $\quad l_e = \phi(temp)$
10: $\quad \bar{l} = \bar{l} :: [(l_x, 0), (l_t, 0), (l_e, 0)]$
11: $\quad \gamma_1 = \gamma_1[x_t_\text{acc} \rightarrow (l_t, ty)][x_e_\text{acc} \rightarrow (l_e, ty)]$
12: $\quad (\omega_x, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha)) = \sigma_1(l_x)$
13: $\quad$ **if** $(ty = \text{private const } bty*)$ **then**
14: $\quad\quad l_{ta} = \phi(temp)$
15: $\quad\quad l_{ea} = \phi(temp)$
16: $\quad\quad [1, [(l_{xa}, 0)], [1], 1] = \text{DecodePtr}(ty, 1, \omega_x)$
17: $\quad\quad (\omega_{xa}, \text{private } bty, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha)) = \sigma_1(l_{xa})$
18: $\quad\quad \sigma_1 = \sigma_1[l_{ta} \rightarrow (\omega_{xa}, \text{private } bty, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha))]$
19: $\quad\quad \sigma_1 = \sigma_1[l_{ea} \rightarrow (\omega_{xa}, \text{private } bty, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha))]$
20: $\quad\quad \omega_t = \text{EncodePtr}(ty, [1, [(l_t, 0)], [1], 1])$
21: $\quad\quad \omega_e = \text{EncodePtr}(ty, [1, [(l_e, 0)], [1], 1])$
22: $\quad\quad \sigma_1 = \sigma_1[l_t \rightarrow (\omega_t, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{private}, 1))]$
23: $\quad\quad \sigma_1 = \sigma_1[l_e \rightarrow (\omega_e, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{private}, 1))]$
24: $\quad\quad$ **for all** $i \in \{0...\alpha - 1\}$ **do**
25: $\quad\quad\quad \bar{l} = \bar{l} :: [(l_{xa}, i), (l_{ta}, i), (l_{ea}, i)]$
26: $\quad\quad$ **end for**
27: $\quad$ **else**
28: $\quad\quad \sigma_1 = \sigma_1[l_t \rightarrow (\omega_x, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha))]$
29: $\quad\quad \sigma_1 = \sigma_1[l_e \rightarrow (\omega_x, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha))]$
30: $\quad$ **end if**
31: **end for**
32: **return** $(\gamma_1, \sigma_1, \bar{l})$

value of x. To do this, we first add the mapping of these temporaries to the environment (line 11). When x refers to an entire array, we have the special case of needing to look up the array data from the pointer that refers to it. To handle this, we have split out the behavior for arrays within the **`if`** branch in the algorithm, and the **`else`** branch handles both pointers and regular variables appropriately, as these are single-level temporary variables.

If the variable is an array type, we must grab new temporary variable locations for the array data of the **`then`** and **`else`** variables (lines 14-15), look up the array data of x (lines 16-17), then add the mappings for both the array data (lines 18-19) and the array pointer (lines 20-23) to memory. For other types of variables, we can simply add the mappings for the **`then`** and **`else`** variables to memory directly using the data from x (lines 28-29), as the data that will be changed within the branches for these variables is at this level. Lines 5, 10, and 24-26 facilitate our analysis of which locations have been accessed or modified, which allows us to more easily reason about this within the rules as needed for our noninterference result.

Algorithm 116 $(\sigma_4, \bar{l}) \leftarrow \text{RestoreVariables}(x_{list}, \gamma, \sigma, \text{acc})$

1: $\bar{l} = [\,]$
2: **for all** $x \in x_{list}$ **do**
3: $\quad (l_x, ty) = \gamma(x)$
4: $\quad (l_t, ty) = \gamma(x_t_\text{acc})$
5: $\quad (l_e, ty) = \gamma(x_e_\text{acc})$
6: $\quad \bar{l} = \bar{l} :: [(l_x, 0), (l_t, 0), (l_e, 0)]$
7: $\quad$ **if** $(ty = \text{private const } bty*)$ **then**
8: $\quad\quad (\omega_{xa}, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{private}, 1)) = \sigma(l_x)$
9: $\quad\quad (\omega_{ta}, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{private}, 1)) = \sigma(l_t)$
10: $\quad\quad (\omega_{ea}, ty, 1, \text{PermL}(\text{Freeable}, ty, \text{private}, 1)) = \sigma(l_e)$
11: $\quad\quad [1, [(l_{xa}, 0)], [1], 1] = \text{DecodePtr}(ty, 1, \omega_{xa})$
12: $\quad\quad [1, [(l_{ta}, 0)], [1], 1] = \text{DecodePtr}(ty, 1, \omega_{ta})$
13: $\quad\quad [1, [(l_{ea}, 0)], [1], 1] = \text{DecodePtr}(ty, 1, \omega_{ea})$
14: $\quad\quad \sigma_1[l_{xa} \rightarrow (\omega_t, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha))] = \sigma$
15: $\quad\quad \sigma_2[l_{ta} \rightarrow (\omega_x, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha))] = \sigma_1$
16: $\quad\quad \sigma_3 = \sigma_2[l_{ta} \rightarrow (\omega_t, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha)]$
17: $\quad\quad (\omega_x, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha)) = \sigma_3(l_{ea})$
18: $\quad\quad \sigma_4 = \sigma_3[l_{xa} \rightarrow (\omega_x, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha)]$
19: $\quad\quad$ **for all** $i \in \{0...\alpha - 1\}$ **do**
20: $\quad\quad\quad \bar{l} = \bar{l} :: [(l_{xa}, i), (l_{ta}, i), (l_{ea}, i)]$
21: $\quad\quad$ **end for**
22: $\quad$ **else**
23: $\quad\quad \sigma_1[l_x \rightarrow (\omega_t, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha))] = \sigma$
24: $\quad\quad \sigma_2[l_t \rightarrow (\omega_x, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha)] = \sigma_1$
25: $\quad\quad \sigma_3 = \sigma_2[l_t \rightarrow (\omega_t, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha)]$
26: $\quad\quad (\omega_x, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha)) = \sigma_3(l_e)$
27: $\quad\quad \sigma_4 = \sigma_3[l_x \rightarrow (\omega_x, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha)]$
28: $\quad$ **end if**
29: $\quad \sigma = \sigma_4$
30: **end for**
31: **return** $(\sigma_4, \bar{l})$

In Algorithm 116, for each variable x within x_{list}, we must save the current value for the variable and then restore it to other value it had before execution of the **`then`** branch. We first look up x and its associated temporary variables within our environment. Then we proceed to restore based on the type (array vs. non-array). For arrays, we first find where the array data is stored (lines 8-13), then proceed to pull out the data for x and **`then`** from memory (lines 14-15). We then take the data that was in x, which is the resulting data from the **`then`** branch, and store it back into memory as the updated mapping for the **`then`** temporary (line 16). Finally, we look up the original data of x stored in the **`else`** temporary (line 17), and store it back into memory as the data for x (line 18).

It is useful to note that within this rule, we explicitly show that x currently contains the value for the else branch (ω_t), which we proceed to store in the **`then`** variable, and the **`else`** variable contains the original value for x (ω_x), which we proceed to store back into x. When an entire array has been modified, we have the special case of needing to look up the array data from the pointer that refers to it. To handle this, we have

split out the behavior for arrays within the **`if`** branch in the algorithm, and the **`else`** branch handles both pointers and regular variables appropriately, as these are single-level modifications (no pointer dereference writes occurred).

The behavior of lines 23-27 corresponds to the behavior of lines 14-18, but for variables that are not arrays. This is because the data that was modified for int, float, and pointer variables is at the first level lookup within memory, but for arrays it is stored at one level of indirection due to the structure of array variables as being a const pointer to the larger set of array data. In lines 6 and 19-21, we are facilitating the analysis of which locations have been accessed or modified, which allows us to more easily reason about this within the rules.

Algorithm 117 $(\overline{v}, n_{res}, \overline{l}) \leftarrow \text{ResolveVariables_Retrieve}(x_{list}, \text{acc}, \gamma, \sigma)$

1: $\overline{v} = [\,]$
2: $(l_{res}, \text{private int}) = \gamma(res_\text{acc})$
3: $(\omega_{res}, \text{private int}, 1, \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1)) = \sigma(l_{res})$
4: $n_{res} = \text{DecodeVal}(\text{private int}, \omega_{res})$
5: $\overline{l} = [(l_{res}, 0)]$
6: **for all** $x \in x_{list}$ **do**
7: $\quad (l_x, ty) = \gamma(x)$
8: $\quad (l_t, ty) = \gamma(x_t)$
9: $\quad (\omega_x, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha)) = \sigma(l_x)$
10: $\quad (\omega_t, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha)) = \sigma(l_t)$
11: $\quad \overline{l} = \overline{l} :: [(l_x, 0), (l_t, 0)]$
12: $\quad$ **if** $(ty = \text{private } bty)$ **then**
13: $\quad\quad v_x = \text{DecodeVal}(\text{private } bty, \omega_x)$
14: $\quad\quad v_t = \text{DecodeVal}(\text{private } bty, \omega_t)$
15: $\quad\quad \overline{v} = \overline{v}.append((v_t, v_x))$
16: $\quad$ **else if** $(ty = \text{private const } bty*)$ **then**
17: $\quad\quad [1, [(l_{xa}, 0)], [1], 1] = \text{DecodePtr}(ty, 1, \omega_x)$
18: $\quad\quad [1, [(l_{ta}, 0)], [1], 1] = \text{DecodePtr}(ty, 1, \omega_t)$
19: $\quad\quad (\omega_{xa}, \text{private } bty, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha)) = \sigma(l_{xa})$
20: $\quad\quad (\omega_{ta}, \text{private } bty, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha)) = \sigma(l_{ta})$
21: $\quad\quad$ **for all** $i \in \{0...\alpha - 1\}$ **do**
22: $\quad\quad\quad v_{xi} = \text{DecodeArr}(\text{private } bty, i, \omega_{xa})$
23: $\quad\quad\quad v_{ti} = \text{DecodeArr}(\text{private } bty, i, \omega_{ta})$
24: $\quad\quad\quad \overline{v} = \overline{v}.append((v_{ti}, v_{xi}))$
25: $\quad\quad\quad \overline{l} = \overline{l} :: [(l_{xa}, i), (l_{ta}, i)]$
26: $\quad\quad$ **end for**
27: $\quad$ **else if** $(ty = \text{private } bty*)$ **then**
28: $\quad\quad [\alpha, \overline{l}_x, \overline{j}_x, i] = \text{DecodePtr}(ty, \alpha, \omega_x)$
29: $\quad\quad [\alpha, \overline{l}_t, \overline{j}_t, i] = \text{DecodePtr}(ty, \alpha, \omega_t)$
30: $\quad\quad \overline{v} = \overline{v}.append(([\alpha, \overline{l}_t, \overline{j}_t, i], [\alpha, \overline{l}_x, \overline{j}_x, i]))$
31: $\quad$ **end if**
32: **end for**
33: **return** $(\overline{v}, n_{res}, \overline{l})$

In Algorithm 117, we retrieve all of the data needed to resolve what the true value for each modified variable x within x_{list} should be. First, we retrieve the value for the result of the conditional expression (lines

2-4), then we retrieve the values for each variable within x_{list}. We will retrieve the **`else`** value by looking up the value currently stored in x, as we have just completed execution of the **`else`** branch (lines 7, 9 and 13/17,19,22/28 by type). We will retrieve the **`then`** value by looking up the value currently stored in the temporary variable x_t (lines 8,10, and 14/18,20,23/29 by type). We append a tuple of the **`then`** and **`else`** values for each variable to the list of values (lines 15/24/30 by type). This list of values will then be used within the multiparty resolve algorithm $\mathrm{MPC}_{resolve}$ to obtain the true values for each variable. As with the previous helper algorithms, we collect a list of which locations we have accessed in order to facilitate our analysis of location accesses.

Algorithm 118 $(\sigma_1, \bar{l}) \leftarrow \mathrm{ResolveVariables_Store}(x_{list}, \mathrm{acc}, \gamma, \sigma, \overline{v})$

1: $\bar{l} = [\,]$
2: $\sigma_1 = \sigma$
3: **for all** $i \in \{0...|\overline{v}| - 1\}$ **do**
4: $\quad x = x_{list}[i]$
5: $\quad v_x = \overline{v}[i]$
6: $\quad (l_x, ty) = \gamma(x)$
7: $\quad \bar{l} = \bar{l}.append((l_x, 0))$
8: $\quad$ **if** $(ty = \text{private } bty)$ **then**
9: $\quad\quad \sigma_2 = \mathrm{UpdateVal}(\sigma_1,\ l_x,\ v_x,\ ty)$
10: $\quad\quad \sigma_1 = \sigma_2$
11: $\quad$ **else if** $(ty = \text{private const } bty*)$ **then**
12: $\quad\quad [1, [(l_{xa}, 0)], [1], 1] = \mathrm{DecodePtr}(ty, 1, \omega_x)$
13: $\quad\quad$ **for all** $\mu \in \{0...\alpha - 1\}$ **do**
14: $\quad\quad\quad v_\mu = v_x[\mu]$
15: $\quad\quad\quad \sigma_{2+\mu} = \mathrm{UpdateArr}(\sigma_{1+\mu},\ (l_{xa}, \mu),\ v_\mu,\ ty)$
16: $\quad\quad\quad \bar{l} = \bar{l}.append((l_{xa}, \mu))$
17: $\quad\quad$ **end for**
18: $\quad\quad \sigma_1 = \sigma_{2+\mu}$
19: $\quad$ **else if** $(ty = \text{private } bty*)$ **then**
20: $\quad\quad \sigma_2 = \mathrm{UpdatePtr}(\sigma_1,\ (l_x, 0),\ v_x,\ ty)$
21: $\quad\quad \sigma_1 = \sigma_2$
22: $\quad$ **end if**
23: **end for**
24: **return** $(\sigma_1, \bar{l})$

Once we have completed resolution of true values, we then use Algorithm 118 to store the true value for each modified variable x within x_{list} back into memory. The list of values maintains its ordering during resolution, so we simply iterate through the list of variables and values, updating each variable with its corresponding value. As with the previous helper algorithms, we collect a list of which locations we have accessed in order to facilitate our analysis of location accesses.

Algorithms 119 (Initialize), 120 (DynamicUpdate), 121 (Restore), 122 (Resolve_Retrieve), and 123 (Resolve_Store) are specific to the location tracking style of conditional code block tracking, as shown in rule Private If Else (Location Tracking) in Figure 5.16.

Algorithm 119 $(\gamma_1, \sigma_1, \Delta_1, \bar{l}_1) \leftarrow \text{Initialize}(\Delta, x_{list},\ \gamma,\ \sigma,\ \text{acc})$

1: $l_{res} = \phi(temp)$
2: $\gamma_1 = \gamma[res_\text{acc} \rightarrow (l_{res}, \text{private int})]$
3: $\omega_{res} = \text{EncodeVal}(\text{private int}, n)$
4: $\sigma_1 = \sigma[l_{res} \rightarrow (\omega_{res}, \text{private int}, 1, \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))]$
5: $\bar{l}_1 = [(l_{res}, 0)]$
6: **for all** $x \in x_{list}$ **do**
7: $\quad (l, ty) = \gamma(x)$
8: $\quad \bar{l}_1 = \bar{l}_1.append((l, 0))$
9: $\quad$ **if** $(ty == \text{private } bty)$ **then**
10: $\quad\quad (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) = \sigma(l)$
11: $\quad\quad v = \text{DecodeVal}(\text{private } bty, \omega)$
12: $\quad\quad \Delta_1 = \Delta[\text{acc}].push(((l, 0) \rightarrow (v, \text{NULL}, 0, \text{private } bty)))$
13: $\quad$ **else if** $(ty == \text{private } bty*)$ **then**
14: $\quad\quad (\omega,\ \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) = \sigma(l)$
15: $\quad\quad [\alpha,\ \bar{l},\ \bar{j},\ i] = \text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega)]$
16: $\quad\quad \Delta_1 = \Delta[\text{acc}].push(((l, 0) \rightarrow ([\alpha,\ \bar{l},\ \bar{j},\ i], \text{NULL}, 0, \text{private } bty*)))$
17: $\quad$ **else if** $(ty = \text{private const } bty*)$ **then**
18: $\quad\quad (\omega,\ \text{private const } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1)) = \sigma(l)$
19: $\quad\quad [1,\ [(l_1, 0)],\ [1],\ 1] = \text{DecodePtr}(\text{private const } bty*,\ 1,\ \omega)]$
20: $\quad\quad (\omega_1, \text{private } bty, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha)) = \sigma_2(l_1)$
21: $\quad\quad$ **for all** $i \in \{0...\alpha - 1\}$ **do**
22: $\quad\quad\quad \bar{l}_1 = \bar{l}_1.append((l_1, i))$
23: $\quad\quad\quad v_i = \text{DecodeArr}(\text{private } bty, i, \omega_1)$
24: $\quad\quad\quad \Delta_1 = \Delta_1[\text{acc}].push(((l_1, i) \rightarrow (v_i, \text{NULL}, 0, \text{private } bty)))$
25: $\quad\quad$ **end for**
26: $\quad$ **end if**
27: $\quad \Delta = \Delta_1$
28: **end for**
29: **return** $(\gamma_1, \sigma_1, \Delta_1, \bar{l}_1)$

It is worthwhile to start by noting the structure of Δ for SMC^2. Δ is a list of lists, with the inner lists storing the mapping of location to data for dynamic tracking at each level of nesting of private-conditioned branches. Each mapping is structured as $(l, \mu) \rightarrow (v_{orig}, v_{then}, j, ty)$, where (l, μ) is the location that is modified (stored as the memory block identifier and offset into the block), v_{orig} is the original data stored in a location, and v_{then} as the data stored in that location at the end of the execution of the **`then`** branch. The public tag j is set to 0 when a new mapping is added to Δ, signifying that we have stored data into v_{orig}, but there is currently no data in v_{then}. During restoration between branches, we update j to 1 as we store the data from that location into the map. This is needed due to dynamic tracking of pointer dereference writes and potential out of bounds array accesses - we can see such a modification to a untracked location for the first time in the **`else`** branch, and this allows us to add these new locations without needing to track which branch we are currently in (i.e., for the current level of nesting and all outer levels, as this may be a new location for all levels). Using this tag, we are able to resolve at all levels of nesting with ease, using the tag to indicate whether we should use v_{orig} or v_{then} as the **`then`** data in resolution. This tag does not need to be private, as

it is visible to an observer whether or not the data at a given location was modified during the execution of either branch.

Algorithm 119, $\mathrm{Initialize}$, stores the result of the conditional and then iterates through all of the variables within the variable list obtained from $\mathrm{Extract}$, adding a new mapping for the location at which it is stored and storing its current value into the $orig$ portion, as well as initializing the tag j for that location as 0. As the modification of the location where a variable is stored is not allowed, it is safe to add all of these locations and their original values into Δ before the execution of the **`then`** branch. This allows execution of the **`then`** branch to proceed as normal, only incurring additional costs when a pointer dereference write or an array write at a public index occurs. In this Algorithm, we have built in the tracking of which locations we are accessing, adding them to the variable $\bar{l}_1$ and then returning this information to the rule from which it was called.

Algorithm 120 $(\Delta_1, \bar{l}_1) \leftarrow \mathrm{DynamicUpdate}(\Delta, \sigma,\ \bar{l}, \mathrm{acc}, ty)$

1: **if** $(\mathrm{acc} = 0)$ **then**
2: **return** $(\Delta, [\,])$
3: **end if**
4: $\bar{l}_1 = [\,]$
5: $\Delta_1 = \Delta$
6: **for** $(l, \mu) \in \bar{l}$ **do**
7: **if** $((l, \mu) \notin \Delta_1[\mathrm{acc}])$ **then**
8: $\bar{l}_1 = \bar{l}_1 :: [(l, \mu)]$
9: $\sigma_1[l \rightarrow (\omega,\ ty',\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable}, ty', \mathrm{private}, \alpha))] = \sigma$
10: **if** $(ty = ty' = \mathrm{private}\ bty) \wedge (0 \leq \mu < \alpha)$ **then**
11: $v = \mathrm{DecodeArr}(ty,\ \mu, \omega)$
12: $\Delta_1 = \Delta_1[\mathrm{acc}].push(((l, \mu) \rightarrow (v,\ \mathrm{NULL},\ 0, \mathrm{private}\ bty)))$
13: **else if** $(ty = ty' = \mathrm{private}\ bty*) \wedge (\mu = 0)$ **then**
14: $[\alpha, \bar{l}_1, \bar{j}, i] = \mathrm{DecodePtr}(\mathrm{private}\ bty*, \alpha, \omega)$
15: $\Delta_1 = \Delta_1[\mathrm{acc}].push(((l, 0) \rightarrow ([\alpha, \bar{l}_1, \bar{j}, i], \mathrm{NULL}, 0, \mathrm{private}\ bty*)))$
16: **else**
17: $v = \mathrm{GetBytes}((l, \mu), ty, \sigma)$
18: $\Delta_1 = \Delta_1[\mathrm{acc}].push(((l, \mu) \rightarrow (v,\ \mathrm{NULL},\ 0, ty)))$
19: **end if**
20: **if** $(\mathrm{acc} > 0)$ **then**
21: $\Delta_1 = \mathrm{DynamicUpdate}(\Delta_1, \sigma,\ [(l, \mu)], \mathrm{acc} - 1)$
22: **end if**
23: **end if**
24: **end for**
25: **return** $(\Delta_1, \bar{l}_1)$

Algorithm 120, $\mathrm{DynamicUpdate}$, is used prior to performing a pointer dereference write, an array write at a public index, and within Algorithm $\mathrm{WriteOOB}$ in order to ensure that we are *correctly* tracking *all* locations that get modified. It takes the location that is about to be modified and ensures that this location is either already being tracked by Δ for the current level of nesting, or adds the location and its original value to

Δ for that level of nesting. If this location is not already in Δ for the current level of nesting, and we are not in the outer-most private-conditioned branch, it will recursively call itself for all outer levels of nesting. This is to ensure that the location will be properly tracked at all levels. If the location is found to already be tracked at an outer level, it will return. We chose to perform this more costly checking at this point of execution, as we know whether or not the location is new to this level of nesting at this point, and can easily propagate this information upward to the outer levels of nesting here. The most costly check, where this new location needs to be added to all outer levels of nesting, can only occur once for each new location and will only occur once as subsequent modifications will find that the location is already being tracked. This propagation must happen at some point during execution, and would only require additional memory resources if it is not performed at this point, as, in order to put off the propagation until later, it would be necessary to tag this location as one that had been added during this level of nesting. Algorithm $\mathrm{GetBytes}$ is used within this Algorithm when the location that is given and what is stored at that location do not match up perfectly, such as the case when the given type and the type at that location do not match (in which case, we would need to grab additional bytes from the next location in order to properly decode a value based on our expected type.

It is important to reiterate here that during an out of bounds array write, $\mathrm{DynamicUpdate}$ is called from within $\mathrm{WriteOOB}$ in order to properly track the location being written to, since we are overshooting the location containing the array data. For pointer dereference writes, we use this to ensure we are tracking the most current location referred to by the pointer, since it is possible that it has changed during execution of either branch. For array writes at public indices, we must track dynamically (whether out of bounds or in bounds) due to the possibility of an out of bounds access.

Algorithm 121 $(\sigma_2, \Delta_3, \bar{l}) \leftarrow \text{Restore}(\sigma,\ \Delta,\ \text{acc})$

```
1: Δ1 = Δ
2: l̄ = [ ]
3: for all ((l, μ) → (v_orig, NULL, 0, ty)) ∈ Δ[acc] do
4:     v_then = NULL
5:     if (μ = 0) then
6:         if (ty = private bty) then
7:             σ1[l → (ω, private bty, 1, PermL(Freeable, private bty, private, 1))] = σ
8:             ω1 = EncodeVal(private bty, v_orig)
9:             σ2 = σ1[l → (ω1, private bty, 1, PermL(Freeable, private bty, private, 1))]
10:            v_then = DecodeVal(private bty, ω)
11:        else if (ty = private bty∗) then
12:            [α_orig, l̄_orig, j̄_orig, i] = v_orig
13:            σ1[l → (ω_then, private bty∗, α_then, PermL(Freeable, private bty∗, private, α_then))] = σ
14:            ω_orig = EncodePtr(ty, [α_orig, l̄_orig, j̄_orig, i])
15:            σ2 = σ1[l → (ω_orig, private bty∗, α_orig, PermL(Freeable, private bty, private, α_orig))]
16:            v_then = DecodePtr(private bty∗, α_then, ω_then)
17:        end if
18:    else
19:        v_then = GetBytes((l, μ), ty, σ)
20:        σ2 = SetBytes((l, μ), ty, v_orig, σ)
21:    end if
22:    Δ2[acc][(l, 0) → (v_orig, NULL, 0, ty)] = Δ1
23:    Δ3 = Δ2[acc][(l, 0) → (v_orig, v_then, 1, ty)]
24:    l̄ = l̄.append(l, μ)
25:    σ = σ2
26:    Δ1 = Δ3
27: end for
28: return (σ2, Δ3, l̄)
```

Algorithm 121, Restore, iterates through all locations in Δ at the current level of nesting acc, storing the current data for each location into the $then$ portion of the mapping for the given location, and restoring the original data to the location from the $orig$ portion. Additionally, it will update the tag j to 1 for all locations. This allows Resolve to know whether a new location was added during the execution of the **`else`** branch, and to use the value stored in $orig$ when such a location is found.

Algorithm 122 $(\overline{v}, n_{res}, \overline{l}) \leftarrow$ Resolve_Retrieve$(\gamma, \sigma,\ \Delta,\ \text{acc})$

1: $\overline{v} = [\,]$
2: $(l_{res}, \text{private int}) = \gamma(res_\text{acc})$
3: $(\omega_{res}, \text{private int}, 1, \text{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1)) = \sigma(l_{res})$
4: $n_{res} = \text{DecodeVal}(\text{private int}, \omega_{res})$
5: $\overline{l} = [(l_{res}, 0)]$
6: **for all** $((l, \mu) \rightarrow (v_{orig}, v_{then}, j, ty)) \in \Delta[\text{acc}]$ **do**
7: $\quad v_t = \text{NULL}$
8: $\quad$ **if** $j = 0$ **then**
9: $\quad\quad v_t = v_{orig}$
10: $\quad$ **else**
11: $\quad\quad v_t = v_{then}$
12: $\quad$ **end if**
13: $\quad v_e = \text{NULL}$
14: $\quad$ **if** $(\mu = 0)$ **then**
15: $\quad\quad (\omega, ty, \alpha, \text{PermL}(\text{Freeable}, ty, \text{private}, \alpha)) = \sigma(l)$
16: $\quad\quad$ **if** $(ty = \text{private } bty)$ **then**
17: $\quad\quad\quad (\omega, \text{private } bty, 1, \text{PermL}(\text{Freeable}, \text{private } bty, \text{private}, 1)) = \sigma(l)$
18: $\quad\quad\quad v_e = \text{DecodeVal}(\text{private } bty, \omega)$
19: $\quad\quad$ **else if** $(ty = \text{private } bty*)$ **then**
20: $\quad\quad\quad (\omega, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha)) = \sigma(l)$
21: $\quad\quad\quad v_e = \text{DecodePtr}(\text{private } bty*, \alpha, \omega)$
22: $\quad\quad$ **end if**
23: $\quad$ **else**
24: $\quad\quad v_e = \text{GetBytes}((l, \mu), ty, \sigma)$
25: $\quad$ **end if**
26: $\quad \overline{v} = \overline{v}.append(v_t, v_e)$
27: $\quad \overline{l} = \overline{l}.append((l, \mu))$
28: **end for**
29: **return** $(\overline{v}, n_{res}, \overline{l})$

Algorithm 122, Resolve_Retrieve, returns the result of the conditional, a list of tuples of the **`then`** and **`else`** values for each location in $\Delta[\text{acc}]$, and a list of locations that it has accessed. To get values for each branch, it iterates through all the locations in $\Delta[\text{acc}]$. To get the **`then`** value, it uses the tag indicating whether that location was modified in the **`then`** branch or not; if it is 0, it will use the stored original value from before execution of either branch, if it is 1, it will use the stored then value. The data currently stored in each location is used for the **`else`** data, as execution of the **`else`** branch has just completed.

Algorithm 123, Resolve_Store, stores all the true values back into memory. It iterates through all the locations in $\Delta[\text{acc}]$, encoding the values as their expected type and writing this byte representation into memory. Once all the changes have been stored into memory, it removes the list of mappings for this level of private-conditioned branches using $\Delta.pop()$, as it becomes out of scope once we leave this algorithm and the Private If Else rule. This algorithm takes as input the location map, memory, the accumulator, and the list of values to be stored back into memory (one for every location in the location map). When complete, it returns the updated memory and location map, as well as the list of locations that were modified within this

Algorithm 123 $(\sigma_1, \Delta_1, \bar{l}) \leftarrow \text{Resolve_Store}(\Delta,\ \sigma,\ \text{acc},\ \overline{v})$

```
1: l̄ = [ ]
2: σ1 = σ
3: for all i ∈ {0...|v̄| − 1} do
4:     v_f = v̄[i]
5:     ((l, μ) → (v_orig, v_then, j, ty)) = Δ[acc][i]
6:     if (μ = 0) then
7:         if (ty = private bty) then
8:             (ω, ty, α, PermL(Freeable, ty, private, α)) = σ1(l)
9:             if (α = 1) then
10:                σ1 = UpdateVal(σ, l, v_f, ty)
11:            else
12:                σ1 = UpdateArr(σ, (l, 0), v_f, ty)
13:            end if
14:        else if (ty = private bty∗) then
15:            σ1 = UpdatePtr(σ, (l, 0), v_f, ty)
16:        end if
17:    else
18:        σ1 = SetBytes((l, μ), ty, v_f, σ)
19:    end if
20:    l̄ = l̄.append((l, μ))
21:    σ = σ1
22: end for
23: Δ1 = Δ.pop()
24: return (σ1, Δ1, s, l̄)
```

algorithm.

Algorithm 124, ($\text{MPC}_{resolve}$), is the multiparty algorithm for facilitating the secure resolution of the values of which branch are the true values. We have already read the elements from memory, so each tuple within the parties value list $\overline{v}^{\text{p}}$ is either a pointer data structure or an int (or float) value. We proceed to find the true value based upon what type of value we are currently viewing, leveraging Algorithm 125 to compute the final pointer data structure for each pointer.

Algorithm 125 (CondAssign) is an algorithm that requires multiparty computation. Due to the complexity of this algorithm, that it is always called by each party within a different multiparty algorithm, and that it directly calls specific multiparty protocols that give the behavior for a single party, we show the behavior as it would occur at a single party. CondAssign takes two pointer data structures with the associated number of locations, lists of locations, and lists of tags as well as a flag n_{res}. Its primary purpose is to merge two pointer data structures during the execution of conditional statements with private conditions. Here, n_{res} is a flag that indicates whether the true pointer location should be taken from the first or the second data structure; $n_{res} == 1$ means that the true location is in the first data structure. For example, when executing code `if (priv) p1 = p2;`, n_{res} is the result of evaluating private condition `priv`, the first data structure corresponds to `p1`'s data structure prior to executing this statement, and the second data structure corresponds

Algorithm 124 $(\overline{v}^1_f, ..., \overline{v}^{\mathrm{q}}_f) \leftarrow \mathrm{MPC}_{resolve}([n^1_{res}, ..., n^{\mathrm{q}}_{res}], [\overline{v}^1, ..., \overline{v}^{\mathrm{q}}])$

```
for all p ∈ {1...q} do
    v̄^p_f = [ ]
    for all i ∈ {0...|v̄^p| − 1} do
        (v^p_t, v^p_e) = v̄^p[i]
        v^p_f = NULL
        if ([α_t, l̄_t, j̄_t, i] = v^p_t) then
            [α^p_e, l̄^p_e, j̄^p_e, i] = v^p_e
            [α^p_f, l̄^p_f, j̄^p_f] = CondAssign([α^p_t, l̄^p_t, j̄^p_t], [α^p_e, l̄^p_e, j̄^p_e], n^p_res)
            v^p_f = [α^p_f, l̄^p_f, j̄^p_f, i]
        else
            v^p_f = MPC_add(MPC_mult(n^p_res, v^p_t), MPC_mult(MPC_sub(1, n^p_res), v^p_e))
        end if
        v̄^p_f.append(v^p_f)
    end for
end for
return (v̄^1_f, ..., v̄^q_f)
```

Algorithm 125 $[\alpha_3,\ \bar{l}_3,\ \bar{j}_3] \leftarrow \mathrm{CondAssign}([\alpha_1,\ \bar{l}_1,\ \bar{j}_1],\ [\alpha_2,\ \bar{l}_2,\ \bar{j}_2],\ n_{res})$

```
l̄_3 = l̄_1 ∪ l̄_2
α_3 = |l̄_3|
j̄_3 = [ ]
for all (l_m, μ_m) ∈ l̄_3 do
    pos_1 = l̄_1.find((l_m, μ_m))
    pos_2 = l̄_2.find((l_m, μ_m))
    if (pos_1 ∧ pos_2) then
        j''_m = MPC_add(MPC_mult(n_res, j'_pos2), MPC_mult(MPC_sub(1, n_res), j_pos1))
    else if (¬ pos_2) then
        j''_m = MPC_mult(MPC_sub(n_res), j_pos1)
    else
        j''_m = MPC_mult(n_res, j'_pos2)
    end if
    j̄_3.append(j''_m)
end for
return [α_3, l̄_3, j̄_3]
```

to p2's data structure. The function first computes the union of the two lists of locations and then updates their corresponding tags based on their tags at the time of calling this function and the value of n_{res}. For example, if a particular location l_m is found on both lists, we retain its tag from the first list if n_{res} is set and otherwise retain its tag from the second list if n_{res} is not set. When l_m is found only in one of the lists, we use a similar logic and conditionally retain its original tag based on the value of n_{res}. If a tag is not retained, it is reset to 0. This ensures that for any pointer data structure only one tag is set to 1 and all others are set to 0.

Algorithm 126 $\sigma_3 \leftarrow \mathrm{Free}(\sigma_1,\ l)$

```
σ_2[l → (ω, a bty, 1, PermL(Freeable, a bty, a, 1))] = σ_1
σ_3 = σ_2[l → (ω, a bty, 1, PermL(None, a bty, a, 1))]
return (σ_3, (l, 0))
```

Algorithm 126 corresponds to conventional memory deallocation when we call free to deallocate memory associated with some pointer. We simply set the permissions for this location to be None, indicating that it has been freed and is no longer intended to be in use.

Algorithm 127 $j \leftarrow \mathrm{CheckFreeable}(\gamma, \bar{l}, \bar{j}, \sigma)$

```
if (l_default, 0) ∈ l̄ then
    return 0
end if
for all (l_m, μ_m) ∈ l̄ do
    if μ_m ≠ 0 then
        return 0
    end if
end for
if 1 ∉ j̄ then
    return 0
end if
for all x ∈ γ do
    (l_x, ty_x) = γ(x)
    if (l_x, 0) ∈ l̄ then
        return 0
    else if ty_x = a const bty* then
        (ω, ty_x, 1, PermL(Freeable, ty_x, a, 1)) = σ(l_x)
        [1, [(l_1, 0)], [1], 1] = DecodePtr(ty_x, 1, ω)
        if (l_1, 0) ∈ l̄ then
            return 0
        end if
    end if
end for
return 1
```

Algorithm 127 ($\mathrm{CheckFreeable}$) and ensures the behavior expected of free: if the location was properly allocated via a call to malloc, it is deallocatable for the purpose of this function. In particular, the default location $l_{default}$ that corresponds to uninitialized pointers is not deallocatable (and freeing such a pointer has no effect); similarly memory associated with statically declared variables is not de-allocatable via this mechanism (and freeing it here also has no effect). Thus, if $\mathrm{CheckFreeable}$ returns 1, we will proceed to mark location l as unavailable within the rules this is called from, otherwise the freeing rules have no effect on the state of memory.

Algorithm 128 (MPC_{free}) corresponds to deallocating memory associated with a pointer to private data which may be associated with multiple locations where the data may actually reside. The true location is not publicly known and the location to be removed should be chosen based on public knowledge. For the purposes of this functionality, and without loss of generality, we deallocate the first location on the list, l_0. Deallocation of l_0 requires additional work because that location might not be the true location, and may still be validly in use by other pointers. In other words, based on the fact that freeing a pointer has been called, we

Algorithm 128

$([[\omega'^{1}_0, ..., \omega'^{1}_n], ..., [\omega'^{\mathrm{q}}_0, ..., \omega'^{\mathrm{q}}_n]], [\bar{j}'^{1}, ..., \bar{j}'^{\mathrm{q}}]) \leftarrow \mathrm{MPC}_{free}([[\omega^{1}_0, ..., \omega^{1}_n], ..., [\omega^{\mathrm{q}}_0, ..., \omega^{\mathrm{q}}_n]], [\bar{j}^{1}, ..., \bar{j}^{\mathrm{q}}])$

```
for all p ∈ {1...q} do
    ω'^p_0 = ω^p_0
    [j^p_0, ..., j^p_n] = j̄^p
    j'^p_0 = j^p_0
    for all m ∈ {1...n} do
        ω'^p_m = MPC_add(MPC_mult(ω^p_m, MPC_sub(1, j^p_m)), MPC_mult(ω^p_0, j^p_m))
        ω'^p_0 = MPC_add(MPC_mult(ω'^p_0, MPC_sub(1, j^p_m)), MPC_mult(ω^p_m, j^p_m))
        j'^p_0 = MPC_add(j'^p_0, j^p_m)
    end for
    j̄'^p = [j'^p_0, j^p_1, ..., j^p_n]
end for
return ([[ω'^1_0, ..., ω'^1_n], ..., [ω'^q_0, ..., ω'^q_n]], [j̄'^1, ..., j̄'^q])
```

know that the true location can be released, but it might not be safe to deallocate other locations associated with the pointer.

For that reason, in Algorithm 128 we iterate through all locations l_1 through $l_{\alpha-1}$ and swap the content of the current location l_m and l_0 if l_m is in fact the true location (i.e., flag j_m is set). That is, ω'_m corresponds to the updated content of location l_m: the content will remain unchanged if j_m is not set, and otherwise, it will be replaced with the content of location l_0. Similarly, ω'_0 corresponds to the updated content of location l_0. Note that it may be modified in at most one iteration of the loop, namely, when j_m is set. All other iterations will keep the value unchanged (and it will never be modified if none of the tags $j_1, \ldots, j_{\alpha-1}$ are set and j_0 is). The function is written to be data-oblivious, i.e., to not reveal the true location associated with the pointer. We additionally compute an update to the tag for l_0, ensuring that if it was swapped with another location, we will have two tags set to 1 to indicate the two locations whose data we swapped. This algorithm then returns the updated set of bytes and tag list with the updated first tag.

In Algorithm 129 ($\mathrm{UpdatePointerLocations}$), we are given location l_r which is being removed and a list of other locations $\bar{l}$ associated with the pointer in question. In the event that l_r was not the true pointer location, its content has been moved to another location, but it still may remain in the lists of other pointers, which is what this function is to correct. In particular, the function iterates through other pointers in the system and searches for location l_r in their lists. If l_r is present (i.e., $l_r \in \bar{l}_k$), we need to remove it and replace it with another location from $\bar{l}$ to which the data has been moved. However, because we do not know which location in $\bar{l}$ is set and contains the relevant data, we are left with merging all locations in L with the pointer's current locations $\bar{l}'_k$ after removing l_r. This is done using Algorithm 125, $\mathrm{CondAssign}$.

Notice that we are also merging two pointer data structures based on a condition. This time the condition

Algorithm 129 $(\sigma_1, \bar{l}_1) \leftarrow \text{UpdatePointerLocations}(\sigma, \bar{l}, \bar{j}, (l_r, \mu_r), j_r)$

```
1: σ1 = [ ]
2: l̄1 = [ ]
3: for all lk ∈ σ do
4:     (ωk, ty, n, PermL(Freeable, ty, a, n)) = σ(lk)
5:     if (ty = private btyp*) then
6:         l̄1 = l̄1 :: [(lk, 0)]
7:         [n, l̄k, j̄k, i] = DecodePtr(private btyp*, n, ω)
8:         if (lr, μr) ∈ l̄k then
9:             pos = l̄k.find((lr, μr))
10:            j̄'k = j̄k \ j̄k[pos]
11:            l̄'k = l̄k \ (lr, μr)
12:            [αnew, l̄new, j̄new] = CondAssign([|l̄|, l̄, j̄], [n − 1, l̄'k, j̄'k], j̄k[pos])
13:            ω'k = EncodePtr(private btyp*, [αnew, l̄new, j̄new, i])
14:            σ1 = σ1[lk → (ω'k, ty, n, PermL(Freeable, ty, a, n))]
15:        else
16:            σ1 = σ1[lk → (ωk, ty, n, PermL(Freeable, ty, a, n))]
17:        end if
18:    else
19:        σ1 = σ1[lk → (ωk, ty, n, PermL(Freeable, ty, a, n))]
20:    end if
21: end for
22: return (σ1, l̄1)
```

is $j_{k_{pos}}$, which indicates whether the true location is in the first or second list of locations. That is, if l_r was the true location of the pointer, the data has been moved and resides in one of the locations in $\bar{l}$. Otherwise, if l_r was not the true location, the data resides at one of the remaining locations associated with the pointer on its location list $\bar{l}'_k$. Thus, we merge the list of locations and update the corresponding tags in the same way this is done during evaluation of conditional statements with private conditions.

Algorithm 130 $\sigma_2 \leftarrow \text{UpdateBytesFree}(\sigma,\ [(l_0, 0), ..., (l_n, 0)],\ [\omega_0, ..., \omega_n])$

```
1: σ1[l0 → (ω'0, ty, α, PermL(Freeable, ty, private, α))] = σ
2: σ2 = σ1[l0 → (ω0, ty, α, PermL(Freeable, ty, private, α))]
3: for all m ∈ {1...n} do
4:     σ3[lm → (ω'm, ty, αm, PermL(Freeable, ty, private, αm))] = σ2
5:     σ4 = σ3[lm → (ωm, ty, αm, PermL(Freeable, ty, private, αm))]
6:     σ2 = σ4
7: end for
8: return σ2
```

Algorithm 130 (UpdateBytesFree) if the final step of the rule for `pfree` when the pointer has multiple locations. Here, we are modifying the permissions of the first location l_0 to be None, indicating that this location has been freed, and storing the updated set of bytes for this location into memory. We then iterate through all other locations in the list, storing their modified byte representations into memory. Once this is complete, we will have completed the swap of data if l_0 was not the true location. Otherwise, we are simply writing the original data into memory again.

Algorithm 131 $\sigma_2 \leftarrow \mathrm{UpdateArr}(\sigma,\ (l, i),\ v,\ a\ bty)$

1: $\sigma_1[l \rightarrow (\omega,\ ty,\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable}, ty, a, \alpha))] = \sigma$
2: $\mu = i \cdot \mathrm{sizeof}(a\ bty)$
3: $\omega_1 = \omega[0 : \mu]$
4: $\omega_2 = \mathrm{EncodeVal}(a\ bty,\ v)$
5: $\omega_3 = \omega[\mu + \mu :]$
6: $\omega_4 = \omega_1 :: \omega_2 :: \omega_3$
7: $\sigma_2 = \sigma_1[l \rightarrow (\omega_4,\ ty,\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable}, ty, a, \alpha))]$
8: **return** σ_2

Algorithm 131 ($\mathrm{UpdateArr}$) is used to update a value in memory at an index within an array. It takes as input memory σ, the location (memory block identifier and offset) and we will be updating, the value to store into memory, and the type to store the value as. Here, we first remove the mapping from memory (line 1), then find where the offset we will be updating will be within the array data (line 2). Next, we separate out the bytes before (line 3) and after (line 5) the data we will be replacing. We encode the new value based on the specified type (line 4), then combine these byte data to obtain the updated array byte data (line 6). We then place the new mapping with the updated data into memory (line 7) and return the updated memory. Here, we would like to highlight that we only update the portion of memory associated with the given offset (array index), which is public.

Now, we will present the algorithms used for encoding and decoding bytes in memory for arrays in our semantics. Again, it is important to note that we leave the specifics of encoding to bytes and decoding from bytes up to the implementation, as this low-level function may vary based on the system and underlying architecture.

Algorithm 132 $\omega \leftarrow \mathrm{EncodeArr}(ty, \alpha, v)$

$\omega_v = \mathrm{EncodeVal}(ty, v)$
$\omega = \omega_v$
for all $i \in \{1...\alpha - 1\}$ **do**
    $\omega = \omega + \omega_v$
end for
return ω

Algorithm 132 ($\mathrm{EncodeArr}$) takes an value and creates byte data for an array of length α, with every element initialized to the value. It is currently only used in the semantics when declaring an array, to initialize the newly declared array as being filled with NULL elements. $\mathrm{EncodeArr}$ takes as input the type, number of elements, and the value to be used to initialize the array. It will encode the given value as byte data based on the type, and duplicate that α times to get the byte data for the entire array initialized with that value. This full byte data is then returned.

Algorithm 133 $v \leftarrow \mathrm{DecodeArr}(ty, i, \omega)$

1: $\mu = i \cdot \mathrm{sizeof}(ty)$
2: $\omega_1 = \omega[\mu : \mu + \mu]$
3: $v = \mathrm{DecodeVal}(ty, \omega_1)$
4: **return** v

Algorithm 133 (DecodeArr) takes byte data and returns the element of the given type at the specified index from the byte data. It takes as input a type, an index, and bytes of data for an array. It then finds the portion of bytes corresponding to that index, and calls Algorithm DecodeVal to obtain the value represented by those bytes. This value is then returned.

Algorithm 134 $\sigma_f \leftarrow \mathrm{UpdateDerefVals}(\alpha, \bar{l}, \overline{v}, ty, \sigma)$

$\sigma_f = \sigma$
for all $(m \in \{0...\alpha - 1\})$ **do**
  $(l, \mu) = \bar{l}[m]$
  $v = \overline{v}[m]$
  $\sigma_f = \mathrm{SetBytes}((l, \mu), ty, v, \sigma_f)$
end for
return σ_f

Algorithm 134 (UpdateDerefVals) is designed to store values of the specified type back into memory for every location within the location list. It is used within the multiparty private pointer dereference write rules to store all of the possible values for the pointer back into memory after we have used the multiparty protocol to find the replacement for the value at every location. With this, the value at the the true location, when decrypted, is the only value that is different from what was originally stored there; however, given that we need to hide the true location, the multiparty protocol obtained new values for all locations and now we must update each in memory. This algorithm takes the number of locations, the location list, the value list, the expected type, and the memory as input, and returns the updated memory. Given that we have already assessed whether any of the locations are not *well-aligned* and we have already handled ensuring that each of these locations are tracked if we are inside a private-conditioned branch, we simplify this algorithm and use Algorithm 65 (SetBytes) to store the values back into memory at their corresponding location.

Algorithm 135 (Retrieve_Values) is designed to obtain values of the specified type from every location within the location list. It is used within the multiparty private pointer dereference read and write rules to obtain all of the possible values for the pointer before we use the multiparty protocol to find the true value (i.e., during a dereference read) or replace the value at the true location (i.e., during a dereference write). It takes as input the number of locations, the list of locations, the expected type of value to obtain from the

Algorithm 135 $(\overline{v}, j) \leftarrow \text{Retrieve_Values}(\alpha,\ \bar{l},\ ty,\ \sigma)$

```
v̄ = [ ]
j = 1
for all ((l, μ) ∈ l̄) do
    v1 = NULL
    (ω1, ty1, α1, PermL(Freeable, ty1, a1, α1) = σ(l1)
    if (μ = 0) ∧ (ty1 = ty) ∧ (α1 = 1) then
        if (ty = private bty) then
            v1 = DecodeVal(private bty, 1, ω1)
        else if (ty = private bty∗) then
            v1 = DecodePtr(private bty∗, 1, ω1)
        end if
    else if (μ < α1) ∧ (ty1 = ty) ∧ (ty = private bty) then
        v1 = DecodeArr(private bty, μ, ω1)
    else
        v1 = GetBytes((l, μ), ty, σ)
        j = 0
    end if
    v̄ = v̄.append(v1)
end for
return (v̄, j)
```

locations, and memory. When complete, it returns the final list of values corresponding to the locations and the final tag indicating whether all values were obtained from *well-aligned* memory locations or not.

Algorithm 136 (ReadOOB) is designed to read a value of the given type from memory as though it was at index i of the array in memory block l. It takes as input the out of bounds index i, the number of values in the array n, the memory block of the array data l, the type of elements in the array $a\ bty$, and memory σ. It then iterates through memory until it finds the bytes that would be at index i and decodes them as the expected type bty to obtain value v. It is important to note here that index i will be public, as we do not overshoot the bounds of an array when we have a private index. As the algorithm iterates through memory, if all locations we iterate over are of the same type as the expected type, and the location we are reading the value from is also the expected type, then it will return tag 1, indicating that our read was *well-aligned*. Otherwise, tag 0 will be returned. We currently only show the algorithm handling overshooting in the positive direction, however, it can trivially extended to grab the previous blocks and iterate backwards through memory to handle a negative index as well. This algorithm is nearly identical to Algorithm 57, with the addition of returning the location (memory block and offset) from which we start performing the out-of-bounds read. To do this, we initialize the location as NULL, then, the first time we begin to read bytes to form our value, we set it with the current location and offset we are beginning to read from. We highlight the added functionality using green text.

Algorithm 137 (WriteOOB) is designed to store a value of the given type from memory as though it was

Algorithm 136 $(v,\ j, (l_f, \mu_f)) \leftarrow \text{ReadOOB}(i,\ n,\ l,\ a\ bty,\ \sigma)$

```
1: n_v = τ(a bty)
2: n_b = (i − n) · n_v
3: j = 1
4: ω_v = [ ]
5: (l_f, μ_f) = (NULL, NULL)
6: while (n_b > 0) ∨ (n_v > 0) do
7:     l = GetBlock(l)
8:     (ω, ty_1, α, PermL(Freeable, ty_1, a_1, α)) = σ(l)
9:     if (ty_1 ≠ a bty) then
10:        j = 0
11:    end if
12:    if (n_b < τ(ty_1) · α) then
13:        if (l_f, μ_f) = (NULL, NULL) then
14:            (l_f, μ_f) = (l, n_b)
15:        end if
16:        ω_v = ω_v :: ω[n_b : min(n_b + n_v, τ(ty_1) · α − 1)]
17:        n_v = n_v − τ(ty_1) · α + n_b
18:    end if
19:    n_b = max(0, n_b − τ(ty_1) · α)
20: end while
21: v = DecodeVal(a bty, 1, ω_v)
22: return (v, j, (l_f, μ_f))
```

at index i of the array in memory block l. It takes as input the value to write in memory v, the out of bounds index i, the number of values in the array n, the memory block of the array data l, the type of elements in the array $a\ bty$, and memory σ. It then iterates through memory until it finds the position that would be for index i, encodes value v as the expected type, and places its byte representation into memory starting at that position. It is important to note here that index i will be public, as we do not overshoot the bounds of an array when we have a private index. As the algorithm iterates through memory, if all locations we iterate over are of the same type as the expected type, and the location we are writing the value to is also the expected type, then it will return tag 1, indicating that our read was *well-aligned*. Otherwise, tag 0 will be returned. We currently only show the algorithm handling overshooting in the positive direction, however, it can trivially extended to grab the previous blocks and iterate backwards through memory to handle a negative index as well. This algorithm is nearly identical to Algorithm 58, with the addition of returning the location (memory block and offset) from which we start performing the out-of-bounds write and ensuring that if we are within a private-conditioned branch, we add the locations we are modifying and their original values to location map Δ if they are not already tracked. We highlight the added functionality using green text.

Algorithm 138, (ϕ) defines how new memory block identifiers are obtained - each party will have a counter that is monotonically increasing after each time ϕ is called, and a *temp* counter that is monotonically decreasing after each time $\phi(temp)$ is called. The *temp* argument is optional, and it signifies when the *temp*

Algorithm 137 $(\sigma,\ \Delta_f, j, (l_f, \mu_f)) \leftarrow \text{WriteOOB}(v,\ i,\ n,\ l,\ a\ bty,\ \sigma, \Delta, \text{acc})$

$\omega_v = \text{EncodeVal}(a\ bty, v)$
$n_b = (i - n) \cdot \tau(a\ bty)$
$j = 1$
$(l_f, \mu_f) = (\text{NULL}, \text{NULL})$
while $(n_b > 0) \vee (|\omega_v| > 0)$ **do**
  $l = \text{GetBlock}(l)$
  $\sigma_1[l \rightarrow (\omega,\ ty_1,\ \alpha,\ \text{PermL}(\text{Freeable}, ty_1, a_1, \alpha))] = \sigma$
  if $(ty_1 \neq a\ bty)$ **then**
    $j = 0$
  end if
  if $(n_b < \tau(ty_1) \cdot \alpha)$ **then**
    if $(|\omega_v| > \tau(ty_1) \cdot \alpha - n_b)$ **then**
      $\omega_1 = \omega[0 : n_b] + \omega_v + \omega[|\omega_v| + n_b :]$
      $\omega_v = [\,]$
    else if $(|\omega_v| = \tau(ty_1) \cdot \alpha - n_b)$ **then**
      $\omega_1 = \omega[0 : n_b] + \omega_v$
      $\omega_v = [\,]$
    else
      $\omega_1 = \omega[0 : n_b] + \omega_v[0 : \tau(ty_1) \cdot \alpha - n_b - 1]$
      $\omega_v = \omega_v[\tau(ty_1) \cdot \alpha - n_b :]$
    end if
    if $((l_f, \mu_f) = (\text{NULL}, \text{NULL}))$ **then**
      $(l_f, \mu_f) = (l, n_b)$
    end if
    $(\Delta_f, [(l_1, \mu_1)]) = \text{DynamicUpdate}(\Delta, \sigma,\ [(l, n_b)], \text{acc}, a\ bty)$
    $\sigma = \sigma_1[l \rightarrow (\omega_1,\ ty_1,\ \alpha,\ \text{PermL}(\text{Freeable}, ty_1, a_1, \alpha))]$
  end if
  $n_b = max(0, n_b - \tau(ty_1) \cdot \alpha)$
end while
return $(\sigma, \Delta_f, j, (l_f, \mu_f))$

counter is to be used – that is, only during the allocation of temporary variables used within the Private If Else rules. We separate these elements into their own partition of memory in order to easily show correctness of the memory with regards to Vanilla C- it is possible to provide a more robust mapping scheme between locations in Vanilla C and locations in SMC^2, but this extension provides unnecessary complexity for our proofs.

Algorithm 139 illustrates how two locations-touched data structures are merged. This merging maintains the ordering of which locations were touched and how many times for each party.

Algorithm 140 illustrates how two evaluation code data structures are merged. This merging maintains the ordering of when the evaluation was completed in respect to other evaluations completed by each party (i.e., a local party evaluation ordering, not a total ordering). When a multiparty evaluation code is entered (i.e., $\mathcal{D}_2 == (\text{ALL}, d)$), we iterate through and add the code to the evaluation code lists of each of the parties.

Algorithm 141 illustrates the filtering of the locations touched in the memory of a single party from the overall data structure showing the locations touched by each party in their respective memories.

Algorithm 138 $l \leftarrow \phi^{\mathrm{p}}(\{temp\})$

1: $next = l_{default}$
2: **if** $temp$ **then**
3: $\quad next = \mathrm{p}_global_location_temp_counter - -$
4: **else**
5: $\quad next = \mathrm{p}_global_location_counter\ ++$
6: **end if**
7: **return** l_{next}

Algorithm 139 $\mathcal{L}_3 \leftarrow \mathcal{L}_1 :: \mathcal{L}_2$

1: $\mathcal{L}_3 = \mathcal{L}_1$
2: **for all** $(\mathrm{p}, \bar{l}) \in \mathcal{L}_2$ **do**
3: $\quad$ **if** $(\mathcal{L}_3 = (\mathrm{p}, \bar{l}_1) \parallel \mathcal{L}_4)$ **then**
4: $\quad\quad \mathcal{L}_3 = (\mathrm{p}, \bar{l}_1 :: \bar{l}) \parallel \mathcal{L}_4$
5: $\quad$ **else**
6: $\quad\quad \mathcal{L}_3 = (\mathrm{p}, \bar{l}) \parallel \mathcal{L}_3$
7: $\quad$ **end if**
8: **end for**
9: **return** $\mathcal{L}_3$

Algorithm 142 illustrates the filtering of the evaluation codes executed by a single party from the overall data structure showing the evaluation codes executed by each party, respectively.

Algorithm 143 illustrates how $(\overline{e}) \vdash \gamma$ is evaluated, finding if there is at least one private element in the expression list $\overline{e}$. As we iterate through each expression in $\overline{e}$, if we find an expression that is private, $(\overline{e}) \vdash \gamma$ holds as true. Otherwise, if we have evaluated all expressions and found none are private, we return false. In this case, as we show in Algorithm 144, $(\overline{e}) \nvdash \gamma$ holds as true, because all elements are public.

Algorithm 140 $\mathcal{D}_3 \leftarrow \mathcal{D}_1 :: \mathcal{D}_2$

if $((\mathrm{ALL}, [d]) == \mathcal{D}_2)$ **then**
  $\mathcal{D}_3 = \epsilon$
  for all $(\mathrm{p}, \overline{d}) \in \mathcal{D}_1$ **do**
    $\mathcal{D}_3 = (\mathrm{p}, \overline{d} :: d) \parallel \mathcal{D}_3$
  end for
else
  $\mathcal{D}_3 = \mathcal{D}_1$
  for all $(\mathrm{p}, \overline{d}) \in \mathcal{D}_2$ **do**
    if $(\mathcal{D}_3 = (\mathrm{p}, \overline{d}_1) \parallel \mathcal{D}_4)$ **then**
      $\mathcal{D}_3 = (\mathrm{p}, \overline{d}_1 :: \overline{d}) \parallel \mathcal{D}_4$
    else
      $\mathcal{D}_3 = (\mathrm{p}, \overline{d}) \parallel \mathcal{D}_3$
    end if
  end for
end if
return $\mathcal{D}_3$

Algorithm 141 $\mathcal{L}^{\mathrm{p}} \leftarrow \mathcal{L}(\mathrm{p})$

$\mathcal{L}^{\mathrm{p}} = \epsilon$
if $(((\mathrm{p}, \overline{l}) \parallel \mathcal{L}_1) == \mathcal{L})$ **then**
  $\mathcal{L}^{\mathrm{p}} = (\mathrm{p}, \overline{l})$
end if
return $\mathcal{L}^{\mathrm{p}}$

Algorithm 142 $\mathcal{D}^{\mathrm{p}} \leftarrow \mathcal{D}(\mathrm{p})$

$\mathcal{D}^{\mathrm{p}} = \epsilon$
if $(((\mathrm{p}, \overline{d}) \parallel \mathcal{D}_1) == \mathcal{D})$ **then**
  $\mathcal{D}^{\mathrm{p}} = (\mathrm{p}, \overline{d})$
end if
return $\mathcal{D}^{\mathrm{p}}$

Algorithm 143 $j \leftarrow (\overline{e}) \vdash \gamma$

```
for all e ∈ ē do
    if (e = x(ē)) ∧ ((l, ty̅ → ty) = γ(x)) then
        if ((ty = public bty∗) ∨ (ty = public bty)) then
            return 0
        end if
    else if (e = uop var) ∨ (e = var) then
        if (var = x) ∧ ((l, ty) = γ(x)) then
            if (ty = public bty∗) ∨ (ty = public bty) then
                return 0
            end if
        else if (var = x[e1]) ∧ ((l, ty) = γ(x)) then
            if (ty = public const bty∗) ∧ ((e1) ⊬ γ) then
                return 0
            end if
        end if
    else if (e = e1 bop e2) ∧ ((e1, e2) ⊬ γ) then
        return 0
    else if (e = (e1)) ∧ ((e1) ⊬ γ) then
        return 0
    else if (e = (ty) e1) then
        if (ty = public bty) ∨ (ty == public bty∗) ∨ ((e1) ⊬ γ) then
            return 0
        end if
    else if (e = v) then
        if (e = [v0, ..., vn]) ∧ ((v0, ..., vn) ⊬ γ) then
            return 0
        else if (e = n) ∧ (n ≠ encrypt(n)) then
            return 0
        end if
    else
        return 0
    end if
end for
return 1
```

Algorithm 144 $j \leftarrow (\overline{e}) \nvdash \gamma$

```
if ((ē) ⊢ γ) then
    return 0
else
    return 1
end if
```

5.2 Correctness

In our semantics, we give each evaluation an identifying code as a shorthand way to refer to that specific evaluation, as well as to allow us to quickly reason about the Multiparty Vanilla C and Multiparty SMC^2 evaluations that are congruent to each other (i.e., a Multiparty Vanilla C rule and an identical one handling only public data in Multiparty SMC^2).

The list of Multiparty Vanilla C codes are as follows: $MVanC$ = [mpb, $mpcmpt$, $mpcmpf$, $mppin$, $mpra$, $mpwe$, $mpfre$, $mpiet$, $mpief$, $mprdp$, $mprdp1$, $mpwdp$, $mpwdp1$, fls, ss, sb, ep, cv, cl, r, w, ds, dv, dp, da, wle, wlc, bp, bs, bm, bd, ltf, ltt, eqf, eqt, nef, net, mal, fre, wp, wdp, $wdp1$, rp, rdp, $rdp1$, ra, wa, rao, wao, rae, wae, loc, iet, ief, inp, $inp1$, out, $out1$, df, ty, fd, fpd, fc, pin, $pin1$, $pin2$].

The list of Multiparty SMC^2 codes are as follows: $MSmcC$ = [mpb, $mpcmp$, $mpra$, $mpwa$, $mppin$, $mpdp$, $mpdph$, $mpfre$, $mprdp$, $mprdp1$, $mpwdp$, $mpwdp1$, $mpwdp2$, $mpwdp3$, iet, ief, iep, $iepd$, wle, wlc, dp, $dp1$, rp, $rp1$, rdp, $rdp1$, $rdp2$, wp, $wp1$, $wp2$, wdp, $wdp1$, $wdp2$, $wdp3$, $wdp4$, $wdp5$, da, $da1$, das, ra, $ra1$, rea, wa, $wa1$, $wa2$, wea, $wea1$, $wea2$, rao, $rao1$, wao, $wao1$, $wao2$, pin, $pin1$, $pin2$, $pin3$, $pin4$, $pin5$, $pin6$, $pin7$, mal, $malp$, fre, $pfre$, cv, $cv1$, cl, $cl1$, loc, ty, df, fd, fpd, fc, $fc1$, bp, bs, bm, bd, ltf, ltt, eqf, eqt, nef, net, dv, $d1$, r, $r1$, w, $w1$, $w2$, ds, ss, sb, ep, inp, $inp1$, $inp2$, $inp3$, out, $out1$, $out2$, out].

The list of Multiparty Vanilla C codes that would lead to differences with a Multiparty SMC^2 evaluation are as follows: $MVanCX$ = [$rao'*$, $wao'*$, $pin2'*$, $pin3'*$] The list of Multiparty SMC^2 codes that would lead to differences with a Multiparty Vanilla C evaluation are as follows: $MSmcCX$ = [$rao*$, $rao1*$, $wao*$, $wao1*$, $wao2*$, $pin2*$, $pin3*$, $pin4*$, $pin5*$, $pin6*$, $pin7*$]. In all of these rules, where the algorithms return the tag 1 to indicate the access is well-aligned, the * versions of the rules would return 0. With these rules, it is not possible to prove correctness, as they would return garbage values that no longer are congruent between Multiparty SMC^2 and Multiparty Vanilla C. We can prove all of these rules to maintain noninterference - each case is similar to the corresponding non-* version, and therefore does not add anything of interest to the proof, so we omit these cases from this document.

In this section we present the most challenging methatheoretic result of correctness. We will begin by discussing how we leverage multiparty protocols, then proceed to discuss correctness. Once correctness is proven, noninterference follows from a standard argument, with some adaptations needed to deal with the fact that private data is encrypted and that we want to show indistinguishability of evaluation traces.

$MSmcC$	$MVanC$ Equivalent Cases	
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{ALL},[mpcmp])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{ALL},[\widehat{mpcmpt}])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{ALL},[\widehat{mpcmpf}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[iep])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{mpiet}])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{mpief}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[iepd])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{mpiet}])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{mpief}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[malp])}$	$\Downarrow'_{\widehat{d}::[(\mathrm{p},[\widehat{ty}]),(\mathrm{p},[\widehat{bm}]),(\mathrm{p},[\widehat{mal}])]}$	

Figure 5.33: Table of more complex Multiparty SMC^2 evaluation codes and their congruent Multiparty Vanilla C evaluation codes.

In our semantics, we leverage multiparty protocols to compartmentalize the complexity of handling private data. In the formal treatment this corresponds to using Axioms in our proofs to reason about protocols. These Axioms allow us to guarantee the desired properties of correctness and noninterference for the overall model, to provide easy integration with new, more efficient protocols as they become available, and to avoid re-proving the formal guarantees for the entire model when new protocols are added. Proving that these Axioms hold is a responsibility of the library implementor in order to have the system fully encompassed by our formal model. Secure multiparty computation protocols that already come with guarantees of correctness and security are the only ones worth considering, so the implementor would only need to ensure that these guarantees match our definitions of correctness and noninterference.

For example, if private values are represented using Shamir secret sharing [1], Algorithm 97, MPC_{mult}, represents a simple multiparty protocol for multiplying private values from [41]. In Algorithm 97, lines 2 and 3 define the protocol, while lines 1, 4, and 5 relate the protocol to our semantic representation.

When computation is performed by q parties, at most t of whom may collude ($t < q/2$), Shamir secret sharing encodes a private integer a by choosing a polynomial $f(x)$ of degree t with random coefficients such that $f(0) = a$ (all computation takes place over a finite field). Each participant obtains evaluation of f on a unique non-zero point as their representation of private a; for example, party p obtains $f(\mathrm{p})$. This representation has the property that combining t or fewer shares reveals no information about a as all values of a are equally likely; however, possession of $t+1$ or more shares permits recovering of $f(x)$ via polynomial interpolation and thus learning $f(0) = a$.

Multiplication in Algorithm 97 corresponds to each party locally multiplying shares of inputs a and b, which computes the product, but raises the polynomial degree to $2t$. The parties consequently re-share their private intermediate results to lower the polynomial degree to t and re-randomize the shares. Values λ_{p} refer to interpolation coefficients which are derived from the computation setup and party p index.

In order to preserve the correctness and noninterference guarantees of our model when such an algorithm is

$MSmcC$	$MVanC$	$MSmcC$	$MVanC$	$MSmcC$	$MVanC$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{ALL},[mpra])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{ALL},[\widehat{mpra}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[fc1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{fc}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[fc])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{fc}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{ALL},[mpwa])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{ALL},[\widehat{mpwa}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[fpd])}$	$\Downarrow'_{(\mathrm{p},[\widehat{fpd}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[df])}$	$\Downarrow'_{(\mathrm{p},[\widehat{df}])}$
$\Downarrow^{\mathcal{L}}_{(\mathrm{ALL},[mprdp])}$	$\Downarrow'_{(\mathrm{ALL},[\widehat{mprdp}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[pin])}$	$\Downarrow'_{(\mathrm{p},[\widehat{pin}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[fd])}$	$\Downarrow'_{(\mathrm{p},[\widehat{fd}])}$
$\Downarrow^{\mathcal{L}}_{(\mathrm{ALL},[mprdp1])}$	$\Downarrow'_{(\mathrm{ALL},[\widehat{mprdp1}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[pin3])}$	$\Downarrow'_{(\mathrm{p},[\widehat{pin}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[cl1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{cl}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{ALL},[mpwdp2])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{ALL},[\widehat{mpwdp1}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[pin1])}$	$\Downarrow'_{(\mathrm{p},[\widehat{pin1}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[cl])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{cl}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{ALL},[mpwdp1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{ALL},[\widehat{mpwdp1}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[pin2])}$	$\Downarrow'_{(\mathrm{p},[\widehat{pin2}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[cv1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{cv}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{ALL},[mpwdp3])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{ALL},[\widehat{mpwdp}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[pin4])}$	$\Downarrow'_{(\mathrm{p},[\widehat{pin1}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[cv])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{cv}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{ALL},[mpwdp])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{ALL},[\widehat{mpwdp}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[pin5])}$	$\Downarrow'_{(\mathrm{p},[\widehat{pin2}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[ltt])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{ltt}])}$
$\Downarrow^{\mathcal{L}}_{(\mathrm{ALL},[mppin])}$	$\Downarrow'_{(\mathrm{ALL},[\widehat{mppin}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[pin6])}$	$\Downarrow'_{(\mathrm{p},[\widehat{pin1}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[ltf])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{ltf}])}$
$\Downarrow^{\mathcal{L}}_{(\mathrm{ALL},[mpfre])}$	$\Downarrow'_{(\mathrm{ALL},[\widehat{mpfre}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[pin7])}$	$\Downarrow'_{(\mathrm{p},[\widehat{pin2}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[eqt])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{eqt}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{ALL},[mpb])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{ALL},[\widehat{mpb}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[eqf])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{eqf}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[sb])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{sb}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wdp1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wdp1}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[net])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{net}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[d1])}$	$\Downarrow'_{(\mathrm{p},[\widehat{dv}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wdp2])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wdp1}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[nef])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{nef}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[dv])}$	$\Downarrow'_{(\mathrm{p},[\widehat{dv}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wdp])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wdp}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[fre])}$	$\Downarrow'_{(\mathrm{p},[\widehat{fre}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[pfre])}$	$\Downarrow'_{(\mathrm{p},[\widehat{fre}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wdp3])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wdp}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[ief])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{ief}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[iet])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{iet}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wdp4])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wdp}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wle])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wle}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wlc])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wlc}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wdp4])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wdp}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[ss])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{ss}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[ds])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{ds}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[w])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{w}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[w1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{w}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[w2])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{w}])}$
$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[dp1])}$	$\Downarrow'_{(\mathrm{p},[\widehat{dp}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[dp])}$	$\Downarrow'_{(\mathrm{p},[\widehat{dp}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[rp])}$	$\Downarrow'_{(\mathrm{p},[\widehat{rp}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[mal])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{mal}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[rp1])}$	$\Downarrow'_{(\mathrm{p},[\widehat{rp}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wp1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wp}])}$
$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[r])}$	$\Downarrow'_{(\mathrm{p},[\widehat{r}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[r1])}$	$\Downarrow'_{(\mathrm{p},[\widehat{r}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wp])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wp}])}$
$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[ty])}$	$\Downarrow'_{(\mathrm{p},[\widehat{ty}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[loc])}$	$\Downarrow'_{(\mathrm{p},[\widehat{loc}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[rdp2])}$	$\Downarrow'_{(\mathrm{p},[\widehat{rdp2}])}$
$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[rdp1])}$	$\Downarrow'_{(\mathrm{p},[\widehat{rdp1}])}$	$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[rdp])}$	$\Downarrow'_{(\mathrm{p},[\widehat{rdp}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wp2])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wp}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[da])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{da}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[da1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{da}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[das])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{das}])}$
$\Downarrow^{\mathcal{L}}_{(\mathrm{p},[rea])}$	$\Downarrow'_{(\mathrm{p},[\widehat{rea}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[ra1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{ra}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[ra])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{ra}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wea2])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wea}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wea1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wea}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wea])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wea}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wa])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wa}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wa1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wa}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wa2])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wa}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[rao1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{rao}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[rao])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{rao}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[ep])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{ep}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wao2])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wao}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wao1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wao}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wao])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wao}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[bd])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{bd}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[bp])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{bp}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[bs])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{bs}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[inp3])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{inp1}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[inp2])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{inp}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[bm])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{bm}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[inp1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{inp1}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[inp])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{inp}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[out])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{out}])}$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[out3])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{out1}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[out2])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{out}])}$	$\Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[out1])}$	$\Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{out1}])}$

Figure 5.34: Table of Multiparty SMC^2 evaluation codes in $MSmcC$ and their congruent Multiparty Vanilla C evaluation codes in $MVanC$.

added, a library developer will need to guarantee that the implementation of this algorithm is correct, meaning that it has the expected input output behavior, and it guarantees noninterference on what is observable.

We first show the correctness of the Multiparty SMC^2 semantics with respect to the Multiparty Vanilla C semantics. As usual we will do this by establishing a simulation relation between a Multiparty SMC^2 program and a corresponding Multiparty Vanilla C program. To do so we face two main challenges.

First, we need to guarantee that the private operations in a Multiparty SMC^2 program are reflected in the corresponding Multiparty Vanilla C program and that the evaluation steps between the two programs correspond. To address the former issue, we define an *erasure function* $\mathbf{Erase}$ which translates a Multiparty SMC^2 program into a Multiparty Vanilla C program by erasing all labels and replacing all functions specific to Multiparty SMC^2 with their public equivalents. This function also translates memory. As an example, let us consider `pmalloc`; in this case, we have $\mathbf{Erase}(\mathrm{pmalloc}(e,\ ty) = (\mathrm{malloc}(\mathbf{Erase}(e)\cdot \mathrm{sizeof}(\mathbf{Erase}(ty)))))$. That is, `pmalloc` is rewritten to use `malloc`, and since the given private type is now public we can use the sizeof function to find the size we will need to allocate. To address the latter issue, we have defined our operational semantics in terms of big-step evaluation judgments which allow the evaluation trees of the two programs to have a corresponding structure. In particular, notice how we designed the Private If Else (Variable Tracking) and Private If Else (Location Tracking) rules to perform multiple operations in one step, guaranteeing that we have similar "synchronization points" in the two evaluation trees.

Second, we need to guarantee that at each evaluation step the memory used by a Multiparty SMC^2 program corresponds to the one used by the Multiparty Vanilla C program. Given that we simulate multiparty execution over q parties in Multiparty SMC^2, we will also use q parties in Multiparty Vanilla C. This allows us to easily reason about both local and global semantic rules, as each Multiparty SMC^2 party has a corresponding Multiparty Vanilla C party at an identical position in the evaluation trace. Unfortunately, just applying the function $\mathbf{Erase}$ to the Multiparty SMC^2 memories in the evaluation trace is not enough. In our setting, with explicit memory management, manipulations of pointers, and array overshooting, guaranteeing a correspondence between the memories becomes particularly challenging. To better understand the issue here, let us consider the rule Private Free. Remember that our semantic model associates a pointer with a list of locations, and the Private Free rule frees the first location in the list, and relocates the content of that location if it is not the true location. Essentially, this rule may swap the content of two locations if the first location in the list is not the location intended to be freed and make the Multiparty SMC^2 memory and the Multiparty Vanilla C memory look quite different. To address this challenge in the proof of correctness, we

use a *map*, denoted ψ, to track the swaps that happen when the rule Private Free is used. The simulation uses and modifies this map to guarantee that the two memories correspond. Another related challenge comes from array overshooting. If, by overshooting an array, a program goes over or into memory blocks of different types, we may end up in a situation where the locations in the Multiparty SMC^2 memory are significantly different from the ones in the Multiparty Vanilla C memory. This is mostly due to the size of private types being larger than their public counterpart. One option to address this problem would be to keep a more complex map between the two memories. However, this can result in a much more complex proof, for capturing a behavior that is faulty, in principle. Instead, we prefer to focus on situations where overshooting arrays are *well-aligned*, in the sense that they access only memory locations and blocks of the right type and size. An illustration of this is given in Figure 3.4.

Before stating our correctness, we need to introduce some notation. We use party-wise lists of codes $\mathcal{D} = (1, [d_1, \ldots, d_n]) \parallel ... \parallel (\mathrm{q}, [d_1, \ldots, d_n])$, $\widehat{\mathcal{D}} = (1, [\widehat{d}_1, \ldots, \widehat{d}_m]) \parallel \ ... \ \parallel (\mathrm{q}, [\widehat{d}_1, \ldots, \widehat{d}_m])$ in evaluations (i.e., $\Downarrow_{\mathcal{D}}$) to describe the rules of the semantics that are applied in order to derive the result. We write $\mathcal{D} \cong \widehat{\mathcal{D}}$ to state that the Multiparty SMC^2 codes are in correspondence with the Multiparty Vanilla C codes, $\mathcal{D}^{\mathrm{p}}$ to denote the list of codes for a specific party p, and $\mathcal{D}_1 :: \mathcal{D}_2$ to denote concatenation of the party-wise evaluation code lists. We write $\{...\}_{\mathrm{p}=1}^{\mathrm{q}}$ to show that an assertion holds for all parties. Almost every Multiparty SMC^2 rule is in one-to-one correspondence with a single Multiparty Vanilla C rule within an execution trace (exceptions being private-conditioned branches, `pmalloc`, and multiparty comparison operations).

We write $s \cong \widehat{s}$ to state that the Multiparty Vanilla C configuration statement $\widehat{s}$ can be obtained by applying the erasure function to the Multiparty SMC^2 statement s. Similarly, we can extend this notation to configuration by also using the map ψ. That is, we write $(\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s) \cong_{\psi} (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s})$ to state that the Multiparty Vanilla C configuration $(\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s})$ can be obtained by applying the erasure function to the Multiparty SMC^2 configuration $(\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s)$, and memory $\widehat{\sigma}$ can be obtained from σ by using the map ψ.

We state correctness in terms of evaluation trees, since we will use evaluation trees to prove a strong form of noninterference in the next subsection. We use capital Greek letters Π, Σ to denote evaluation trees. In the Multiparty SMC^2 semantics, we write $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, s^1) \ \parallel ... \parallel \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}^{\mathrm{q}}, s^{\mathrm{q}})) \Downarrow_{\mathcal{D}}^{\mathcal{L}} ((1, \gamma_1^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}_1^1, v^1) \parallel ... \parallel \ (\mathrm{q}, \gamma_1^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}_1^{\mathrm{q}}, v^{\mathrm{q}}))$, to stress that the evaluation tree Π proves as conclusion that, for each party p, configuration $(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}^{\mathrm{p}}, s^{\mathrm{p}})$ evaluates to configuration $(\mathrm{p}, \gamma_1^{\mathrm{p}},$

$\sigma_1^{\mathrm{p}}, \Delta_1^{\mathrm{p}}, \mathrm{acc}_1^{\mathrm{p}}, v^{\mathrm{p}})$ by means of the codes in $\mathcal{D}^{\mathrm{p}}$. Similarly, for the Multiparty Vanilla C semantics. We then write $\Pi \cong_{\psi} \Sigma$ for the extension to evaluation trees of the congruence relation with map ψ.

In order to properly reason about global multiparty rules, we must assert that all parties are executing from the same original program with corresponding start states and input. To do this, we first show that the non-determinism of the semantics will always bring all parties to the same outcome: given q parties with corresponding start states, if we reach intermediate states that are not corresponding for one or more parties, then there exists a set of steps that will bring all parties to corresponding states again. This is formalized within Theorem 5.2.2, Confluence.

We can now state our correctness result showing that if an Multiparty SMC^2 program s can be evaluated to a value v, and the evaluation is well-aligned (it is an evaluation where all the overshooting of arrays are well-aligned), then the Multiparty Vanilla C program $\hat{s}$ obtained by applying the erasure function to s, i.e., $s \cong \hat{s}$, can be evaluated to $\hat{v}$ where $v \cong \hat{v}$. This property can be formalized in terms of congruence.

Axiom 5.2.1. *For purposes of correctness, we assume all parties are executing a program s from initial state* $(\mathrm{p}, [\,], [\,], [\,], 0, s)$ *with congruent input data. We assume that s does not contain hard-coded locations, has* well-aligned *out-of-bounds memory accesses where private indices are not used and no out-of-bounds accesses where private indices are used, and type-casts for private locations match the intended type that the location was allocated for.*

Theorem 5.2.1 (Semantic Correctness)**.**
For every configuration $\{(\mathrm{p},\ \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}^{\mathrm{p}}, s^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{(\mathrm{p}, \widehat{\gamma}^{\mathrm{p}}, \widehat{\sigma}^{\mathrm{p}}, \Box, \Box, \widehat{s}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and map* ψ
such that $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}^{\mathrm{p}}, s^{\mathrm{p}}) \cong_{\psi} (\mathrm{p}, \widehat{\gamma}^{\mathrm{p}}, \widehat{\sigma}^{\mathrm{p}}, \Box, \Box, \widehat{s}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$,
if $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, s^1)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}^{\mathrm{q}}, s^{\mathrm{q}}))$
$\Downarrow_{\mathcal{D}}^{\mathcal{L}} ((1, \gamma_1^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}_1^1, v^1)\ \|\ ...\ \|\ (\mathrm{q}, \gamma_1^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}_1^{\mathrm{q}}, v^{\mathrm{q}}))$
for codes $\mathcal{D} \in MSmcC$*, then there exists a derivation*
$\Sigma \triangleright\ ((1, \widehat{\gamma}^1, \widehat{\sigma}^1, \Box, \Box, \widehat{s}^1)\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma}^{\mathrm{q}}, \widehat{\sigma}^{\mathrm{q}}, \Box, \Box, \widehat{s}^{\mathrm{q}}))$
$\Downarrow_{\widehat{\mathcal{D}}} ((1, \widehat{\gamma}_1^1, \widehat{\sigma}_1^1, \Box, \Box, \widehat{v}^1)\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma}_1^{\mathrm{q}}, \widehat{\sigma}_1^{\mathrm{q}}, \Box, \Box, \widehat{v}^{\mathrm{q}}))$
for codes $\widehat{\mathcal{D}} \in MVanC$ *and a map* ψ_1 *such that*
$\mathcal{D} \cong \widehat{\mathcal{D}}$, $\{(\mathrm{p}, \gamma_1^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, \Delta_1^{\mathrm{p}}, \mathrm{acc}_1^{\mathrm{p}}, v^{\mathrm{p}}) \cong_{\psi_1} (\mathrm{p}, \widehat{\gamma}_1^{\mathrm{p}}, \widehat{\sigma}_1^{\mathrm{p}}, \Box, \Box, \widehat{v}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, *and* $\Pi \cong_{\psi_1} \Sigma$.

Proof. Proof Sketch: By induction over all Multiparty SMC^2 semantic rules.

The bulk of the complexity of this proof lies with rules pertaining to Private If Else, handling of pointers, and freeing of memory. We first provide a brief overview of the intuition for the simpler cases and then dive deeper into the details for the more complex cases. The full proof is available in Section 5.2.5, with this theorem identical to Theorem 5.2.3.

For the rules evaluating over public data, correctness follows simply as the Multiparty Vanilla C and Multiparty SMC^2 rules for public data are nearly identical. For all the semantic rules that use general helper algorithms (i.e., algorithms in common to both Multiparty Vanilla C and Multiparty SMC^2), we also reason about the correctness of the helper algorithms, comparing the Multiparty Vanilla C version and the Multiparty SMC^{2}version. Correctness over such algorithms is easily proven, as these algorithms are nearly identical, differing on privacy labels as we do not have private data in Multiparty Vanilla C.

For all Multiparty SMC^2 multiparty semantic rules, we relate them to the multiparty versions of the Multiparty Vanilla C rules. To reason about the multiparty protocols, we leverage Axioms, such as Axiom 5.2.4, to prove these rules correct. These Axioms should be proven correct by a library developer to ensure the completeness of the formal model. The correctness of most multiparty semantic rules follows easily, with Multiparty Private Free being an exception. For this rule, we also must reason about our helper algorithms that are specific to the Multiparty SMC^2 semantics (e.g., $\mathrm{UpdateBytesFree}$, $\mathrm{UpdatePointerLocations}$). We leverage the correctness of the behavior of the multiparty protocol MPC_{free}, to show that correctness of these algorithms follows due to the deterministic definitions of the algorithms. In this case, we must also show that the locations that are swapped within this rule (which is done to hide the true location) are deterministic based on our memory model definition. We use ψ to map the swapped locations, enabling us to show that, if these swaps were reversed, we would once again have memories that are directly congruent. This concept of locations being ψ-congruent is particularly necessary when reasoning about pointers in other rule cases. For all the rules using private pointers, we will rely upon the pointer data structure containing a set of locations and their associated tags, only one of which being the true location. With this proven to be the case, it is then clear that the true location indicated within the private pointer's data structure in Multiparty SMC^2 will be ψ-congruent with the location given by the pointer data structure in Multiparty Vanilla C. In our proof, we make the assumption that locations are not hard-coded, as hard-coded locations would lead to potentially differing results between Multiparty Vanilla C and Multiparty SMC^2 execution due to the behavior of `pfree`. Additionally, given the distributed nature of the Multiparty SMC^2, it would not make sense to allow hard-coded locations, as a single program will be executed on several different machines.

For rule Private Malloc, we must relate this rule to the sequence of Multiparty Vanilla C rules for Malloc, Multiplication, and Size Of Type. This is due to the definition of `pmalloc` as a helper that allows the user to write programs without knowing the size of private types. This case follows from the definition of translating the Multiparty SMC^2 program to a Multiparty Vanilla C program, $\mathbf{Erase}(\mathrm{pmalloc}(e,\ ty) = (\mathrm{malloc}(\mathrm{sizeof}(\mathbf{Erase}(ty)) \cdot \mathbf{Erase}(e))))$.

For the Private If Else rules, we must reason that our end results in memory after executing both branches and resolving correctly match the end result of having only executed the intended branch. The cases for both of these rules will have two subcases - one for the conditional being true, and the other for false. To obtain correctness, we use multiparty versions of the if else true and false rules that execute both branches - this allows us to reason that both

branches will evaluate properly, and that we will obtain the correct ending state once completed. For both rules, we must first show that $\mathrm{Extract}$ will correctly find all non-local variables that are modified within both branches, including non-assignment modifications such as use of the pre-increment operator $++x$, and that all such modified variables will be added to the list (excluding pointers modified exclusively by pointer dereference write statements). We must also show that it will correctly find and tag if a pointer dereference write statement was found. These properties follow deterministically from the definition of the algorithm.

For rule Private If Else Variable Tracking, we will leverage the correctness of $\mathrm{Extract}$, and that if $\mathrm{Extract}$ returns the tag 0, no pointer dereference writes were found. We then reason that $\mathrm{InitializeVariables}$ will correctly create the assignment statements for our temporary variables, and that the original values for each of the modified variables will be stored into the **`else`** temporary variables. The temporaries being stored into memory correctly through the evaluation of these statements follows by induction. Next we have the evaluation of the **`then`** branch, which will result in the values that are correct for if the condition had been true - this holds by induction. We then proceed to reason that $\mathrm{RestoreVariables}$ will properly create the statements to store the ending results of the **`then`** branch into the **`then`** temporary variables, and restore all of the original values from the **`else`** variables (the original values being correctly stored follows from $\mathrm{InitializeVariables}$ and the evaluation of it's statements). The correct evaluation of the this set of statements follows by induction. Next we have the evaluation of the **`else`** branch, which will result in the values that are correct for if the condition had been false - this holds by induction and the values having been restored to the original values properly. We will then reason about the correctness of the statements created by $\mathrm{ResolveVariables}$. These statements must be set up to correctly take the information from the **`then`** temporary variable, the temporary variable for the condition for the branch, and the ending result for all variables from the **`else`** branch. For the resolution of pointers, we insert a call for a resolution function ($\mathrm{resolve}$), because the resolution of pointer data is more involved. The evaluation of this function is shown in rule Multiparty Resolve Pointer Locations. By proving that this rule will correctly resolve the true locations for pointers, we will then have that the statements created by $\mathrm{ResolveVariables}$ will appropriately resolve all pointers.

For rule Private If Else Location Tracking, the structure of the case is similar to the rule for variable tracking, but with a few differences we will discuss here. For this rule, we will need to reason about $\mathrm{DynamicUpdate}$, and that we will catch all modifications by pointer dereference writes and properly add them to Δ if the location being modified is not already tracked. If a new mapping is added, we store the current value in v_{orig} (as this location has not yet been modified) and the tag has to be set to 0. This behavior will be used to ensure the correctness during resolution. For $\mathrm{Initialize}$, we must reason that we correctly initialize the map Δ with all of the locations we found within $\mathrm{Extract}$ to be modified by means other than pointer dereference writes and store their original values in v_{orig}. Then we can evaluate the **`then`** branch, which will result in the values that are correct for if the condition had been true - this holds by induction. For $\mathrm{Restore}$, we reason that we properly store the results of the **`then`** branch, and update the tag for the

location to signify that we should use v_{then} instead of v_{orig}. We will then restore the original values, leveraging the correctness of Initialize to prove this will happen correctly. Then we can evaluate the **else** branch, which will result in the values that are correct for if the condition had been false - this holds by induction. For Resolve, we reason that we will create the appropriate resolution statements to be executed. For the **then** result, these statements must use the value stored in v_{orig} if the tag is set to 0 (this occurs if the first modification to the location was a pointer dereference write within the **else** branch), and the value stored in v_{then} if the tag is set to 1. We prove this to be the correct **then** result through the correctness of DynamicUpdate and Restore. The **else** result must use the current value for that location in memory, which is proven to be the correct **else** result through the correctness of Initialize and Resolve. In this way, we can prove the correctness the contents of the statements created by Resolve, and then the correctness of the evaluation of the statements created by Restore will hold as we discussed for with those created by ResolveVariables for Private If Else Variable tracking. □

The full erasure function is shown in Section 3.2.1, Figure 3.31. The only difference is the update to the erasure function over configurations, replacing subfigure 3.31a with subfigure 5.35a. Figure 5.35 shows erasure over an entire Multiparty SMC^2 configuration, calling Erase on the four-tuple of the environment, memory, and two empty maps needed as the base for the Vanilla C environment and memory; removing the accumulator (i.e., replacing it with □); and calling Erase on the statement.

$$
\begin{aligned}
&\mathbf{Erase}(\boldsymbol{C}) = \\
&\mid C_1 \parallel C_2 \;=>\; \mathbf{Erase}(\boldsymbol{C_1}) \parallel \mathbf{Erase}(\boldsymbol{C_2}) \\
&\mid (\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s) \;=>\; (\mathbf{p}, \mathbf{Erase}(\boldsymbol{\gamma}, \boldsymbol{\sigma}, [\,], [\,]), \square, \square, \mathbf{Erase}(\boldsymbol{s}))
\end{aligned}
$$

(a) Erasure function over configurations.

Figure 5.35: The Erasure function from Multiparty SMC^2 configurations to Multiparty Vanilla C configurations.

Algorithm 145 $j \leftarrow (l) \nvdash \sigma$

```
j = 0
(ω, ty, n, PermL(perm, ty, a, n)) = σ(l)
if a = public then
    j = 1
end if
return j
```

Algorithm 146 $j \leftarrow (l) \vdash \sigma$

```
j = 0
(ω, ty, n, PermL(perm, ty, a, n)) = σ(l)
if a = private then
    j = 1
end if
return j
```

5.2.1 Definitions

Definition 5.2.1 (ψ)**.** A map ψ is defined as a list of lists of locations, in symbols $\psi = [\,] \mid \psi[\bar{l}]$, that is formed by tracking which locations are privately switched during the execution of the statement $\mathrm{pfree}(x)$ in a SMC^2 program s to enable comparison with the *congruent* Vanilla C program $\widehat{s}$.

Definition 5.2.2 (aligned memory location)**.** A memory location $(l, \mu), (\widehat{l}, \widehat{\mu})$ is *aligned*
if and only if the location refers to either the beginning of a memory block ($\mu = \widehat{\mu} = 0$)
or the beginning of an element inside an array.

Definition 5.2.3 (*well-aligned* memory access)**.** An overshooting memory access by an array is *well-aligned* if and only if:

- the initial memory location is *aligned* and of the expected type,
- the ending memory location is *aligned* and of the expected type, and
- all memory blocks or elements iterated over are of the expected type.

Definition 5.2.4 ($j \cong \widehat{j}$)**.** A SMC^2 alignment indicator and a Vanilla C alignment indicator are *congruent*, in symbols $j \cong \widehat{j}$,
if and only if either $j = 1$ and $\widehat{j} = 1$
or $j = 0$ and $(\widehat{j} = 0) \vee (\widehat{j} = 1)$.

Definition 5.2.5 (*aligned* location list)**.** A location list is *aligned* if and only if for all locations (l_i, μ_i) in the list:

- all memory block identifiers l_i are of the expected type,
- all memory block identifiers l_i are of the same size, and

- all offsets μ_i are equal.

Definition 5.2.6 (*well-aligned* pointer access)**.** An overshooting memory access by a pointer is *well-aligned* if and only if:

- the initial location list $\bar{l}_i$ is *aligned*,
- the final location list $\bar{l}_f$ is *aligned*, and
- for each location in the initial location list, all memory blocks or elements iterated over to get to the corresponding location in the final location list are of the expected type.

Definition 5.2.7 ($ty \cong \hat{ty}$)**.** A SMC^2 type and a Vanilla C type are *congruent*, in symbols $ty \cong \hat{ty}$,
if and only if $\mathrm{Erase}(ty) = \hat{ty}$.

Definition 5.2.8 ($ty \cong_\psi \hat{ty}$)**.** A SMC^2 type and a Vanilla C type are ψ-*congruent*, in symbols $ty \cong_\psi \hat{ty}$,
if and only if $ty \cong \hat{ty}$.

Definition 5.2.9 ($\overline{ty} \cong \hat{\overline{ty}}$)**.** A SMC^2 type list and a Vanilla C type list are *congruent*, in symbols $\overline{ty} \cong \hat{\overline{ty}}$, if and only if $\mathrm{Erase}(\overline{ty}) = \hat{\overline{ty}}$.

Definition 5.2.10 ($\overline{e} \cong \hat{\overline{e}}$)**.** A SMC^2 expression list and a Vanilla C expression list are *congruent*, in symbols $\overline{e} \cong \hat{\overline{e}}$, if and only if $\mathrm{Erase}(\overline{e}) = \hat{\overline{e}}$.

Definition 5.2.11 ($\overline{p} \cong \hat{\overline{p}}$)**.** A SMC^2 parameter list and a Vanilla C parameter list are *congruent*, in symbols $\overline{p} \cong \hat{\overline{p}}$, if and only if $\mathrm{Erase}(\overline{p}) = \hat{\overline{p}}$.

Definition 5.2.12. A SMC^2 statement and a Vanilla C statement are *congruent*, in symbols $s \cong \hat{s}$, if and only if $\mathrm{Erase}(s) = \hat{s}$.

Definition 5.2.13 ($l \cong_\psi \widehat{l}$)**.** A SMC^2 memory block identifier and a Vanilla C memory block identifier are ψ-*congruent*, in symbols $l \cong_\psi \widehat{l}$, given map ψ,
if and only if $\mathrm{CheckIDCongruence}(\psi, l, \widehat{l}) = 1$.

Definition 5.2.14 ($(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$)**.** A SMC^2 location and a Vanilla C location are ψ-*congruent*, in symbols $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$, given SMC^2 type ty correlating to (l, μ) and Vanilla C type $\widehat{ty}$ correlating to $(\widehat{l}, \widehat{\mu})$,
if and only if $ty \cong \widehat{ty}$, $l \cong_\psi \widehat{l}$, and
either ty is a public type and $\mu = \widehat{\mu}$,
or ty is a private type and $(\mu) \cdot \left(\frac{\tau(\widehat{ty})}{\tau(ty)}\right) = \widehat{\mu}$.

Definition 5.2.15 ($ptr \cong_\psi ptr$)**.** A SMC^2 pointer data structure for a pointer of type $ty \in \{a \text{ const } bty*, a\ bty*\}$ and a Vanilla C pointer data structure for a pointer of type $\widehat{ty} \in \{\text{const } \widehat{bty}*, \widehat{bty}*\}$ are ψ-*congruent*, in symbols $[\alpha, \bar{l}, \bar{j}, i] \cong_\psi [1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}]$, given map ψ,

if $ty \cong \widehat{ty}$, $i = \widehat{i}$ and

either $a = \text{public}$, $\alpha = 1$, $\bar{l} = (l, \mu)$ such that $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$ and $\bar{j} = [1]$

or $a = \text{private}$ and $\text{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ i],\ \text{private } bty*) = (l, \mu)$ such that $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$.

Definition 5.2.16 ($(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$)**.** A SMC^2 environment and memory pair and a Vanilla C environment and memory pair are ψ-*congruent*, in symbols $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$,

if and only if $\text{Erase}(\gamma, \sigma, [\,], [\,]) = (\widehat{\gamma}, \widehat{\sigma}')$ and $\text{SwapMemory}(\widehat{\sigma}', \psi) = \widehat{\sigma}$.

Definition 5.2.17 ($\omega \cong_\psi \hat{\omega}$)**.** A SMC^2 byte-wise representation ω of a given type ty and size n and a Vanilla C byte-wise representation $\widehat{\omega}$ are ψ-*congruent*, in symbols $\omega \cong_\psi \hat{\omega}$,

if and only if either $ty \neq \text{private } bty*$ and $\text{Erase}(\omega, ty, n) = \hat{\omega}$

or $ty = \text{private } bty*$ and $\text{Erase}(\omega, ty, n) = \hat{\omega}_1$ such that the pointer data structure stored in ω and the pointer data structure stored in $\hat{\omega}$ are ψ-*congruent* by Definition 5.2.15.

Definition 5.2.18 ($v \cong \widehat{v}$)**.** A SMC^2 value and Vanilla C value are *congruent*, in symbols $v \cong \widehat{v}$, if and only if $\text{Erase}(v) = \widehat{v}$.

Definition 5.2.19 ($v \cong_\psi \widehat{v}$)**.** A SMC^2 value and Vanilla C value are ψ-*congruent*, in symbols $v \cong_\psi \widehat{v}$,

if and only if either $v \neq (l, \mu)$, $\widehat{v} \neq (\widehat{l}, \widehat{\mu})$ and $v \cong \widehat{v}$,

or $v = (l, \mu)$, $\widehat{v} = (\widehat{l}, \widehat{\mu})$ and $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$.

Definition 5.2.20 ($s \cong_\psi \widehat{s}$)**.** A SMC^2 statement and Vanilla C statement are ψ-*congruent*, in symbols $s \cong_\psi \widehat{s}$, if and only if for all $v_i \in s$, $\widehat{v}_i \in \widehat{s}$ such that $v_i \cong_\psi \widehat{v}_i$ and otherwise $s \cong \widehat{s}$.

Definition 5.2.21 ($\overline{e} \cong_\psi \hat{\overline{e}}$)**.** A SMC^2 expression list and a Vanilla C expression list are ψ-*congruent*, in symbols $\overline{e} \cong_\psi \hat{\overline{e}}$, given a map ψ, if and only if $\forall e \neq (l, \mu) \in \overline{e}$, $\text{Erase}(e) = \widehat{e}$ and $\forall e = (l, \mu) \in \overline{e}$, $e \cong_\psi \widehat{e}$ by Definition 5.2.20.

Definition 5.2.22 ($\{(\text{p},\ \gamma^{\text{p}},\ \sigma^{\text{p}},\ \Delta^{\text{p}},\ \text{acc}^{\text{p}},\ s^{\text{p}}) \cong_\psi (\text{p}, \widehat{\gamma}^{\text{p}}, \widehat{\sigma}^{\text{p}}, \square, \square, \widehat{s}^{\text{p}})\}_{\text{p}=1}^{\text{q}}$)**.** A SMC^2 configuration and a Vanilla C configuration are ψ-*congruent*, in symbols $\{(\text{p},\ \gamma^{\text{p}},\ \sigma^{\text{p}},\ \Delta^{\text{p}},\ \text{acc}^{\text{p}},\ s^{\text{p}}) \cong_\psi (\text{p}, \widehat{\gamma}^{\text{p}}, \widehat{\sigma}^{\text{p}}, \square, \square, \widehat{s}^{\text{p}})\}_{\text{p}=1}^{\text{q}}$ or $C \cong_\psi \widehat{C}$, if and only if $\{(\gamma^{\text{p}}, \sigma^{\text{p}}) \cong_\psi (\widehat{\gamma}^{\text{p}}, \widehat{\sigma}^{\text{p}})\}_{\text{p}=1}^{\text{q}}$ and $\{s^{\text{p}} \cong_\psi \widehat{s}^{\text{p}}\}_{\text{p}=1}^{\text{q}}$.

Definition 5.2.23 ($d \cong \widehat{d}$)**.** We define *congruence* over SMC^2 codes $d \in SmcC$ and $\widehat{d} \in VanC$, in symbols $d \cong \widehat{d}$, by cases as follows:

if $d = \widehat{d}$, then $d \cong \widehat{d}$,

if $d = iep \oplus iepd$, then $\widehat{d} = \widehat{mpiet} \oplus \widehat{mpief}$ and $d \cong \widehat{d}$,

if $d = mpcmp$, then $\widehat{d} = \widehat{mpcmpt} \oplus \widehat{mpcmpf}$ and $d \cong \widehat{d}$,

otherwise we have $[malp] \cong [\widehat{ty}, \widehat{bm}, \widehat{mal}]$, $fc1 \cong \widehat{fc}$, $pin3 \cong \widehat{pin}$, $cl1 \cong \widehat{cl}$, $mpwdp2 \cong \widehat{mpwdp1}$, $cv1 \cong \widehat{cv}$, $mpwdp \cong \widehat{mpwdp}$, $pin4 \cong \widehat{pin1}$, $pin5 \cong \widehat{pin2}$, $mpwdp3 \cong \widehat{mpwdp}$, $pin6 \cong \widehat{pin1}$, $pin7 \cong \widehat{pin2}$, $r1 \cong \widehat{r}$, $w1 \cong \widehat{w}$, $w2 \cong \widehat{w}$, $d1 \cong \widehat{d}$, $wdp2 \cong \widehat{wdp1}$, $dp1 \cong \widehat{dp}$, $wdp3 \cong \widehat{wdp}$, $rp1 \cong \widehat{rp}$, $wdp4 \cong \widehat{wdp}$, $wp1 \cong \widehat{wp}$, $rdp1 \cong \widehat{rdp1}$, $wp2 \cong \widehat{wp}$, $da1 \cong \widehat{da}$, $ra1 \cong \widehat{ra}$, $wea2 \cong \widehat{wea}$, $wea1 \cong \widehat{wea}$, $rao1 \cong \widehat{rao}$, $wa1 \cong \widehat{wa}$, $wa2 \cong \widehat{wa}$, $wa1p \cong \widehat{wa}$, $wa2p \cong \widehat{wa}$, $wao2 \cong \widehat{wao}$, $wao1 \cong \widehat{wao}$, $inp3 \cong \widehat{inp1}$, $inp2 \cong \widehat{inp}$, $out3 \cong \widehat{out1}$, and $out2 \cong \widehat{out}$.

Definition 5.2.24 ($\overline{d} \cong \widehat{\overline{d}}$)**.** A SMC^2 evaluation code trace for a single party and a Vanilla C evaluation code trace for a single party are *congruent*, in symbols $\overline{d} \cong \widehat{\overline{d}}$, if and only if $\mathrm{CheckCodeCongruence}(\overline{d}, \widehat{\overline{d}}) = 1$ by Algorithm 83.

Definition 5.2.25 ($(\mathrm{p}, \overline{d}) \cong (\mathrm{p}, \widehat{\overline{d}})$)**.** A party-wise SMC^2 code trace $(\mathrm{p}, \overline{d})$ and a party-wise Vanilla C code trace $(\mathrm{p}, \widehat{\overline{d}})$ are *congruent*, in symbols $(\mathrm{p}, \overline{d}) \cong (\mathrm{p}, \widehat{\overline{d}})$, if and only if $\overline{d} \cong \widehat{\overline{d}}$.

Definition 5.2.26 ($\Pi \cong_{\psi} \Sigma$)**.** Two derivations and ψ-*congruent*, in symbols $\Pi \cong_{\psi} \Sigma$, if and only if
$\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, s^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}^{\mathrm{q}}, s^{\mathrm{q}}))$
$\Downarrow^{\mathcal{L}}_{\mathcal{D}} ((1, \gamma^1_1, \sigma^1_1, \Delta^1_1, \mathrm{acc}^1_1, v^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}_1, \sigma^{\mathrm{q}}_1, \Delta^{\mathrm{q}}_1, \mathrm{acc}^{\mathrm{q}}_1, v^{\mathrm{q}}))$ and
$\Sigma \triangleright ((1, \widehat{\gamma}^1, \widehat{\sigma}^1, \Box, \Box, \widehat{s}^1) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}^{\mathrm{q}}, \widehat{\sigma}^{\mathrm{q}}, \Box, \Box, \widehat{s}^{\mathrm{q}})) \Downarrow_{\widehat{\mathcal{D}}} ((1, \widehat{\gamma}^1_1, \widehat{\sigma}^1_1, \Box, \Box, \widehat{v}^1) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}^{\mathrm{q}}_1, \widehat{\sigma}^{\mathrm{q}}_1, \Box, \Box, \widehat{v}^{\mathrm{q}}))$ such that $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}^{\mathrm{p}}, s^{\mathrm{p}}) \cong_{\psi_1} (\mathrm{p}, \widehat{\gamma}^{\mathrm{p}}, \widehat{\sigma}^{\mathrm{p}}, \Box, \Box, \widehat{s}^{\mathrm{p}})\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\mathcal{D} \cong \widehat{\mathcal{D}}$, and $\{(\mathrm{p}, \gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_1, \Delta^{\mathrm{p}}_1, \mathrm{acc}^{\mathrm{p}}_1, v^{\mathrm{p}}) \cong_{\psi} (\mathrm{p}, \widehat{\gamma}^{\mathrm{p}}_1, \widehat{\sigma}^{\mathrm{p}}_1, \Box, \Box, \widehat{v}^{\mathrm{p}})\}^{\mathrm{q}}_{\mathrm{p}=1}$ such that ψ was derived from ψ_1 and the derivation Π.

Definition 5.2.27. Two input files are *congruent*, in symbols $inp \cong \hat{inp}$, if and only if for all mappings of variables to number values $x = v \in inp$ and $\widehat{x} = \widehat{v} \in \hat{inp}$, $x = \widehat{x}$ and $v \cong \widehat{v}$ by Definition 5.2.12.

Definition 5.2.28. Two output files are *congruent*, in symbols $out \cong \hat{out}$, if and only if for all mappings of variables to number values $x = v \in out$ and $\widehat{x} = \widehat{v} \in \hat{out}$, $x = \widehat{x}$ and $v \cong \widehat{v}$ by Definition 5.2.12.

Definition 5.2.29 (non-constant location)**.** A statement s is considered to update a *non-constant location* if the location that is being updated by such a statement can be modified (such as that which a pointer refers to) or overshot (such as that of a public index into an array).

Definition 5.2.30 (constant location)**.** A statement s is considered to update a *constant location* if the location that is being updated by such a statement cannot be modified (such l when $\gamma(x) = (l, ty)$) or overshot (such as that of a private index into an array).

Definition 5.2.31 (Δ complete)**.** The given nesting level of a location map $\Delta[\mathrm{acc}]$ is considered to be *complete* if all non-local locations that have been updated within the evaluation of the Private If Else statement have mappings within $\Delta[\mathrm{acc}]$ such that the value in v_{orig} is the original value.

Definition 5.2.32 (Δ then-complete)**.** The given nesting level of a location map $\Delta[\mathrm{acc}]$ is considered to be *then-complete* if $\Delta[\mathrm{acc}]$ is *complete* and

either the location was updated in the **`then`** branch and therefore has a value stored for v_{then} and tag set to 1,

or the location was not updated in the **`then`** branch.

Definition 5.2.33 (Δ else-complete)**.** The given nesting level of a location map $\Delta[\mathrm{acc}]$ is considered to be *else-complete* if $\Delta[\mathrm{acc}]$ was *then-complete* after the evaluation of restoration and $\Delta[\mathrm{acc}]$ is *complete* after evaluation after the **`else`** branch.

Definition 5.2.34 ($(\gamma, \sigma) \models (x \equiv v)$)**.** Variable x is equivalent to value v in the environment and memory pair (γ, σ), in symbols $(\gamma, \sigma) \models (x \equiv v)$, if and only if there is a valid mapping for x in the environment $\gamma(x) = (l, ty)$ and a corresponding mapping in memory $\sigma(l) = (\omega, ty, \alpha, \mathrm{PermL}(perm, ty, a, \alpha))$ such that the byte representation ω can be decoded by the given type ty to obtain value v.

Definition 5.2.35 ($(\sigma) \models_l ((l, \mu) \equiv_{ty} v)$)**.** The bytes at location (l, μ) interpreted as type ty in the given memory σ are equivalent to the given value v, in symbols $(\sigma) \models_l ((l, \mu) \equiv_{ty} v)$, if and only if there is a valid mapping for l in memory $\sigma(l) = (\omega, ty_1, \alpha, \mathrm{PermL}(perm, ty_1, a, \alpha))$ such that the byte representation ω from the offset μ can be decoded by the given type ty to obtain value v.

5.2.2 Lemmas

Axiom 5.2.2. *Given a SMC^2 program of statement s and a ψ-congruent Vanilla C program of statement $\widehat{s}$, in symbols $s \cong_\psi \widehat{s}$, any time a new memory block identifier is obtained from the available pool in the SMC^2 program such that $l = \phi()$, an identical memory block identifier is obtained from the available pool in the Vanilla C program such that $\widehat{l} = \phi()$ and $l = \widehat{l}$ and $(l, 0) \cong_\psi (\widehat{l}, 0)$.*

Axiom 5.2.3. *Given a SMC^2 private pointer data structure $[\alpha, \bar{l}, \bar{j}, i]$ stored at memory block l and ψ-congruent Vanilla C pointer data structure $[1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$ stored at ψ-congruent memory block $\widehat{l}$, we consider $l, \widehat{l}$ to be* equally freeable *if either:*

- *both* $\mathrm{CheckFreeable}(\gamma,\ \bar{l},\ \bar{j}) = 1$ *and* $\mathrm{CheckFreeable}(\widehat{\gamma},\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1]) = 1$*, or*
- *both* $\mathrm{CheckFreeable}(\gamma,\ \bar{l},\ \bar{j}) = 0$ *and* $\mathrm{CheckFreeable}(\widehat{\gamma},\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1]) = 0$.

Lemma 5.2.1. *Given $*, *$ if $\mathrm{GetIndirection}(*) = i$ and $|*| = |\ *\ |$, then $\mathrm{GetIndirection}(*) = \widehat{i}$ such that $i = \widehat{i}$.*

Proof. Proof Sketch:

By definition of function Erase, when two types are congruent, their levels of indirection will be the same. Therefore, when we evaluate the level of indirection from the number of $*$, we will get the same number in both SMC^2 and Vanilla C. □

Lemma 5.2.2. *Given parameter list* $\overline{p}, \widehat{\overline{p}}$ *and* ψ*, if* $\mathrm{GetFunTypeList}(\overline{p}) = \overline{ty}$ *and* $\overline{p} \cong_\psi \widehat{\overline{p}}$*, then* $\mathrm{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}}$ *such that* $\overline{ty} \cong_\psi \widehat{\overline{ty}}$*.*

Proof. Proof Sketch:

By the definition of Algorithm GetFunTypeList, GetFunTypeList, and function Erase. □

Lemma 5.2.3. *Given parameter list* $\overline{p}, \widehat{\overline{p}}$ *and expression list* $\overline{e}, \widehat{\overline{e}}$*, if* $\mathrm{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$*,* $\overline{p} \cong \widehat{\overline{p}}$*, and* $\overline{e} \cong \widehat{\overline{e}}$*, then* $\mathrm{GetFunParamAssign}(\widehat{\overline{p}}, \widehat{\overline{e}}) = \widehat{s}_1$ *where* $s_1 \cong_\psi \widehat{s}_1$*.*

Proof. By definition of GetFunParamAssign. □

Lemma 5.2.4. *Given map* ψ*, pointer type* $ty \in \{a \text{ const } bty*, a\ bty*\}$*,* $\widehat{ty} \in \{\text{const } \widehat{bty}*, \widehat{bty}*\}$*, and pointer data structure* $[1, [(l, \mu)], [1], i]$*,* $[1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}]$*, if* $\mathrm{EncodePtr}(ty, [1, [(l, \mu)], [1], i]) = \omega$*,* $ty \cong \widehat{ty}$*,* $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$*, then* $\mathrm{EncodePtr}(\widehat{ty}, [1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}]) = \widehat{\omega}$ *such that* $\omega \cong_\psi \widehat{\omega}$*.*

Proof. By definition of Algorithm EncodePtr, EncodePtr, and function Erase. □

Lemma 5.2.5. *Given map* ψ*, type* $ty \in \{a\ bty\}$*,* $\widehat{bty}$*, and value* n*,* $\widehat{n}$*, if* $\mathrm{EncodeVal}(ty, n) = \omega$*,* $n \cong \widehat{n}$*,* $ty \cong \widehat{bty}$*, then* $\mathrm{EncodeVal}(\widehat{bty}, \widehat{n}) = \widehat{\omega}$ *such that* $\omega \cong_\psi \widehat{\omega}$*.*

Proof. By definition of Algorithm EncodeVal, EncodeVal, and definition of function Erase. □

Lemma 5.2.6. *Given map* ψ*, type* $ty \in \{a\ bty\}$*,* $\widehat{bty}$*, value* n*,* $\widehat{n}$*,* i_1, i_2*,* $\hat{i}_1, \hat{i}_2$*, if* $\mathrm{EncodeArr}(ty, i_1, i_2, n) = \omega$*,* $n \cong_\psi \widehat{n}$ $i_1 = \hat{i}_1$*,* $i_2 = \hat{i}_2$*, and* $ty \cong_\psi \widehat{bty}$*, then* $\mathrm{EncodeArr}(\widehat{bty}, \hat{i}_1, \hat{i}_2, v) = \widehat{\omega}$ *such that* $\omega \cong_\psi \widehat{\omega}$*.*

Proof. By definition of Algorithm EncodeArr, EncodeArr, and the definition of function Erase. □

Lemma 5.2.7. *Given map* ψ*, statement* $s, \widehat{s}$*, value* n*, and parameter list* $\overline{p}, \widehat{\overline{p}}$*, if* $\mathrm{EncodeFun}(s,\ n,\ \overline{p}) = \omega$*,* $s \cong_\psi \widehat{s}$*, and* $\overline{p} \cong_\psi \widehat{\overline{p}}$*, then* $\mathrm{EncodeFun}(\widehat{s}, \Box, \widehat{\overline{p}}) = \widehat{\omega}$ *such that* $\omega \cong_\psi \widehat{\omega}$*.*

Proof. By definition of Algorithm EncodeFun, EncodeFun, and the definition of Erase. □

Lemma 5.2.8. *Given map* ψ*, type* $a\ bty, \widehat{bty}$*, and byte representation* $\omega, \widehat{\omega}$*, if* $\mathrm{DecodeVal}(a\ bty, \omega) = n$*,* $a\ bty \cong \widehat{bty}$ *and* $\omega \cong_\psi \widehat{\omega}$*, then* $\mathrm{DecodeVal}(\widehat{bty}, \widehat{\omega}) = \widehat{n}$ *and* $n \cong_\psi \widehat{n}$*.*

Proof. By case analysis of the semantics, Lemma 5.2.5, definition of Algorithm DecodeVal, DecodeVal and function Erase. □

Lemma 5.2.9. *Given map* ψ*, type* $a\ bty, \widehat{bty}$*, index* $i, \widehat{i}$*, and byte representation* $\omega, \widehat{\omega}$*, if* $\mathrm{DecodeArr}(a\ bty,\ i\ \omega) = n$*,* $a\ bty \cong \widehat{bty}$*,* $i \cong_\psi \widehat{i}$*, and* $\omega \cong_\psi \widehat{\omega}$*, then* $\mathrm{DecodeArr}(\widehat{bty}, \widehat{i}, \widehat{\omega}) = \widehat{n}$ *and* $n \cong_\psi \widehat{n}$*.*

Proof. By case analysis of the semantics, Lemma 5.2.6, definition of Algorithm DecodeArr, DecodeArr and function Erase. □

Lemma 5.2.10. *Given map ψ, type a bty, $\widehat{bty}$, index $i \in \{0...\alpha - 1\}, \widehat{i} \in \{0...\widehat{\alpha} - 1\}$, and byte representation $\omega, \widehat{\omega}$, if $\forall i \in \{0...\alpha - 1\}$ $\mathrm{DecodeArr}(a\ bty,\ i\ \omega) = n_i$, a bty $\cong \widehat{bty}$, $\alpha = \widehat{\alpha}$, and $\omega \cong_\psi \widehat{\omega}$, then $\forall \widehat{i} \in \{0...\widehat{\alpha} - 1\}$ $\mathrm{DecodeArr}(\widehat{bty}, \widehat{i}, \widehat{\omega}) = \widehat{n}_{\widehat{i}}$ such that $\forall i \in \{0...\alpha - 1\}$ $n_i \cong_\psi \widehat{n}_i$.*

Proof. By case analysis of the semantics, Lemma 5.2.6, definition of Algorithm DecodeArr, DecodeArr and function Erase. □

Lemma 5.2.11. *Given map ψ, type a bty$*$, $\widehat{bty}*$, number of locations $\alpha, 1$, and byte representation $\omega, \widehat{\omega}$, if $\mathrm{DecodePtr}(a\ bty*,\ \alpha\ \omega) = [\alpha, \overline{l}, \overline{j}, i]$, a bty$* \cong \widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, then $\mathrm{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}]$ such that $[\alpha, \overline{l}, \overline{j}, i] \cong_\psi [1, [(\widehat{l}, \widehat{\mu})], [1], \widehat{i}]$.*

Proof. By case analysis of the semantics, Lemma 5.2.4, definition of Algorithm DecodePtr, DecodePtr and function Erase. □

Lemma 5.2.12. *Given map ψ, type a const bty$*$, const $\widehat{bty}*$, number of locations $\alpha, 1$, and byte representation $\omega, \widehat{\omega}$, if $\mathrm{DecodePtr}(a\ \mathrm{const}\ bty*,\ \alpha\ \omega) = [\alpha, \overline{l}, \overline{j}, i]$, a const bty$* \cong$ const $\widehat{bty}*$, and $\omega \cong_\psi \widehat{\omega}$, then $\mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}, 0)], [1], \widehat{i}]$ such that $[1, [(l, 0)], [1], i] \cong_\psi [1, [(\widehat{l}, 0)], [1], \widehat{i}]$ and $l = \widehat{l}$.*

Proof. By case analysis of the semantics, Lemma 5.2.4, definition of Algorithm DecodePtr, DecodePtr and function Erase.

We obtain that $l = \widehat{l}$ for a constant pointer (array) type by case analysis of the semantics, showing that the location that a constant pointer cannot be changed after it is declared. □

Lemma 5.2.13. *Given map ψ and byte representation $\omega, \widehat{\omega}$, if $\mathrm{DecodeFun}(\omega) = (s, n, \overline{p})$, and $\omega \cong_\psi \omega$, then $\mathrm{DecodeFun}(\widehat{\omega}) = (\widehat{s}, \square, \widehat{\overline{p}})$, $s \cong_\psi \widehat{s}$ and $\overline{p} \cong_\psi \widehat{\overline{p}}$.*

Proof. By case analysis of the semantics, Lemma 5.2.7, definition of Algorithm DecodeFun and DecodeFun, and definition of function Erase. □

Lemma 5.2.14. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, value $n, \widehat{n}$, and type a bty, $\widehat{bty}$, if $\mathrm{UpdateVal}(\sigma_1, l, n, a\ bty) = \sigma_2$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l \cong_\psi \widehat{l}$, $n \cong_\psi \widehat{n}$, and a bty $\cong \widehat{bty}$, then $\mathrm{UpdateVal}(\widehat{\sigma}_1, \widehat{l}, \widehat{n}, \widehat{bty}) = \widehat{\sigma}_2$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.*

Proof. By definition of Algorithms UpdateVal, UpdateVal, and Erase. □

Lemma 5.2.15. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, value n, $\widehat{n}$, index $i, \widehat{i} \in \{0...\alpha - 1\}$, and type a bty, $\widehat{bty}$, if $\mathrm{UpdateArr}(\sigma_1,\ (l, i),\ n,\ a\ bty) = \sigma_2$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l = \widehat{l}$, $i = \widehat{i}$, $n \cong_\psi \widehat{n}$, and a bty $\cong_\psi \widehat{bty}$, then $\mathrm{UpdateArr}(\widehat{\sigma}_1,\ (\widehat{l}, \widehat{i}),\ \widehat{n},\ \widehat{bty}) = \sigma_2$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.*

Proof. By definition of Algorithms UpdateArr, UpdateArr, and Erase. □

Lemma 5.2.16. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, list of values $[n_0, ..., n_{\alpha-1}]$, $[\widehat{n}_0, ..., \widehat{n}_{\widehat{\alpha}-1}]$, and type $a\ bty, \widehat{bty}$, if $\forall i \in \{0...\alpha-1\}$ $\mathrm{UpdateArr}(\sigma_{1+i},\ (l, i),\ n_i,\ a\ bty) = \sigma_{2+i}$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $l = \widehat{l}$, $\alpha = \widehat{\alpha}$, $[n_0, ..., n_{\alpha-1}] \cong_\psi [\widehat{n}_0, ..., \widehat{n}_{\widehat{\alpha}-1}]$, and $a\ bty \cong_\psi \widehat{bty}$, then $\forall \widehat{i} \in \{0...\widehat{\alpha}-1\}$ $\mathrm{UpdateArr}(\widehat{\sigma}_{1+\widehat{i}},\ (\widehat{l}, \widehat{i}),\ \widehat{n}_{\widehat{i}},\ \widehat{bty}) = \sigma_{2+\widehat{i}}$ such that $(\gamma, \sigma_{2+i}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_{2+\widehat{i}})$.*

Proof. By definition of Algorithms UpdateArr, UpdateArr, and Erase, and Lemma 5.2.15.

Lemma 5.2.15 gives us that this holds when updating a single value within an array. Given that we have α values and are updating each of them sequentially, we have that each intermediate step i maintains $(\gamma, \sigma_{2+i}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_{2+i})$, and therefore the final memory maintains $(\gamma, \sigma_{2+\alpha-1}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_{2+\alpha-1})$. □

Lemma 5.2.17. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, location $(l, \mu), (\widehat{l}, \widehat{\mu})$, pointer data structure $[\alpha, \bar{l}, \bar{j}, i]$, $[1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, and type $a\ bty*, \widehat{bty}*$, if $\mathrm{UpdatePtr}(\sigma,\ (l, \mu),\ [\alpha,\ \bar{l},\ \bar{j},\ i],\ a\ bty*) = (\sigma_1, j)$, $a\ bty* \cong \widehat{bty}*$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$, and $[\alpha, \bar{l}, \bar{j}, i] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, then $\mathrm{UpdatePtr}(\widehat{\sigma},\ (\widehat{l}, \widehat{\mu}),\ [1,\ [(\widehat{l}_1, \widehat{\mu}_1)],\ [1],\ i],\ \widehat{bty}*) = (\widehat{\sigma}_1, \widehat{j})$ such that $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$ and $j = \widehat{j}$.*

Proof. By definition of UpdatePtr, UpdatePtr, and Erase, as well as Definition 5.2.15, 5.2.14, and 5.2.4. □

Lemma 5.2.18. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, memory block identifier $l, \widehat{l}$, type $a\ bty, \widehat{bty}$, and array index $i, \widehat{i}$ and size $n, \widehat{n}$, if $\mathrm{ReadOOB}(i,\ n,\ l,\ a\ bty,\ \sigma) = (n,\ j, (l_1, \mu))$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $i = \widehat{i}$, $n = \widehat{n}$, $l = \widehat{l}$, and $a\ bty \cong \widehat{bty}$, then $\mathrm{ReadOOB}(\hat{i},\ \widehat{n},\ \widehat{l},\ \widehat{bty},\ \widehat{\sigma}) = (\widehat{v},\ \widehat{j})$ such that $n \cong_\psi \widehat{n}$ and $j = \widehat{j}$.*

Proof. By definition of ReadOOB, if the number returned with the updated memory is 1, then the out of bounds access was *well-aligned* by Definition 5.2.3. Therefore, when we iterate over the *ψ-congruent* Vanilla C memory, the resulting out of bounds access will also be *well-aligned*. We use the definition of ReadOOB, ReadOOB, and Erase to help prove this. □

Lemma 5.2.19. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, type $a\ bty, \widehat{bty}$, value $n, \widehat{n}$, array index $i, \widehat{i}$ and size $\alpha, \widehat{\alpha}$, if $\mathrm{WriteOOB}(n, i, \alpha, l, a\ bty, \sigma_1) = (\sigma_2, j, (l_2, \mu))$, $n \cong_\psi \widehat{n}$, $i = \widehat{i}$, $\alpha = \widehat{\alpha}$, $l = \widehat{l}$, $a\ bty \cong_\psi \widehat{bty}$, and $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, then $\mathrm{WriteOOB}(\widehat{n},\ \hat{i},\ \widehat{\alpha},\ \widehat{l},\ \widehat{bty},\ \widehat{\sigma}_1) = (\widehat{\sigma}_2,\ \widehat{j})$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ and $j \cong \widehat{j}$.*

Proof. Proof Idea:

By definition of WriteOOB, if the number returned with the updated memory is 1, then the out of bounds access was *well-aligned* by Definition 5.2.3. Therefore, when we iterate over the *ψ-congruent* Vanilla C memory, the resulting out of bounds access will also be *well-aligned*. We use the definition of WriteOOB, WriteOOB, and Erase to help prove this. □

Lemma 5.2.20. *Given map ψ, location $(l_1, \mu_1), (\widehat{l_1}, \widehat{\mu_1})$, type $ty, \widehat{ty}$, number $n, \widehat{n}$, environment $\gamma, \widehat{\gamma}$, and memory $\sigma, \widehat{\sigma}$, if* $\mathrm{GetLocation}((l_1,\ \mu_1),\ n, \sigma) = ((l_2, \mu_2), j)$, $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu_1})$, $ty \cong \widehat{ty}$, $\tau(ty) = n$, $\tau(\widehat{ty}) = \widehat{n}$, *and* $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, *then* $\mathrm{GetLocation}((\widehat{l_1},\ \widehat{\mu_1}),\ \widehat{n}, \widehat{\sigma}) = ((\widehat{l_2},\ \widehat{\mu_2}), \widehat{j})$ *such that* $(l_2, \mu_2) \cong_\psi (\widehat{l_2}, \widehat{\mu_2})$ *and* $j \cong \widehat{j}$.

Proof. By definition of algorithms GetLocation and Erase and Definition 5.2.14. □

Lemma 5.2.21. *Given location list $\bar{l}$, location $(\widehat{l}, \widehat{\mu})$, type $ty, \widehat{ty}$, number $n, \widehat{n}$, map ψ, environment $\gamma, \widehat{\gamma}$, and memory $\sigma, \widehat{\sigma}$, if* $\mathrm{IncrementList}(\bar{l}, n, \sigma) = (\bar{l}', j)$, $\mathrm{DeclassifyPtr}([\alpha,\ \bar{l},\ \bar{j},\ i],\ ty) = (l_1, \mu_1)$ *such that* $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu_1})$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $ty \cong \widehat{ty}$, $\tau(ty) = n$, $\tau(\widehat{ty}) = \widehat{n}$, *and* $\mathrm{DeclassifyPtr}([\alpha,\ \bar{l}',\ \bar{j},\ i],\ \mathrm{private}\ bty*) = (l_2, \mu_2)$, *then* $\mathrm{GetLocation}((\widehat{l_1}, \widehat{\mu_1}), \widehat{n}, \widehat{\sigma}) = ((\widehat{l_2}, \widehat{\mu_2}), \widehat{j})$ *such that* $(l_2, \mu_2) \cong_\psi (\widehat{l_2}, \widehat{\mu_2})$ *and* $j \cong \widehat{j}$.

Proof. By definition of algorithms IncrementList, GetLocation, and Erase and Definitions 5.2.14 and 5.2.15. □

Lemma 5.2.22. *Given map ψ, memory $\sigma, \widehat{\sigma}$ and environment $\gamma, \widehat{\gamma}$ such that $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and memory block identifier $l, \widehat{l}$, if* $\mathrm{Free}(\sigma, l, \gamma) = \sigma_1$ *and* $l \cong_\psi \widehat{l}$, *then* $\mathrm{Free}(\widehat{\sigma},\ \widehat{l}, \widehat{\gamma}) = \widehat{\sigma}_1$ *such that* $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$.

Proof. By definition of Free, the ψ-congruent location will be marked as deallocated. □

Lemma 5.2.23. *Given type ty, $\widehat{ty}$ and value n, $\widehat{n}$, if* $n_1 = \mathrm{Cast}(\mathrm{public}, ty, n)$, $ty \cong_\psi \widehat{ty}$, *and* $n = \widehat{n}$ *then* $\widehat{n}_1 = \mathrm{Cast}(\mathrm{public},\ \widehat{ty},\ \widehat{n})$ *such that* $n_1 = \widehat{n}_1$.

Proof. By definition of algorithm Cast and Cast. □

Lemma 5.2.24. *Given map ψ, type ty, $\widehat{ty}$ and number $n, \widehat{n}$, if* $n_1 = \mathrm{Cast}(\mathrm{private},\ ty,\ n)$, $ty \cong_\psi \widehat{ty}$, *and* $n \cong_\psi \widehat{n}$ *then* $\widehat{n}_1 = \mathrm{Cast}(\mathrm{public},\ \widehat{ty},\ \widehat{n})$ *such that* $n_1 \cong_\psi \widehat{n}_1$.

Proof. By definition of algorithms Cast and Cast and function Erase. □

Lemma 5.2.25. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, type $a\ bty, \widehat{bty}$, and location $(l_1, \mu_1), (\widehat{l_1}, \widehat{\mu_1})$, if* $\mathrm{DerefPtr}(\sigma, a\ bty, (l_1, \mu_1)) = (n, j)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $a\ bty \cong \widehat{bty}$, *and* $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu_1})$, *then* $(\widehat{n}, \widehat{j}) = \mathrm{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l_1}, \widehat{\mu_1}))$ *such that* $n \cong_\psi \widehat{n}$ *and* $j \cong \widehat{j}$.

Proof. By definition of Algorithms DerefPtr, DerefPtr, and Erase. □

Lemma 5.2.26. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, type* $\mathrm{public}\ bty*, \widehat{bty}*$, *and location (l_1, μ_1), $(\widehat{l_1}, \widehat{\mu_1})$, if* $\mathrm{DerefPtrHLI}(\sigma, a\ bty*, (l_1, \mu_1)) = ([\alpha, \bar{l}, \bar{j}, i-1], j)$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, $a\ bty* \cong \widehat{bty}*$, *and* $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu_1})$, *then* $([1, [(\widehat{l_2}, \widehat{\mu_2})], [1], \widehat{i}-1], \widehat{j}) = \mathrm{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty}*, (\widehat{l_1}, \widehat{\mu_1}))$ *such that* $[\alpha, \bar{l}, \bar{j}, i-1] \cong_\psi [1, [(\widehat{l_2}, \widehat{\mu_2})], [1], \widehat{i}-1]$ *and* $j \cong \widehat{j}$.

Proof. By definition of Algorithms DerefPtrHLI, DerefPtrHLI, and Erase. □

Lemma 5.2.27. *Given map ψ, environment γ, $\widehat{\gamma}$, memory σ_1, $\widehat{\sigma}_1$, location $(l,\ \mu)$, $(\widehat{l},\ \widehat{\mu})$, value n, $\widehat{n}$, and type a bty, $\widehat{bty}$, if* $\mathrm{UpdateOffset}(\sigma_1, (l, \mu), n, a\ bty) = (\sigma_2, j)$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $(l, \mu) \cong_\psi (\widehat{l}, \widehat{\mu})$, $n \cong_\psi \widehat{n}$, *and a bty* $\cong_\psi \widehat{bty}$, *then* $\mathrm{UpdateOffset}(\widehat{\sigma}_1, (\widehat{l}, \widehat{\mu}), \widehat{n}, \widehat{bty}) = (\widehat{\sigma}_2, \widehat{j})$ *such that* $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$ *and* $j \cong j'$.

Proof. By definition of Algorithm UpdateOffset, UpdateOffset, and Erase, as well as Definition 5.2.14 and 5.2.4. □

Lemma 5.2.28. *Given map ψ, memory $\{\sigma_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}, \widehat{\sigma}_1$, environment $\{\gamma_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}, \widehat{\gamma}_1$, variable list x_{list}, value $\{n^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and accumulator* acc, *if* $\{\mathrm{InitializeVariables}(x_{list}, \gamma_1^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, n^{\mathrm{p}}, \mathrm{acc} + 1) = (\gamma_2^{\mathrm{p}}, \sigma_2^{\mathrm{p}}, \bar{l}_2^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{(\gamma_1^{\mathrm{p}},\ \sigma_1^{\mathrm{p}}) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$, *then* $\{\gamma_2^{\mathrm{p}} = \gamma_1^{\mathrm{p}} :: \gamma_{temp}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\sigma_2^{\mathrm{p}} = \sigma_1^{\mathrm{p}} :: \sigma_{temp}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, *and* $\{(\gamma_2^{\mathrm{p}},\ \sigma_2^{\mathrm{p}}) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Proof. By analysis of Algorithm InitializeVariables, we can see that we do not modify any elements currently in the environment of memory, we only add new mappings for our temporary variables used to for tracking and resolution. Given this, that the original SMC^2 environment and memory pairs were ψ-congruent to the Vanilla C pair, and the definition of Algorithm Erase, we have that the updated SMC^2 environment and memory pair that is returned from Algorithm InitializeVariables is still ψ-congruent to the Vanilla C pair. □

Lemma 5.2.29. *Given statements s, if $s_1 \in s$ modifies memory at a* constant location*, then that location is dictated by a given variable x.*

Proof. Proof by case analysis of the semantics and Definitions 5.2.30 and 5.2.29, we can show that all modifications to memory that are at a *constant location* are able to be found and tracked using the variable x that refers to that location. □

Lemma 5.2.30. *Given statement s, if there exists a possible evaluation of s that results an update to memory that at a* non-constant location*, then s is found by a case in Algorithm* Extract *and the tag j returned by Algorithm* Extract *is returned as 1.*

Proof. By Definition 5.2.29 and case analysis of our semantics, we have statements $*x = e$ and $x[e_1] = e_2$ where $(e_1) \nvdash \gamma$ as the only statements that could possibly lead to updating memory at a *non-constant location*.

By definition of Algorithm Extract, we can see that such statements will always be found and result in the tag being set to 1. We can also show that once the tag is set to 1, it cannot be set back to 0, and therefore will be returned as 1. □

Lemma 5.2.31. *Given statement s, if any possible evaluation of $s_1 \in s$ results in an update to memory, then s_1 is found by a case in Algorithm* Extract *and either*

- *s_1 results in an update to a* non-constant location *and so the tag is set as 1,*
- *s_1 results in an update to a* constant location *dictated by x that is local, or*

- s_1 *results in an update to a* constant location *dictated by* x *and* x *is added to the variable list* x_{list}.

Proof. Proof by contradiction showing there does not exist a statement that can be evaluated via any of our rules and result in a modification in memory that is not found by one of the cases in Algorithm Extract.

By Lemma 5.2.30, we have bullet 1. By Lemma 5.2.29 and analysis of Algorithm Extract, we have bullets 2 and 3. □

Lemma 5.2.32. *Given statements* s_1, s_2*, and environment* γ*, if* $\{\mathrm{Extract}(s_1,\ s_2, \gamma^{\mathrm{p}}) = (x_{list}, 0)\}^{\mathrm{q}}_{\mathrm{p}=1}$ *then the evaluation of* s_1 *and* s_2 *can only result in updates to memory at constant locations, each dictated by variable* x *such that* $x \in x_{list}$.

Proof. By Lemma 5.2.31, we can see that as long as the tag is not returned as 1, this holds and therefore there are no updates in memory to non-local variables that will occur in either branch that cannot be caught by variable tracking. □

Lemma 5.2.33. *Given variable list* x_{list}*, environment* $\{\gamma^{\mathrm{p}}_1\}^{\mathrm{q}}_{\mathrm{p}=1}$*, memory* $\{\sigma^{\mathrm{p}}_1\}^{\mathrm{q}}_{\mathrm{p}=1}$*, value* $\{n^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$*, and accumulator* acc*, if all updates to memory in either branch will be caught by variables* $x \in x_{list}$ *and* $\{\mathrm{InitializeVariables}(x_{list}, \gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_1,\ n^{\mathrm{p}}, \mathrm{acc}) = (\gamma^{\mathrm{p}}_2, \sigma^{\mathrm{p}}_2, \bar{l}^{\mathrm{p}})\}^{\mathrm{q}}_{\mathrm{p}=1}$*, then* $\{\forall x \in x_{list},\ (\gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_1) \models (x \equiv v_x_orig^{\mathrm{p}})\}^{\mathrm{q}}_{\mathrm{p}=1}$ *and* $\{\forall x \in x_{list}\ (\gamma^{\mathrm{p}}_2, \sigma^{\mathrm{p}}_2) \models (x_else_\mathrm{acc} \equiv v_x_orig^{\mathrm{p}})\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. By Lemma 5.2.32 we have all updates to memory in either branch will be caught by variables $x \in x_{list}$. By Definition 5.2.34 and given when Algorithm InitializeVariables is called, the current values of each x will be the original values for the variable. By definition of Algorithm InitializeVariables, we have that original values of x are stored in the temporary **`else`** variable corresponding to x. □

Lemma 5.2.34. *Given evaluation* $((1, \gamma^1_1, \sigma^1_{start}, \Delta^1_1, \mathrm{acc} + 1, s)\ \ \|\ ...\ \|\ \ (\mathrm{q}, \gamma^{\mathrm{q}}_1,\ \sigma^{\mathrm{q}}_{start},\ \Delta^{\mathrm{q}}_1,\ \mathrm{acc} + 1, s)) \Downarrow^{\mathcal{L}}_{\mathcal{D}} ((1, \gamma^1_2, \sigma^1_{end},\ \Delta^1_2,\ \mathrm{acc} + 1, \mathrm{skip})\ \ \|\ ...\ \|\ \ (\mathrm{q}, \gamma^{\mathrm{q}}_2, \sigma^{\mathrm{q}}_{end}, \Delta^{\mathrm{q}}_2, \mathrm{acc} + 1, \mathrm{skip}))$*, if* $\{\sigma^{\mathrm{p}}_{start} = \sigma^{\mathrm{p}}_1 :: \sigma^{\mathrm{p}}_{temp}\}^{\mathrm{q}}_{\mathrm{p}=1}$ *such that* $\{\sigma^{\mathrm{p}}_{temp}\}^{\mathrm{q}}_{\mathrm{p}=1}$ *is the portion containing the temporary variables for this level of nesting designated by* acc*, then* $\{\sigma^{\mathrm{p}}_{end} = \sigma^{\mathrm{p}}_2 :: \sigma'^{\mathrm{p}}_{temp}\}^{\mathrm{q}}_{\mathrm{p}=1}$ *such that* $\{\sigma'^{\mathrm{p}}_{temp} = \sigma^{\mathrm{p}}_{temp}\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. Using case analysis of the semantics, it is clear that the temporary variables given used by the Private If Else rules can only be modified during execution of a Private If Else rule. It is also clear that each level of nesting will increase the accumulator acc, and given this is appended to each of the temporary variables, it is clear that there can be no overlap of temporary variable names between levels of nesting, and so the only rule that can modify the temporary variables is the one of the level at which they were created. Therefore, we have that given the execution of one of the branches, the temporary variables used for tracking remain unchanged in the execution of that branch, or $\{\sigma^{\mathrm{p}}_{end} = \sigma^{\mathrm{p}}_2 :: \sigma^{\mathrm{p}}_{temp}\}^{\mathrm{q}}_{\mathrm{p}=1}$ such that $\{\sigma^{\mathrm{p}}_{temp}\}^{\mathrm{q}}_{\mathrm{p}=1}$ remains unchanged. □

Lemma 5.2.35. *Given environment* $\{\gamma_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\gamma}$, ***then*** *branch memory* $\{\sigma_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\sigma}_1$, *original memory* $\{\sigma_{orig}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\sigma}_{orig}$, *variable list* x_{list}, *and accumulator* acc, *if all updates to memory in either branch will be caught by variables* $x \in x_{list}$, $\{\mathrm{RestoreVariables}(x_{list},\ \gamma_1^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, \mathrm{acc}) = (\sigma_2^{\mathrm{p}}, \bar{l}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\forall x \in x_{list},\ (\gamma_1^{\mathrm{p}}, \sigma_1^{\mathrm{p}}) \models (x_else_\mathrm{acc} \equiv v_x_orig^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, $(\gamma_1^{\mathrm{p}}, \sigma_1^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$, *and* $(\gamma_1^{\mathrm{p}}, \sigma_{orig}^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_{orig})$,
then $\{\forall x \in x_{list},\ (\gamma_1^{\mathrm{p}}, \sigma_1^{\mathrm{p}}) \models (x \equiv v_x_then^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\forall x \in x_{list}\ (\gamma_2^{\mathrm{p}}, \sigma_2^{\mathrm{p}}) \models (x_then_\mathrm{acc} \equiv v_x_then^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{\forall x \in x_{list},\ (\gamma_2^{\mathrm{p}}, \sigma_2^{\mathrm{p}}) \models (x \equiv v_x_orig^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ *such that* $\{\sigma_2^{\mathrm{p}} = \sigma_{orig}^{\mathrm{p}} :: \sigma_{temp}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{(\gamma_1^{\mathrm{p}}, \sigma_2^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_{orig})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Proof. By Lemma 5.2.32 we have that all variables x that will be modified are contained in the variable list x_{list}. By Lemma 5.2.33, we have that all variables x within variable list x_{list} will have a **`then`** and **`else`** temporary created, and the **`else`** temporary stores the original value of x. By Lemma 5.2.34, we have that the temporary variables will remain unchanged throughout the execution of the **`then`** branch statement, and therefore the **`else`** temporary still stores the original values. Given $(\gamma_1^{\mathrm{p}}, \sigma_{orig}^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})$, we have that the original memories are ψ-congruent.

By Definition 5.2.34 and given when Algorithm RestoreVariables is called, the current values of each x will be the **`then`** values for the variable. By definition of Algorithm RestoreVariables, we will store the **`then`** values into the **`then`** temporaries, and then restore the original values (stored in the **`else`** temporaries) back into memory for x. We will then have the resulting memory as the original memory plus our temporaries ($\{\sigma_2^{\mathrm{p}} = \sigma_{orig}^{\mathrm{p}} :: \sigma_{temp}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$). By definition of Algorithm Erase, we will therefore have the resulting SMC^2 environment and memory pair ψ-congruent to the original Vanilla C environment and memory pair. □

Lemma 5.2.36. *Given the evaluation of a Private If Else rule, if* $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n^1)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}}))$, $\{\mathrm{ResolveVariables_Retrieve}(x_{list},\ \mathrm{acc}+1, \gamma_1^{\mathrm{p}}, \sigma_5^{\mathrm{p}}) = ([(v_{t1}^{\mathrm{p}}, v_{e1}^{\mathrm{p}}), ..., (v_{tm}^{\mathrm{p}}, v_{em}^{\mathrm{p}})], n'^{\mathrm{p}}, \bar{l}_6^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, *and* $\{n^{\mathrm{p}} \cong \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *then* $\{n^{\mathrm{p}} = n'\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *such that* $\{n'^{\mathrm{p}} \cong \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Proof. By definition of Algorithm InitializeVariables, the results $\{n^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ from the evaluation of the private conditional e will be stored in temporary variables based on the level of nesting indicated by the accumulator acc. By Lemma 5.2.34, we have that these temporaries cannot be modified by the evaluation of either branch statements s_1, s_2. By definition of Algorithm RestoreVariables, we have that these temporaries cannot be modified during the evaluation of Algorithm RestoreVariables. Therefore, when we retrieve these values from memory using Algorithm ResolveVariables_Retrieve, they will be identical to the values we stored into memory using Algorithm InitializeVariables. □

Lemma 5.2.37. *Given environment* $\{\gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, ***else*** *branch memory* $\{\sigma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, *variable list* x_{list}, *and accumulator* acc, *if all updates to memory in either branch will be caught by variables* $x \in x_{list}$ *and* $\{\mathrm{ResolveVariables_Retrieve}(x_{list},\ \mathrm{acc}+1, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) = ([(v_{t1}^{\mathrm{p}}, v_{e1}^{\mathrm{p}}), ..., (v_{tm}^{\mathrm{p}}, v_{em}^{\mathrm{p}})], n^{\mathrm{p}}, \bar{l}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, *and* $\forall x \in x_{list}, \mathrm{p} \in$

$\{1...\mathrm{q}\}$, $(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \models (x_then_\mathrm{acc} \equiv v_x_then^{\mathrm{p}})$, *then* $\{\forall x_i \in x_{list}, (\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \models (x_i \equiv v^{\mathrm{p}}_{ei})\}^{\mathrm{q}}_{\mathrm{p}=1}$, *and* $\{\forall x_i \in x_{list}, (\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \models (x_i_then_\mathrm{acc} \equiv v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. Given $\forall x \in x_{list}, \mathrm{p} \in \{1...\mathrm{q}\}$, $(\gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_5) \models (x_then_\mathrm{acc} \equiv v_x_then^{\mathrm{p}})$ by Lemma 5.2.34, we have that all of the **`then`** temporary variables currently store the result of the then branch. By definition of Algorithm ResolveVariables_Retrieve, this is what is returned for each variable x_i in value v^{p}_{ti}.

Given that we are executing Algorithm ResolveVariables_Retrieve with the resulting memory from the **`else`** branch, by definition of Algorithm ResolveVariables_Retrieve this is what is returned for each variable x_i in value v^{p}_{ei}. □

Lemma 5.2.38. *Given variable list* x_{list}*, accumulator* acc*, environment* $\{\gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\gamma}$, ***`else`*** *branch memory* $\{\sigma^{\mathrm{p}}_e\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\sigma}_e$*, and values* $\{[v^{\mathrm{p}}_{f1}, ..., v^{\mathrm{p}}_{fm}], [v^{\mathrm{p}}_{e1}, ..., v^{\mathrm{p}}_{em}]\}^{\mathrm{q}}_{\mathrm{p}=1}$*, if all updates to memory in either branch will be caught by variables* $x \in x_{list}$, $\{\mathrm{ResolveVariables_Store}(x_{list},\ \mathrm{acc}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_e, [v^{\mathrm{p}}_{f1}, ..., v^{\mathrm{p}}_{fm}]) = (\sigma^{\mathrm{p}}_f, \bar{l}^{\mathrm{p}})\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_e) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_e)\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{\forall x_i \in x_{list}, (\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_e) \models (x_i \equiv v^{\mathrm{p}}_{ei})\}^{\mathrm{q}}_{\mathrm{p}=1}$, *and* $\{\forall i \in \{1...m\}, v^{\mathrm{p}}_{fi} = v^{\mathrm{p}}_{ei})\}^{\mathrm{q}}_{\mathrm{p}=1}$, *then* $\{\forall x \in x_{list}, (\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_f) \models (x \equiv v^{\mathrm{p}}_{ei})\}^{\mathrm{q}}_{\mathrm{p}=1}$ *and* $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_f) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_e)\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. Given that all changes were caught by variables in the variable list and that the final list of values given matches the **`else`** values, by definition of Algorithm ResolveVariables_Store we will iterate through the list and properly store all final values into memory for their respective variables.

Given the **`else`** environment and memory pairs we ψ-congruent, and that we are placing the **`else`** values into memory, we will have the resulting SMC^2 memory ψ-congruent to the **`else`** Vanilla C memory. □

Lemma 5.2.39. *Given variable list* x_{list}*, accumulator* acc*, environment* $\{\gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\gamma}$, ***`else`*** *branch memory* $\{\sigma^{\mathrm{p}}_e\}^{\mathrm{q}}_{\mathrm{p}=1}$ ***`then`*** *branch memory* $\{\sigma^{\mathrm{p}}_t\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\sigma}_t$*, and values* $\{[v^{\mathrm{p}}_{f1}, ..., v^{\mathrm{p}}_{fm}], [v^{\mathrm{p}}_{t1}, ..., v^{\mathrm{p}}_{tm}]\}^{\mathrm{q}}_{\mathrm{p}=1}$*, if all updates to memory in either branch will be caught by variables* $x \in x_{list}$ *and* $\{\mathrm{ResolveVariables_Store}(x_{list},\ \mathrm{acc}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_e, [v^{\mathrm{p}}_{f1}, ..., v^{\mathrm{p}}_{fm}]) = (\sigma^{\mathrm{p}}_f, \bar{l}^{\mathrm{p}})\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_t) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_t)\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{\forall x_i \in x_{list}, (\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_t) \models (x_i \equiv v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$ *and* $\{\forall i \in \{1...m\}, v^{\mathrm{p}}_{fi} = v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$, *then* $\{\forall x \in x_{list}, (\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_f) \models (x \equiv v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$ *and* $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_f) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_t)\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. Given that all changes were caught by variables in the variable list and that the final list of values given matches the **`then`** values, by definition of Algorithm ResolveVariables_Store we will iterate through the list and properly store all final values into memory for their respective variables.

Given the **`then`** environment and memory pairs we ψ-congruent, and that we are placing the **`then`** values into memory, we will have the resulting SMC^2 memory ψ-congruent to the **`then`** Vanilla C memory. □

Lemma 5.2.40. *Given statement* s_1, s_2*, environment* $\{\gamma^{\mathrm{p}}_1\}^{\mathrm{q}}_{\mathrm{p}=1}$*, memory* $\{\sigma^{\mathrm{p}}_1\}^{\mathrm{q}}_{\mathrm{p}=1}$*, value* $\{n^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$*, location map* $\{\Delta^{\mathrm{p}}_1\}^{\mathrm{q}}_{\mathrm{p}=1}$*, and accumulator* acc*, if* $\{\mathrm{Extract}(s_1, s_2, \gamma^{\mathrm{p}}_1) = (x_{list}, 1)\}^{\mathrm{q}}_{\mathrm{p}=1}$ *and* $\{\mathrm{Initialize}(\Delta^{\mathrm{p}}_1, x_{list}, \gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_1,\ n^{\mathrm{p}}, \mathrm{acc})$

$= (\gamma_2^{\mathrm{p}}, \sigma_2^{\mathrm{p}}, \Delta_2^{\mathrm{p}}, \bar{l}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, *then all updates to a* constant location *dictated by variable x will have their original value stored within location map Δ,* $\{(\gamma_2^{\mathrm{p}}, \sigma_2^{\mathrm{p}}) \models (res_\mathrm{acc} \equiv n^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, *and* $\{\sigma_2^{\mathrm{p}} = \sigma_1^{\mathrm{p}} :: \sigma_{temp1}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Proof. By Definition 5.2.29 and case analysis of our semantics, we have statements $*x = e$ and $x[e_1] = e_2$ where $(e_1) \nvdash \gamma$ as the only statements that could possibly lead to updating memory at a *non-constant location*. By Definition 5.2.30, Lemma 5.2.29, Lemma 5.2.31, and the definition of Algorithm $\mathrm{Extract}$, we can see that all updates made in other semantic rules would be dictated by a variable x and added to x_{list}. By definition of Algorithm $\mathrm{Initialize}$, we can see that all variables in x_{list} will have initial mappings of their location, original value, and type stored into location map $\{\Delta_1^{\mathrm{p}}[\mathrm{acc}]\}_{\mathrm{p}=1}^{\mathrm{q}}$, as well as added the mappings to store the result of the private condition within the temporary variable res_acc. □

Lemma 5.2.41. *Given variable list x_{list}, location map $\{\Delta_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, environment $\{\gamma_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\gamma}$, memory $\{\sigma_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\sigma}$, value $\{n^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and accumulator* acc*, if* $\{\mathrm{Initialize}(\Delta_1^{\mathrm{p}},\ x_{list},\ \gamma_1^{\mathrm{p}},\ \sigma_1^{\mathrm{p}},\ n^{\mathrm{p}}, \mathrm{acc}) = (\gamma_2^{\mathrm{p}}, \sigma_2^{\mathrm{p}}, \Delta_2^{\mathrm{p}}, \bar{l}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{(\gamma_1^{\mathrm{p}}, \sigma_1^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})\}_{\mathrm{p}=1}^{\mathrm{q}}$, *then* $\{(\gamma_2^{\mathrm{p}}, \sigma_2^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Proof. By definition of Algorithm $\mathrm{Initialize}$ and Erase. $\mathrm{Initialize}$ adds a mapping for a temporary variable to store the result of the private condition, and therefore maintains ψ-congruency with the Vanilla C environment and memory pair. □

Lemma 5.2.42. *Given configuration* $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s) \ \parallel \ C)$*, if an update is made at a* non-constant location (l, μ) *during the execution of a statement s within a private-conditioned branch, then* $(l, \mu) \in \Delta[\mathrm{acc}]$ *such that* $\Delta[\mathrm{acc}](l, \mu) = (v_{orig}, v_{then}, j, ty)$ *and* $\Delta[\mathrm{acc}]$ *is* complete.

Proof. By Definition 5.2.29 and case analysis of our semantics, we have statements $*x = e$ and $x[e_1] = e_2$ where $(e_1) \nvdash \gamma$ as the only statements that could possibly lead to updating memory at a *non-constant location*. In each such rule, either $\mathrm{DynamicUpdate}$ is called before the update or $\mathrm{WriteOOB}$ is called to perform the update, and will perform the appropriate checks and add to Δ if necessary before performing the update in memory. By definitions of Algorithms $\mathrm{DynamicUpdate}$ and $\mathrm{WriteOOB}$, we can see that we have the following cases:

- $\mathrm{acc} = 0$, and we are not inside a private-conditioned branch and therefore do not need to track anything,
- the location already exists in $\Delta[\mathrm{acc}]$, and therefore already has the initial value stored and no modification of the entry will occur within Δ, or
- the location does not exist in $\Delta[\mathrm{acc}]$, and we add it with its current value as the initial value, a null **`then`** value, tag 0, and it's expected type, then proceed to ensure it is also tracked in outer levels of nesting (if applicable).

Given these three cases, we can see that while inside a private-conditioned branch, we are either already tracking the location or we will initialize a mapping for the location, and therefore the modification will be properly tracked within Δ. By Definition 5.2.31, we have that $\Delta[\mathrm{acc}]$ is *complete*. □

Lemma 5.2.43. *Given environment* $\{\gamma_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\gamma}$, ***then*** *branch memory* $\{\sigma_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\sigma}_1$, *original memory* $\{\sigma_{orig}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\sigma}_{orig}$, *location map* Δ_1, *and accumulator* acc, *if* $\{\Delta_1^{\mathrm{p}}[\mathrm{acc}+1]\}_{\mathrm{p}=1}^{\mathrm{q}}$ *is* complete, $\{\mathrm{Restore}(\sigma_1^{\mathrm{p}}, \Delta_1^{\mathrm{p}}, \mathrm{acc}) = (\sigma_2^{\mathrm{p}}, \Delta_2^{\mathrm{p}}, \bar{l}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, $(\gamma_1^{\mathrm{p}}, \sigma_1^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$, *and* $(\gamma_1^{\mathrm{p}}, \sigma_{orig}^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_{orig})$, *then* $\{\Delta_2^{\mathrm{p}}[\mathrm{acc}+1]\}_{\mathrm{p}=1}^{\mathrm{q}}$ *is* then-complete *and* $\{(\gamma_1^{\mathrm{p}}, \sigma_2^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma} orig)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Proof. Given $\{\Delta_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ is *complete*, we can see that by Definition 5.2.31 and definition of Algorithm Restore, we will iterate through all non-local locations that were modified within **then** branch, storing the **then** value from the **then** branch memory and resetting the value in memory to be that of the original. We will set the tag to be 1 as we store each **then** value in $\{\Delta_2^{\mathrm{p}}[\mathrm{acc}+1]\}_{\mathrm{p}=1}^{\mathrm{q}}$, indicating that these locations were modified within the **then** branch and ensuring that all non-local locations will be able to be properly resolved after evaluation of the **else** branch. By Definition 5.2.32, we have that $\{\Delta_2^{\mathrm{p}}[\mathrm{acc}+1]\}_{\mathrm{p}=1}^{\mathrm{q}}$ is *then-complete*. □

Lemma 5.2.44. *Given the evaluation of a Private If Else rule, if* $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}}))$, $\{\mathrm{Extract}(s_1, s_2, \gamma^{\mathrm{p}}) = (x_{list}, 1)\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\mathrm{Initialize}(\Delta_1^{\mathrm{p}}, x_{list}, \gamma^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, n^{\mathrm{p}}, \mathrm{acc}+1) = (\gamma_1^{\mathrm{p}}, \sigma_2^{\mathrm{p}}, \Delta_2^{\mathrm{p}}, \bar{l}_2^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, $((1, \gamma_1^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}+1, s_1) \parallel ... \parallel (\mathrm{q}, \gamma_1^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}+1, s_1)) \Downarrow_{\mathcal{D}_2}^{\mathcal{L}_3} ((1, \gamma_2^1, \sigma_3^1, \Delta_3^1, \mathrm{acc}+1, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma_2^{\mathrm{q}}, \sigma_3^{\mathrm{q}}, \Delta_3^{\mathrm{q}}, \mathrm{acc}+1, \mathrm{skip}))$, $\{\mathrm{Restore}(\sigma_3^{\mathrm{p}}, \Delta_3^{\mathrm{p}}, \mathrm{acc}+1) = (\sigma_4^{\mathrm{p}}, \Delta_4^{\mathrm{p}}, \bar{l}_4^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ $((1, \gamma_1^1, \sigma_4^1, \Delta_4^1, \mathrm{acc}+1, s_2) \parallel ... \parallel (\mathrm{q}, \gamma_1^{\mathrm{q}}, \sigma_4^{\mathrm{q}}, \Delta_4^{\mathrm{q}}, \mathrm{acc}+1, s_2)) \Downarrow_{\mathcal{D}_3}^{\mathcal{L}_5} ((1, \gamma_3^1, \sigma_5^1, \Delta_5^1, \mathrm{acc}+1, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma_3^{\mathrm{q}}, \sigma_5^{\mathrm{q}}, \Delta_5^{\mathrm{q}}, \mathrm{acc}+1, \mathrm{skip}))$, $\{\mathrm{Resolve_Retrieve}(\gamma_1^{\mathrm{p}}, \sigma_5^{\mathrm{p}}, \Delta_5^{\mathrm{p}}, \mathrm{acc}+1) = ([(v_{t1}^{\mathrm{p}}, v_{e1}^{\mathrm{p}}), ..., (v_{tm}^{\mathrm{p}}, v_{em}^{\mathrm{p}})], n'^{\mathrm{p}}, \bar{l}_6^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, *and* $\{n^{\mathrm{p}} \cong_{\psi} \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *then* $\{n^{\mathrm{p}} = n'\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *such that* $\{n'^{\mathrm{p}} \cong \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Proof. By definition of Algorithm Initialize, the results $\{n^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ from the evaluation of the private conditional e will be stored in a temporary variable based on the level of nesting indicated by the accumulator acc. By definition of Algorithm Restore, this temporary does not get modified. By Lemma 5.2.34, we have that this temporary cannot be modified by the evaluation of either branch statements s_1, s_2. Therefore, when we retrieve these values from memory using Algorithm Resolve_Retrieve, they will be identical to the values we stored into memory using Algorithm Initialize, and therefore maintain ψ-congruency with the Vanilla C value. By Definition 5.2.19, given these values are not locations, we have that they are congruent, $\{n'^{\mathrm{p}} \cong \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$. □

Lemma 5.2.45. *Given environment* $\{\gamma_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, *statement* s, *memory* $\{\sigma_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, *accumulator* acc, *and location map* $\{\Delta_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, *if* $\{\Delta_1^{\mathrm{p}}[\mathrm{acc}+1]\}_{\mathrm{p}=1}^{\mathrm{q}}$ *is* then-complete, $((1, \gamma_1^1, \sigma_1^1, \Delta_2^1, \mathrm{acc}+1, s) \parallel ... \parallel (\mathrm{q}, \gamma_1^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}+1, s)) \Downarrow_{\mathcal{D}}^{\mathcal{L}} ((1, \gamma_2^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}+1, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma_2^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}+1, \mathrm{skip}))$, *and* $\{\Delta_2^{\mathrm{p}}[\mathrm{acc}+1]\}_{\mathrm{p}=1}^{\mathrm{q}}$ *is* complete, *then* $\{\Delta_2^{\mathrm{p}}[\mathrm{acc}+1]\}_{\mathrm{p}=1}^{\mathrm{q}}$ *is* else-complete

Proof. This holds by Definition 5.2.33. □

Lemma 5.2.46. *Given environment* $\{\gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, *location map* $\{\Delta^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, *accumulator* acc, ***then*** *memory* $\{\sigma_t^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, *and* ***else*** *memory* $\{\sigma_e^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, *if* $\{\Delta^{\mathrm{p}}[\mathrm{acc}]\}_{\mathrm{p}=1}^{\mathrm{q}}$ *is* else-complete *and* $\{\mathrm{Resolve_Retrieve}(\gamma^{\mathrm{p}}, \sigma_e^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}) = ([(v_{t1}^{\mathrm{p}},$

$v^{\mathrm{p}}_{e1}), ..., (v^{\mathrm{p}}_{tm}, v^{\mathrm{p}}_{em})]$, $n^{\mathrm{p}}, \bar{l}^{\mathrm{p}})\}^{\mathrm{q}}_{\mathrm{p}=1}$, *then* $\{\forall(l_i, \mu_i) = (v^{\mathrm{p}}_{oi}, v^{\mathrm{p}}_{ti}, 1, ty_i) \in \Delta^{\mathrm{p}}[\mathrm{acc}], (\sigma^{\mathrm{p}}_t) \models_l ((l_i, \mu_i) \equiv_{ty} v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{\forall(l_i, \mu_i) = (v^{\mathrm{p}}_{ti}, \mathrm{NULL}, 0, ty_i) \in \Delta^{\mathrm{p}}[\mathrm{acc}], (\sigma^{\mathrm{p}}_t) \models_l ((l_i, \mu_i) \equiv_{ty_i} v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$, *and* $\{\forall(l_i, \mu_i) = (v^{\mathrm{p}}_{oi}, v^{\mathrm{p}}_{ti}, j, ty_i) \in \Delta^{\mathrm{p}}[\mathrm{acc}], (\sigma^{\mathrm{p}}_e) \models_l ((l_i, \mu_i) \equiv_{ty_i} v^{\mathrm{p}}_{ei})\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. By definition of Algorithm Resolve_Retrieve, we can see that we pull the **`then`** value from the location map at nesting level acc based on the tag, and the **`else`** value from the given **`else`** memory. Given $\{\Delta^{\mathrm{p}}[\mathrm{acc}]\}^{\mathrm{q}}_{\mathrm{p}=1}$ is *else-complete*, we have that all `original` and **`then`** values have been properly added into $\{\Delta^{\mathrm{p}}[\mathrm{acc}]\}^{\mathrm{q}}_{\mathrm{p}=1}$. By Definitions 5.2.35 and 5.2.33, this gives us $\{\forall(l_i, \mu_i) = (v^{\mathrm{p}}_{oi}, v^{\mathrm{p}}_{ti}, 1, ty_i) \in \Delta^{\mathrm{p}}[\mathrm{acc}], (\sigma^{\mathrm{p}}_t) \models_l ((l_i, \mu_i) \equiv_{ty} v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$ and $\{\forall(l_i, \mu_i) = (v^{\mathrm{p}}_{ti}, \mathrm{NULL}, 0, ty_i) \in \Delta^{\mathrm{p}}[\mathrm{acc}], (\sigma^{\mathrm{p}}_t) \models_l ((l_i, \mu_i) \equiv_{ty_i} v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$. Given we are pulling the **`else`** value from the given **`else`** memory, we have $\{\forall(l_i, \mu_i) = (v^{\mathrm{p}}_{oi}, v^{\mathrm{p}}_{ti}, j, ty_i) \in \Delta^{\mathrm{p}}[\mathrm{acc}], (\sigma^{\mathrm{p}}_e) \models_l ((l_i, \mu_i) \equiv_{ty_i} v^{\mathrm{p}}_{ei})\}^{\mathrm{q}}_{\mathrm{p}=1}$. This gives us that the all of our **`then`** values are those from the end of the **`then`** branch, and all of our **`else`** values are those from the end of the **`else`** branch. □

Lemma 5.2.47. *Given location map* $\{\Delta^{\mathrm{p}}_1\}^{\mathrm{q}}_{\mathrm{p}=1}$, *accumulator* acc, *environment* $\{\gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\gamma}$, ***`else`*** *branch memory* $\{\sigma^{\mathrm{p}}_e\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\sigma}_e$, *and values* $\{[v^{\mathrm{p}}_{f1}, ..., v^{\mathrm{p}}_{fm}], [v^{\mathrm{p}}_{e1}, ..., v^{\mathrm{p}}_{em}]\}^{\mathrm{q}}_{\mathrm{p}=1}$, *if* $\{\Delta^{\mathrm{p}}_1[\mathrm{acc}]\}^{\mathrm{q}}_{\mathrm{p}=1}$ *is* else-complete, $\{\mathrm{Resolve_Store}(\Delta^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_e, \mathrm{acc}, [v^{\mathrm{p}}_{f1}, ..., v^{\mathrm{p}}_{fm}]) = (\sigma^{\mathrm{p}}_f, \Delta^{\mathrm{p}}_2, \bar{l}^{\mathrm{p}})\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_e) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_e)\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{\forall(l_i, \mu_i) = (v^{\mathrm{p}}_{oi}, v^{\mathrm{p}}_{ti}, j, ty_i) \in \Delta^{\mathrm{p}}_1[\mathrm{acc}], (\sigma^{\mathrm{p}}_e) \models_l ((l_i, \mu_i) \equiv_{ty_i} v^{\mathrm{p}}_{ei})\}^{\mathrm{q}}_{\mathrm{p}=1}$, *and* $\{\forall i \in \{1...m\}, v^{\mathrm{p}}_{fi} = v^{\mathrm{p}}_{ei})\}^{\mathrm{q}}_{\mathrm{p}=1}$, *then* $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_f) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_e)\}^{\mathrm{q}}_{\mathrm{p}=1}$ *and* $\{\forall(l_i, \mu_i) = (v^{\mathrm{p}}_{oi}, v^{\mathrm{p}}_{ti}, j, ty_i) \in \Delta^{\mathrm{p}}_1[\mathrm{acc}], (\sigma^{\mathrm{p}}_f) \models_l ((l_i, \mu_i) \equiv_{ty_i} v^{\mathrm{p}}_{ei})\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. Given that $\{\Delta^{\mathrm{p}}_1[\mathrm{acc}]\}^{\mathrm{q}}_{\mathrm{p}=1}$ is *else-complete*, by definition of Algorithm ResolveVariables_Store we will iterate through the list of locations and properly store all final values into memory at their respective locations.

Given the **`else`** environment and memory pairs we ψ-congruent, and that we are placing the **`else`** values into memory, we will have the resulting SMC^2 memory ψ-congruent to the **`else`** Vanilla C memory. □

Lemma 5.2.48. *Given location map* $\{\Delta^{\mathrm{p}}_1\}^{\mathrm{q}}_{\mathrm{p}=1}$, *accumulator* acc, *environment* $\{\gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\gamma}$, ***`else`*** *branch memory* $\{\sigma^{\mathrm{p}}_e\}^{\mathrm{q}}_{\mathrm{p}=1}$ ***`then`*** *branch memory* $\{\sigma^{\mathrm{p}}_t\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\sigma}_t$, *and values* $\{[v^{\mathrm{p}}_{f1}, ..., v^{\mathrm{p}}_{fm}], [v^{\mathrm{p}}_{t1}, ..., v^{\mathrm{p}}_{tm}]\}^{\mathrm{q}}_{\mathrm{p}=1}$, *if* $\{\Delta^{\mathrm{p}}_1[\mathrm{acc}]\}^{\mathrm{q}}_{\mathrm{p}=1}$ *is* else-complete, $\{\mathrm{Resolve_Store}(\Delta^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_e, \mathrm{acc}, [v^{\mathrm{p}}_{f1}, ..., v^{\mathrm{p}}_{fm}]) = (\sigma^{\mathrm{p}}_f, \Delta^{\mathrm{p}}_2, \bar{l}^{\mathrm{p}})\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_t) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_t)\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{\forall(l_i, \mu_i) = (v^{\mathrm{p}}_{oi}, v^{\mathrm{p}}_{ti}, 1, ty_i) \in \Delta^{\mathrm{p}}_5[\mathrm{acc}], (\sigma^{\mathrm{p}}_3) \models_l ((l_i, \mu_i) \equiv_{ty} v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{\forall(l_i, \mu_i) = (v^{\mathrm{p}}_{ti}, \mathrm{NULL}, 0, ty_i) \in \Delta^{\mathrm{p}}_1[\mathrm{acc}], (\sigma^{\mathrm{p}}_t) \models_l ((l_i, \mu_i) \equiv_{ty_i} v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$, *and* $\{\forall i \in \{1...m\}, v^{\mathrm{p}}_{fi} = v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$, *then* $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_f) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_t)\}^{\mathrm{q}}_{\mathrm{p}=1}$ *and* $\{\forall(l_i, \mu_i) = (v^{\mathrm{p}}_{oi}, v^{\mathrm{p}}_{ti}, 1, ty_i) \in \Delta^{\mathrm{p}}_1[\mathrm{acc}], (\sigma^{\mathrm{p}}_f) \models_l ((l_i, \mu_i) \equiv_{ty_i} v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$ *and* $\{\forall(l_i, \mu_i) = (v^{\mathrm{p}}_{ti}, \mathrm{NULL}, 0, ty_i) \in \Delta^{\mathrm{p}}_1[\mathrm{acc}], (\sigma^{\mathrm{p}}_f) \models_l ((l_i, \mu_i) \equiv_{ty_i} v^{\mathrm{p}}_{ti})\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. Given that $\{\Delta^{\mathrm{p}}_1[\mathrm{acc}]\}^{\mathrm{q}}_{\mathrm{p}=1}$ is *else-complete*, by definition of Algorithm ResolveVariables_Store we will iterate through the list of locations and properly store all final values into memory at their respective locations.

Given the **`then`** environment and memory pairs we ψ-congruent, and that we are placing the **`then`** values into memory, we will have the resulting SMC^2 memory ψ-congruent to the **`then`** Vanilla C memory. □

Lemma 5.2.49. *Given configuration* $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s) \parallel C)$*, if* $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s) \parallel C) \Downarrow_{\mathcal{D}}^{\mathcal{L}} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C)$*, then* $(l, \mu) \notin s$*.*

Proof. By case analysis of the semantics, we can see that there is no rule that will evaluate a statement containing (l, μ) to a value n. □

Lemma 5.2.50. *Given map* ψ*, configuration* $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s) \parallel C)$*, environment* $\widehat{\gamma}$*, memory* $\widehat{\sigma}$*, statement* $\widehat{s}$*, and configuration* $\widehat{C}$*, if* $(\gamma, \sigma) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})$*,* $s \cong_{\psi} \widehat{s}$*, and* $C \cong_{\psi} \widehat{C}$*, then* $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{s}) \parallel \widehat{C})$ *such that* $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s)$-$\parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{s}) \parallel \widehat{C})$*.*

Proof. By Definitions 5.2.20 and 5.2.22. □

Lemma 5.2.51. *Given values* $v, \widehat{v}$ *and environment* γ*, if* $v \cong_{\psi} \widehat{v}$ *and* $(v) \nvdash \gamma$*, then* $v = \widehat{v}$*.*

Proof. By Definitions 5.2.19, 5.2.18, and the definition of the erasure function Erase. Case analysis on $\mathrm{Erase}(v) = \widehat{v}$ gives us that $v = \widehat{v}$ when v is public. □

Lemma 5.2.52. *Given expression* e *and configuration* $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C)$ *such that* $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ v) \parallel C_1)$*, if* $(e) \nvdash \gamma$*, then* $(v) \nvdash \gamma$*.*

Proof. By definition of Algorithm 144, we have that all elements in e must be public. By case analysis on rules where $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ v) \parallel C_1)$ and $(e) \nvdash \gamma$, we find that $(v) \nvdash \gamma$ is true. □

Lemma 5.2.53. *Given map* ψ *and statement* $s, \widehat{s}$*, if* $s \cong_{\psi} \widehat{s}$ *and* $(l, \mu) \notin s$*, then* $s \cong \widehat{s}$*.*

Proof. Given that s does not contain (l, μ), by Definition 5.2.12 we have $s \cong \widehat{s}$.

This follows directly from the definition of function Erase, and can be proven by case analysis of all statements that are not locations. □

Lemma 5.2.54. *Given map* ψ *and statement* $s, \widehat{s}$*, if* $s \cong \widehat{s}$ *and* $(l, \mu) \notin s$*, then* $s \cong_{\psi} \widehat{s}$*.*

Proof. Given that s does not contain (l, μ), by Definition 5.2.20 we have $s \cong_{\psi} \widehat{s}$.

This follows directly from the definition of function Erase, and can be proven by case analysis of all statements that are not locations. □

Lemma 5.2.55. *Given map* ψ_1, ψ_2 *and statement* $s, \widehat{s}$*, if* $s \cong_{\psi_1} \widehat{s}$ *and* $(l, \mu) \notin s$*, then* $s \cong_{\psi_2} \widehat{s}$*.*

Proof. Given that s does not contain a hard-coded location (l, μ), by Lemma 5.2.53 we have that $s \cong \widehat{s}$.

Given $s \cong \widehat{s}$ and s does not contain a hard-coded location (l, μ), by Lemma 5.2.54, we have $s \cong_{\psi_2} \widehat{s}$.

This follows directly from the definition of function Erase, and can be proven by case analysis of all statements that are not locations – a statement not containing a location will maintain congruency and in turn ψ-congruency for any given map ψ. □

Lemma 5.2.56. *Given an initial map ψ, environment γ, memory σ, accumulator* acc*, and expression e, if* $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \Downarrow^{\mathcal{L}}_{\mathcal{D}} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, v) \parallel C)$ *such that $v \neq$* skip*, then* $\mathrm{pfree}(e_1) \notin e$ *and the ending map ψ_1 is equivalent to ψ.*

Proof. By definition of SMC^2 rule pfree, skip is returned from the evaluation of $\mathrm{pfree}(e_1)$. Therefore, by case analysis of the rules, if $v \neq$ skip, then $\mathrm{pfree}(e_1) \notin e$. By Definition 5.2.1, ψ is only modified after the execution of function pfree; therefore we have that $\psi_1 == \psi$. □

Lemma 5.2.57. *Given ψ and* $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{s}) \parallel \widehat{C})$*, if* $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s) \parallel C) \Downarrow^{\mathcal{L}}_{\mathcal{D}} ((\mathrm{p}, \gamma_1, \sigma_1, \Delta, \mathrm{acc}, v) \parallel C_1)$ *and* $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{s}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}} ((\widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \widehat{v}) \parallel \widehat{C})$ *such that* $((\mathrm{p}, \gamma_1, \sigma_1, \Delta, \mathrm{acc}, v) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \Box, \widehat{v}) \parallel \widehat{C}_1)$*, then* $(\gamma, \sigma_1) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$.

Proof. Proof Sketch: Proof by induction over congruent evaluations.
Using the definition of function Erase, we show that with every rule that adds to γ or adds to or modifies σ maintains both $(\gamma_1, \sigma_1) \cong_{\psi} (\widehat{\gamma}_1, \widehat{\sigma}_1)$ and $(\gamma, \sigma_1) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$ by Definition 5.2.16. □

Lemma 5.2.58 ($\mathcal{D}_1 :: \mathcal{D}_2 \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2$)**.** *Given party-wise code lists $\mathcal{D}_1, \mathcal{D}_2$, $\widehat{\mathcal{D}}_1, \widehat{\mathcal{D}}_2$ if $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$ and $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$ then $\mathcal{D}_1 :: \mathcal{D}_2 \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2$.*

Proof. By definition of Algorithm 140, the :: operation is deterministic and maintains party-wise ordering. □

Lemma 5.2.59. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, and variable name $x, \widehat{x}$, if $x \notin \gamma$, $x = \widehat{x}$, and $(\gamma, \sigma) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})$, then $\widehat{x} \notin \widehat{\gamma}$.*

Proof. By Definition 5.2.16. □

Lemma 5.2.60. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, variable name $x, \widehat{x}$, memory block identifier $l, \widehat{l}$, and type $ty, \widehat{ty}$, if $\gamma_1 = \gamma[x \rightarrow (l, ty)]$, $x = \widehat{x}$, $l = \widehat{l}$, $ty \cong \widehat{ty}$, and $(\gamma, \sigma) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})$, then $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \widehat{ty})]$ such that $(\gamma_1, \sigma) \cong_{\psi} (\widehat{\gamma}_1, \widehat{\sigma})$.*

Proof. By Definition 5.2.16 and the structure of the environment. □

Lemma 5.2.61. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, type $ty \in \{a\ bty, a$ const $bty*, a\ bty*\}, \widehat{ty}$, byte representation $\omega, \widehat{\omega}$, number $n, \widehat{n}$, and permission $perm, \widehat{perm}$, if $\sigma_2 = \sigma_1[l \rightarrow (\omega, ty, n, \mathrm{PermL}(perm, ty, a, n))]$, $(\gamma, \sigma_1) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)$, $l \cong_{\psi} \widehat{l}$, $\omega \cong_{\psi} \widehat{\omega}$, $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$, and $ty \cong \widehat{ty}$, then $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\omega, ty, \widehat{n}, \mathrm{PermL}(perm, \widehat{ty}, \mathrm{public}, \widehat{n}))]$ such that $(\gamma, \sigma_2) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_2)$.*

Proof. By Definition 5.2.16 and the structure of memory. □

Lemma 5.2.62. *Given ψ, $(\gamma, \sigma) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})$, and $x = \widehat{x}$ such that $x \in \gamma$ and $\widehat{x} \in \widehat{\gamma}$, if $\gamma(x) = (l, ty)$ then $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{ty})$, where $l = \widehat{l}$, $(l, 0) \cong_{\psi} (\widehat{l}, 0)$, and $ty \cong \widehat{ty}$.*

Proof. This holds by Definition 5.2.16 and the definition of function Erase. □

Lemma 5.2.63. *Given ψ, $\{(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})\}$, and $l \cong_\psi \widehat{l}$ such that $l \in \sigma$ and $\widehat{l} \in \widehat{\sigma}$, if $\sigma(l) = (\omega,\ ty,\ n,\ \mathrm{PermL}(perm, ty, a, n))$ and $ty \neq \mathrm{private}\ bty*$, then $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{ty}, \widehat{n}, \mathrm{PermL}(perm, \widehat{ty}, \mathrm{public}, \widehat{n}))$, where $\omega \cong_\psi \widehat{\omega}$, $ty \cong \widehat{ty}$, $n = \widehat{n}$, and $perm = perm$.*

Proof. This holds by Definition 5.2.16 and the definition of function Erase. □

Lemma 5.2.64. *Given ψ, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l \cong_\psi \widehat{l}$ such that $l \in \sigma$ and $\widehat{l} \in \widehat{\sigma}$, if $\sigma(l) = (\omega, \mathrm{private}\ bty*,$ $n, \mathrm{PermL}(perm, \mathrm{private}\ bty*, \mathrm{private}, n))$ then $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \mathrm{PermL}(perm, \widehat{bty}*, \mathrm{public}, 1))$, where $\omega \cong_\psi \widehat{\omega}$, $ty \cong \widehat{ty}$, and $perm = perm$.*

Proof. This holds by Definition 5.2.16 and the definition of function Erase. □

Lemma 5.2.65 ($(l) \nvdash \sigma \implies l = \widehat{l}$)**.** *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, memory block identifier $l, \widehat{l}$, if $(l) \nvdash \sigma$, $(\gamma, \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, and $l \cong_\psi \widehat{l}$, then $l = \widehat{l}$.*

Proof. Using case analysis over the semantics, we can see that public memory blocks are never swapped around (the only rule that triggers locations being swapped is Multiparty Free, which only ever operates over private memory blocks). □

Lemma 5.2.66. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, type $ty \in \{a\ bty, a$ const $bty*$, public $bty*\}$, $\widehat{ty}$, byte representation $\omega, \widehat{\omega}$, number $n, \widehat{n}$, and permission $perm, \widehat{perm}$, if $\sigma_1 = \sigma_2[l \rightarrow (\omega, ty, n, \mathrm{PermL}(perm, ty, a, n))]$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, and $l \cong_\psi \widehat{l}$, then $\widehat{\sigma}_1 = \widehat{\sigma}_2[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{ty}, \widehat{n}, \mathrm{PermL}(perm, \widehat{ty},$ $\mathrm{public}, \widehat{n}))]$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$, $\omega \cong_\psi \widehat{\omega}$, $n = \widehat{n}$, $ty \cong \widehat{ty}$, and $perm = perm$.*

Proof. Using Definition 5.2.16 and the structure of memory, we can perform case analysis of the semantics to show that, for all types except void$*$ and private $bty*$, this holds. The interesting rules for this proof would be those modifying or adding to memory, showing that when they are first stored into memory this holds, and that there isn't anything that will break this property when memory is updated. □

Lemma 5.2.67. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, type $ty \in \{\mathrm{private}$ $bty*\}$, $\widehat{ty} \in \{\widehat{bty}*\}$, byte representation $\omega, \widehat{\omega}$, number $n, \widehat{n}$, and permission $perm, \widehat{perm}$, if $\sigma_1 = \sigma_2[l \rightarrow (\omega, ty, n,$ $\mathrm{PermL}(perm,\ ty, \mathrm{private}, n))]$, $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $l \cong_\psi \widehat{l}$, then $\widehat{\sigma}_1 = \widehat{\sigma}_2[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{ty}, 1, \mathrm{PermL}(perm,\ \widehat{ty},$ $\mathrm{public}, 1))]$ such that $(\gamma, \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, $\omega \cong_\psi \widehat{\omega}$, $ty \cong \widehat{ty}$, and $perm = perm$.*

Proof. Using Definition 5.2.16 and the structure of memory, we can perform case analysis of the semantics to show that this holds for all private pointers. The interesting rules for this proof would be those modifying or adding to memory of private pointers, showing that when they are first stored into memory this holds, and that there isn't anything that will break this property when memory is updated. □

Lemma 5.2.68. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, memory block identifier $l, \widehat{l}$, and size $n, \widehat{n}$, if $\sigma_1 = \sigma\big[l \rightarrow (\omega,\ \text{void}*,\ n,\ \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, n))\big]$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l = \widehat{l}$, then $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \text{void}*, \widehat{n}, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, \widehat{n}))]$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n = \widehat{n}$.*

Proof. Using Definition 5.2.16 and the structure of memory, we can perform case analysis of the semantics to show that this holds for all uncast public memory locations. The interesting rules for this proof would be those operating over uncast public memory (i.e., malloc and cast public location), showing that when they are first stored into memory this holds, and that there isn't anything that will break this property. □

Lemma 5.2.69. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma, \widehat{\sigma}$, memory block identifier $l, \widehat{l}$, type $ty, \widehat{ty}$, and size $n, \widehat{n}$, if $\sigma_1 = \sigma\big[l \rightarrow \big(\omega,\ \text{void}*,\ n,\ \text{PermL}(\text{Freeable}, \text{void}*, \text{private}, n)\big]$, $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, and $l \cong_\psi \widehat{l}$, then $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \text{void}*, \widehat{n}, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, \widehat{n}))]$ such that $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, and $\exists ty, \widehat{ty}$ such that $ty \cong \widehat{ty}$ and $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$.*

Proof. Using Definition 5.2.16 and the structure of memory, we can perform case analysis of the semantics to show that this holds for all uncast private memory locations. The interesting rules for this proof would be those operating over uncast private memory (i.e., pmalloc and cast private location), showing that when they are first stored into memory this holds, and that there isn't anything that will break this property. □

Lemma 5.2.70. *Given map ψ and type $ty \in \{\text{public } bty, \text{public } bty*\}$, $\widehat{ty}$, if $ty \cong_\psi \widehat{ty}$ then $\tau(ty) = \tau(\widehat{ty})$.*

Proof. By definition of τ. □

Lemma 5.2.71. *Given map ψ, variable name $x, \widehat{x}$ and input party number $n, \widehat{n}$ such that the corresponding input files $inp_n, \hat{inp}_\widehat{n}$ are congruent, if $\text{InputValue}(x, n) = n_1$, $x = \widehat{x}$, and $n = \widehat{n}$, then $\text{InputValue}(\widehat{x},\ \widehat{n}) = \widehat{n}_1$ such that $n_1 \cong_\psi \widehat{n}_1$.*

Proof. By definition of algorithm InputValue and by Definition 5.2.27. □

Lemma 5.2.72. *Given map ψ, variable name $x, \widehat{x}$, input party number $n, \widehat{n}$ such that the corresponding input files $inp_n, \hat{inp}_\widehat{n}$ are congruent, and array length $n_1, \widehat{n}_1$, if $\text{InputArray}(x, n, n_1) = [m_0, ..., m_{n_1}]$, $x = \widehat{x}$, $n = \widehat{n}$, and $n_1 = \widehat{n}_1$, then $\text{InputArray}(\widehat{x}, \widehat{n}, \widehat{n}_1) = [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$ such that $[m_0, ..., m_{n_1}] \cong_\psi [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$.*

Proof. By definition of algorithm InputArray and by Definition 5.2.27. □

Lemma 5.2.73. *Given map ψ, variable name $x, \widehat{x}$ and input party number $n, \widehat{n}$ such that the corresponding input files $out_n, \hat{out}_{\widehat{n}}$ are congruent, if $\text{OutputValue}(x, n, n_1)$, $x = \widehat{x}$, $n = \widehat{n}$, and $n_1 \cong_\psi \widehat{n}_1$, then $\text{OutputValue}(\widehat{x}, \widehat{n}, \widehat{n}_1)$ such that $out_n \cong \hat{out}_{\widehat{n}}$.*

Proof. By definition of algorithm OutputArray and by Definition 5.2.28. □

Lemma 5.2.74. *Given map ψ, variable name $x, \widehat{x}$, input party number $n, \widehat{n}$ such that the corresponding input files out_n, $\hat{out}_{\widehat{n}}$ are congruent, and array $[m_0, ..., m_{n_1}]$, $[\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, if* $\mathrm{OutputArray}(x, n, [m_0, ..., m_{n_1}])$*, $x = \widehat{x}$, $n = \widehat{n}$, and $[m_0, ..., m_{n_1}] \cong_\psi [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}]$, then* $\mathrm{OutputArray}(\widehat{x}, \widehat{n}, [\widehat{m}_0, ..., \widehat{m}_{\widehat{n}_1}])$ *such that $out_n \cong \hat{out}_{\widehat{n}}$.*

Proof. By definition of algorithm $\mathrm{OutputArray}$ and by Definition 5.2.28. □

Lemma 5.2.75. *Given map ψ, pointer variable being read or dereferenced $x, \widehat{x}$, and pointer data structure $[\alpha,\ \bar{l},\ \bar{j},\ i]$, $[1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, if x refers to $[\alpha,\ \bar{l},\ \bar{j},\ i]$, $\widehat{x}$ refers to $[1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$, and $x = \widehat{x}$, then $[\alpha,\ \bar{l},\ \bar{j},\ i] \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$ as a resulting value.*

Proof. By case analysis over the semantics, we can see that for every SMC^2 rule that returns multiple locations or accepts multiple locations as a result from an evaluation, there is a congruent Vanilla C rule that has corresponding behavior over a single location, leading to the formation of congruent trees. □

Lemma 5.2.76. *Given map ψ and configuration $((1,\ \gamma^1,\ \sigma^1,\ \Delta^1,\ \mathrm{acc},\ s) \parallel ... \parallel\ (\mathrm{q},\ \gamma^\mathrm{q},\ \sigma^\mathrm{q},\ \Delta^\mathrm{q},\ \mathrm{acc},\ s))$, environment $\widehat{\gamma}$, memory $\widehat{\sigma}$, and statement $\widehat{s}$, if $\{(\gamma^\mathrm{p}, \sigma^\mathrm{p}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $s \cong_\psi \widehat{s}$, then $((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}))$ such that $((1,\ \gamma^1,\ \sigma^1,\ \Delta^1,\ \mathrm{acc},\ s) \parallel ... \parallel\ (\mathrm{q},\ \gamma^\mathrm{q},\ \sigma^\mathrm{q},\ \Delta^\mathrm{q},\ \mathrm{acc},\ s)) \cong_\psi ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}))$.*

Proof. By Definitions 3.2.18 and 3.2.20. □

Lemma 5.2.77. *Given map ψ, environment $\{\gamma^\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\gamma}$, memory $\{\sigma^\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\sigma}$, variable $x, \widehat{x}$ such that $\{x \in \gamma^\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\widehat{x} \in \widehat{\gamma}$, if $\{\gamma^\mathrm{p}(x) = (l^\mathrm{p}, ty)\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{x} = x$, and $\{(\gamma^\mathrm{p}, \sigma^\mathrm{p}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})\}_{\mathrm{p}=1}^{\mathrm{q}}$, then $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{ty})$, where $\{l^\mathrm{p} = \widehat{l}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{(l^\mathrm{p}, 0) \cong_\psi (\widehat{l}, 0)\}_{\mathrm{p}=1}^{\mathrm{q}}$, and $ty \cong \widehat{ty}$.*

Proof. This holds by Definition 3.2.15 and the definition of function Erase. □

Lemma 5.2.78. *Given map ψ, environment $\{\gamma^\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\gamma}$, memory $\{\sigma^\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\sigma}$, and memory block identifier $\{l^\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{l}$ such that $\{l^\mathrm{p} \in \sigma^\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\widehat{l} \in \widehat{\sigma}$, if $\{\sigma^\mathrm{p}(l^\mathrm{p}) = (\omega^\mathrm{p},\ ty,\ n,\ \mathrm{PermL}(perm, ty, a, n))$ such that $ty \neq \mathrm{private}\ bty*$, $\{(\gamma^\mathrm{p}, \sigma^\mathrm{p}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{l^\mathrm{p} = \widehat{l}\}_{\mathrm{p}=1}^{\mathrm{q}}$, then $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{ty}, \widehat{n}, \mathrm{PermL}(perm, \widehat{ty}, \mathrm{public}, \widehat{n}))$, where $\{\omega^\mathrm{p} \cong_\psi \widehat{\omega}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $ty \cong \widehat{ty}$, $n = \widehat{n}$, and $perm = perm$.*

Proof. This holds by Definition 3.2.15 and the definition of function Erase. □

Lemma 5.2.79. *Given map ψ, environment $\{\gamma^\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\gamma}$, memory $\{\sigma^\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{\sigma}$, and memory block identifier $\{l^\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\widehat{l}$ such that $\{l^\mathrm{p} \in \sigma^\mathrm{p}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\widehat{l} \in \widehat{\sigma}$, if $\{\sigma^\mathrm{p}(l^\mathrm{p}) = (\omega^\mathrm{p},\ \mathrm{private}\ bty*,\ n,\ \mathrm{PermL}(perm, \mathrm{private}\ bty*, a, n))$, $\{(\gamma^\mathrm{p}, \sigma^\mathrm{p}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{l^\mathrm{p} = \widehat{l}\}_{\mathrm{p}=1}^{\mathrm{q}}$, then $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \mathrm{PermL}(perm, \widehat{bty}*, \mathrm{public}, 1))$, where $\{\omega^\mathrm{p} \cong_\psi \widehat{\omega}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\mathrm{private}\ bty* \cong \widehat{bty}*$, and $perm = perm$.*

Proof. This holds by Definition 3.2.15 and the definition of function Erase. □

Lemma 5.2.80. *Given map ψ, environment $\gamma, \widehat{\gamma}$, memory $\sigma_1, \widehat{\sigma}_1$, memory block identifier $l, \widehat{l}$, list of values $[n_0, ..., n_{\alpha-1}]$, $[\widehat{n}_0, ..., \widehat{n}_{\widehat{\alpha}-1}]$, and type a bty, $\widehat{bty}$, if* $\{\forall i \in \{0...\alpha-1\}$ $\mathrm{UpdateArr}(\sigma^{\mathrm{p}}_{1+i}, (l^{\mathrm{p}}_1, i), n^{\mathrm{p}}_i,$ private $bty) = \sigma^{\mathrm{p}}_{2+i}\}^{\mathrm{q}}_{\mathrm{p}=1}$ $(\gamma, \sigma_1) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_1)$, $\{l^{\mathrm{p}} = \widehat{l}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\alpha = \widehat{\alpha}$, $\{\omega^{\mathrm{p}} \cong_\psi \widehat{\omega}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{\forall i \in \{0...\alpha-1\}$ $\mathrm{DecodeArr}(\text{private } bty, i, \omega^{\mathrm{p}})$ $= n^{\mathrm{p}}_i\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{n'^{\mathrm{p}}_{\widehat{i}} \cong_{\psi_2} \widehat{n}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{\forall j \neq \widehat{i} \in \{0...\alpha-1\} n^{\mathrm{p}}_j = n'^{\mathrm{p}}_j\}^{\mathrm{q}}_{\mathrm{p}=1}$, *and a bty* $\cong_\psi \widehat{bty}$, *then* $\mathrm{UpdateArr}(\widehat{\sigma}_1, (\widehat{l}, \widehat{i}), \widehat{n}, \widehat{bty}) = \sigma_2$ *such that* $(\gamma, \sigma_{2+\alpha-1}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$.

Proof. Given *a bty* $\cong_\psi \widehat{bty}$, $\{\omega^{\mathrm{p}} \cong_\psi \widehat{\omega}\}^{\mathrm{q}}_{\mathrm{p}=1}$ and $\{\forall i \in \{0...\alpha-1\}$ $\mathrm{DecodeArr}(\text{private } bty, i, \omega^{\mathrm{p}}) = n^{\mathrm{p}}_i\}^{\mathrm{q}}_{\mathrm{p}=1}$, by Definition 3.2.16 and Lemma 5.2.9, we have that these SMC^2 values read from memory are ψ-congruent to the values stored in the array for Vanilla C.

Given updated list of values $\{[n'^{\mathrm{p}}_0, ..., n'^{\mathrm{p}}_{\alpha-1}]\}^{\mathrm{q}}_{\mathrm{p}=1}$ such that $\{n'^{\mathrm{p}}_{\widehat{i}} \cong_{\psi_2} \widehat{n}\}^{\mathrm{q}}_{\mathrm{p}=1}$ and $\{\forall j \neq \widehat{i} \in \{0...\alpha-1\} n^{\mathrm{p}}_j = n'^{\mathrm{p}}_j\}^{\mathrm{q}}_{\mathrm{p}=1}$, we have that only the value at index $\widehat{i}$ is modified in the updated list of values. Given this, we are only storing an updated value in memory once, all other values will simply be overwritten with the same value.

By Lemma 5.2.15, we have that the environment and memory pair maintains ψ-congruency when updating the value that changed within the array for a single party, which in turn holds for all parties.

Given the above, we have the ending environment and memory pairs ψ-congruent, or $(\gamma, \sigma_{2+\alpha-1}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma}_2)$. □

Lemma 5.2.81. *Given map ψ, type* private *bty, $\widehat{bty}$, pointer data structure* $\{[\alpha, \bar{l}^{\mathrm{p}}, \bar{j}^{\mathrm{p}}, 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, $[1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$, *environment* $\{\gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\gamma}$, *and memory* $\{\sigma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\sigma}$,
if $\{\mathrm{Retrieve_vals}(\alpha, \bar{l}^{\mathrm{p}}, \text{private } bty, \sigma^{\mathrm{p}}) = ([n^{\mathrm{p}}_0, ...n^{\mathrm{p}}_{\alpha-1}], 1)\}^{\mathrm{q}}_{\mathrm{p}=1}$ $\mathrm{MPC}_{dv}([[n^1_0, ..., n^1_{\alpha-1}], ..., [n^{\mathrm{q}}_0, ..., n^{\mathrm{q}}_{\alpha-1}]], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) = (n^1, ..., n^{\mathrm{q}})$, $\{[\alpha, \bar{l}^{\mathrm{p}}, \bar{j}^{\mathrm{p}}, 1] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, private *bty* $\cong_\psi \widehat{bty}$, *and* $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})\}^{\mathrm{q}}_{\mathrm{p}=1}$,
then $\mathrm{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{n}, 1)$ *such that* $\{n^{\mathrm{p}} \cong \widehat{n}\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. By definition of Retrieve_vals, we have $\{[n^{\mathrm{p}}_0, ...n^{\mathrm{p}}_{\alpha-1}]\}^{\mathrm{q}}_{\mathrm{p}=1}$ such that each value n^{p}_j is the value stored at location j in $\bar{l}^{\mathrm{p}}$. Therefore, by Axiom 5.2.11 we have that $\{n^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$ is the value stored in the true location referred to by the private pointer.

Given $\{[\alpha, \bar{l}^{\mathrm{p}}, \bar{j}^{\mathrm{p}}, 1] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, private *bty* $\cong_\psi \widehat{bty}$, and $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})\}^{\mathrm{q}}_{\mathrm{p}=1}$, we have the Vanilla C call $\mathrm{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{n}, 1)$ such that $\{n^{\mathrm{p}} \cong \widehat{n}\}^{\mathrm{q}}_{\mathrm{p}=1}$. □

Lemma 5.2.82. *Given map ψ, type* private *bty*, $\widehat{bty}*$, pointer data structure* $\{[\alpha, \bar{l}^{\mathrm{p}}, \bar{j}^{\mathrm{p}}, 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, $[1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$, *environment* $\{\gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\gamma}$, *and memory* $\{\sigma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\sigma}$,
if $\{\mathrm{Retrieve_vals}(\alpha, \bar{l}^{\mathrm{p}}, \text{private } bty*, \sigma^{\mathrm{p}}) = ([[\alpha_0, \bar{l}^{\mathrm{p}}_0, \bar{j}^{\mathrm{p}}_0, i-1], ..., [\alpha_{\alpha-1}, \bar{l}^{\mathrm{p}}_{\alpha-1}, \bar{j}^{\mathrm{p}}_{\alpha-1}, i-1]], 1)\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\mathrm{MPC}_{dp}([[[\alpha_0, \bar{l}^1_0, \bar{j}^1_0], ..., [\alpha_{\alpha-1}, \bar{l}^1_{\alpha-1}, \bar{j}^1_{\alpha-1}]], ..., [[\alpha_0, \bar{l}^{\mathrm{q}}_0, \bar{j}^{\mathrm{q}}_0], ..., [\alpha_{\alpha-1}, \bar{l}^{\mathrm{q}}_{\alpha-1}, \bar{j}^{\mathrm{q}}_{\alpha-1}]]], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) = ([[\alpha_\alpha, \bar{l}^1_\alpha, \bar{j}^1_\alpha], ..., [\alpha_\alpha, \bar{l}^{\mathrm{q}}_\alpha, \bar{j}^{\mathrm{q}}_\alpha]])$, $\{[\alpha, \bar{l}^{\mathrm{p}}, \bar{j}^{\mathrm{p}}, 1] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, private *bty** $\cong_\psi \widehat{bty}*$, *and* $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})\}^{\mathrm{q}}_{\mathrm{p}=1}$,
then $\mathrm{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty}*, (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1], 1)$ *such that* $\{[\alpha_\alpha, \bar{l}^{\mathrm{q}}_\alpha, \bar{j}^{\mathrm{q}}_\alpha, \widehat{i}-1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1]\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. By definition of Retrieve_vals, we have $\{\forall j \in \{0...\alpha-1\}[\alpha_j,\ \bar{l}^{\mathrm{p}}_j,\ \bar{j}^{\mathrm{p}}_j, i-1]\}^{\mathrm{q}}_{\mathrm{p}=1}$ such that each pointer data structure $[\alpha_j,\ \bar{l}^{\mathrm{p}}_j,\ \bar{j}^{\mathrm{p}}_j, i-1]$ is stored at location j in $\bar{l}^{\mathrm{p}}$. Therefore, by Axiom 5.2.12 we have that $\{[\alpha_\alpha,\ \bar{l}^1_\alpha,\ \bar{j}^1_\alpha]\}^{\mathrm{q}}_{\mathrm{p}=1}$ properly indicates the true location of the lower level private pointer that is the true location referred to by the higher level private pointer.

Given $\{[\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}}, 1] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, private $bty*$ $\cong_\psi$ $\widehat{bty}*$, and $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})\}^{\mathrm{q}}_{\mathrm{p}=1}$, we have the Vanilla C call $\mathrm{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty}*, (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1], 1)$ such that $\{[\alpha_\alpha,\ \bar{l}^{\mathrm{q}}_\alpha,\ \bar{j}^{\mathrm{q}}_\alpha, \widehat{i}-1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i}-1]\}^{\mathrm{q}}_{\mathrm{p}=1}$. □

Lemma 5.2.83. *Given map ψ, type* private *bty, $\widehat{bty}$, pointer data structure* $\{[\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}}, 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, $[1, [(\widehat{l}, \widehat{\mu})], [1], 1]$, *values* $\{n^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{n}$, *environment* $\{\gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\gamma}$, *and memory* $\{\sigma^{\mathrm{p}}_1\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\sigma}_1$,
if $\{\mathrm{Retrieve_vals}(\alpha, \bar{l}^{\mathrm{p}}, \text{private } bty, \sigma^{\mathrm{p}}_1) = ([n^{\mathrm{p}}_0, ...n^{\mathrm{p}}_{\alpha-1}], 1)\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\mathrm{MPC}_{wdv}([[n^1_0, ..., n^1_{\alpha-1}], ..., [n^{\mathrm{q}}_0, ..., n^{\mathrm{q}}_{\alpha-1}]], [n^1, ..., n^{\mathrm{q}}], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) = ([n'^1_0, ..., n'^1_{\alpha-1}], ..., [n'^{\mathrm{q}}_0, ..., n'^{\mathrm{q}}_{\alpha-1}])$, *and* $\{\mathrm{UpdateDerefVals}(\alpha, \bar{l}^{\mathrm{p}}, [n'^{\mathrm{p}}_0, ..., n'^{\mathrm{p}}_{\alpha-1}], \text{private } bty, \sigma^{\mathrm{p}}_1) = \sigma^{\mathrm{p}}_2\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{[\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}}, 1] \cong_\psi [1, [(\widehat{l}, \widehat{\mu})], [1], 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{n^{\mathrm{p}} \cong_\psi \widehat{n}\}^{\mathrm{q}}_{\mathrm{p}=1}$, private bty $\cong_\psi$ $\widehat{bty}$, *and* $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)\}^{\mathrm{q}}_{\mathrm{p}=1}$,
then $\mathrm{UpdateOffset}(\widehat{\sigma}_1, (\widehat{l}, \widehat{\mu}), \widehat{n}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$ *such that* $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}_2) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_2)\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. By definition of Retrieve_vals, we have $\{[n^{\mathrm{p}}_0, ...n^{\mathrm{p}}_{\alpha-1}]\}^{\mathrm{q}}_{\mathrm{p}=1}$ such that each value n^{p}_j is the value stored at location j in $\bar{l}^{\mathrm{p}}$. Therefore, by Axiom 5.2.13 we have that $\{n'^{\mathrm{p}}_j = n^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$ and $\{\forall i \neq j \in \{0...\alpha-1\}\ n'^{\mathrm{p}}_i = n^{\mathrm{p}}_i\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Given $\{[\alpha, \bar{l}^{\mathrm{p}}, \bar{j}^{\mathrm{p}}, 1] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, private bty $\cong_\psi$ $\widehat{bty}$, and $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})\}^{\mathrm{q}}_{\mathrm{p}=1}$, we have the Vanilla C call $\mathrm{UpdateOffset}(\widehat{\sigma}_1, (\widehat{l}, \widehat{\mu}), \widehat{n}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$ such that $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}_2) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_2)\}^{\mathrm{q}}_{\mathrm{p}=1}$. □

Lemma 5.2.84. *Given map ψ, type* private *$bty*$, $\widehat{bty}*$, pointer data structure* $\{[\alpha, \bar{l}^{\mathrm{p}}, \bar{j}^{\mathrm{p}}, 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, $[1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$, *location* $\{(l^{\mathrm{p}}_e, \mu^{\mathrm{p}}_e)\}^{\mathrm{q}}_{\mathrm{p}=1}$, $(\widehat{l}_e, \widehat{\mu}_e)$ *environment* $\{\gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\gamma}$, *and memory* $\{\sigma^{\mathrm{p}}_1\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\sigma}_1$,
if $\{\mathrm{Retrieve_vals}(\alpha,\ \bar{l}^{\mathrm{p}},\ \text{private } bty*,\ \sigma^{\mathrm{p}}_1) = ([[\alpha_0,\ \bar{l}^{\mathrm{p}}_0,\ \bar{j}^{\mathrm{p}}_0, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}^{\mathrm{p}}_{\alpha-1}, \bar{j}^{\mathrm{p}}_{\alpha-1}, i-1]], 1)\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\mathrm{MPC}_{wdp}([[[1, [(l^1_e, \mu^1_e)], [1], i-1],\ [\alpha_0,\ \bar{l}^1_0,\ \bar{j}^1_0, i-1], ...,\ [\alpha_{\alpha-1},\ \bar{l}^1_{\alpha-1},\ \bar{j}^1_{\alpha-1}, i-1]], ...,\ [[1, [(l^{\mathrm{q}}_e, \mu^{\mathrm{q}}_e)], [1], i-1],\ [\alpha_0,\ \bar{l}^{\mathrm{q}}_0,\ \bar{j}^{\mathrm{q}}_0, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}^{\mathrm{q}}_{\alpha-1},\ \bar{j}^{\mathrm{q}}_{\alpha-1}, i-1]]], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) = [[[\alpha'_0,\ \bar{l}'^1_0,\ \bar{j}'^1_0, i-1], ..., [\alpha'_{\alpha-1},\ \bar{l}'^1_{\alpha-1},\ \bar{j}'^1_{\alpha-1}, i-1]], ..., [[\alpha'_0,\ \bar{l}'^{\mathrm{q}}_0,\ \bar{j}'^{\mathrm{q}}_0, i-1], ..., [\alpha'_{\alpha-1},\ \bar{l}'^{\mathrm{q}}_{\alpha-1},\ \bar{j}'^{\mathrm{q}}_{\alpha-1}, i-1]]]$, $\{\mathrm{UpdateDerefVals}(\alpha, \bar{l}^{\mathrm{p}}, [[\alpha'_0, \bar{l}'^{\mathrm{p}}_0, \bar{j}'^{\mathrm{p}}_0, i-1], ..., [\alpha'_{\alpha-1}, \bar{l}'^{\mathrm{p}}_{\alpha-1}, \bar{j}'^{\mathrm{p}}_{\alpha-1}, i-1]], \text{private } bty*, \sigma^{\mathrm{p}}_1) = \sigma^{\mathrm{p}}_2\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{[\alpha, \bar{l}^{\mathrm{p}}, \bar{j}^{\mathrm{p}}, 1] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{(l^{\mathrm{p}}_e, \mu^{\mathrm{p}}_e) \cong_{\psi_1} (\widehat{l}_e, \widehat{\mu}_e)\}^{\mathrm{q}}_{\mathrm{p}=1}$, private $bty*$ $\cong_\psi$ $\widehat{bty}*$, *and* $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)\}^{\mathrm{q}}_{\mathrm{p}=1}$,
then $\mathrm{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$ *such that* $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. By definition of Retrieve_vals, we have $\{\forall j \in \{0...\alpha-1\}[\alpha_j,\ \bar{l}^{\mathrm{p}}_j,\ \bar{j}^{\mathrm{p}}_j, i-1]\}^{\mathrm{q}}_{\mathrm{p}=1}$ such that each pointer data structure $[\alpha_j,\ \bar{l}^{\mathrm{p}}_j,\ \bar{j}^{\mathrm{p}}_j, i-1]$ is stored at location j in $\bar{l}^{\mathrm{p}}$. Therefore, by Axiom 5.2.14 we have that $[\alpha_j,\ \bar{l}^{\mathrm{p}}_j,\ \bar{j}^{\mathrm{p}}_j]$ has the true location set as $(l^{\mathrm{p}}_e, \mu^{\mathrm{p}}_e)$ and $\forall i \neq j \in \{0...\alpha-1\}[\alpha_i,\ \bar{l}^{\mathrm{p}}_i,\ \bar{j}^{\mathrm{p}}_i]$, the true location remains the same as what it originally was.

Given $\{[\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}}, 1]\ \cong_{\psi}\ [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, private $bty* \cong_{\psi} \widehat{bty*}$, $\{(l^{\mathrm{p}}_e, \mu^{\mathrm{p}}_e)\ \cong_{\psi_1}\ (\widehat{l}_e, \widehat{\mu}_e)\}^{\mathrm{q}}_{\mathrm{p}=1}$ and $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})\}^{\mathrm{q}}_{\mathrm{p}=1}$, by definition of UpdatePtr, we have the Vanilla C call $\mathrm{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1], \widehat{bty*}) = (\widehat{\sigma}_2, 1)$ such that $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}_2) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma}_2)\}^{\mathrm{q}}_{\mathrm{p}=1}$. □

Lemma 5.2.85. *Given map ψ, pointer data structure $\{[\alpha, \bar{l}^{\mathrm{p}}, \bar{j}^{\mathrm{p}}, 1]\}^{\mathrm{q}}_{\mathrm{p}=1}$, $[1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$, environment $\{\gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\gamma}$, and memory $\{\sigma^{\mathrm{p}}_1\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\widehat{\sigma}_1$,*
if $\{\forall(l^{\mathrm{p}}_m, 0) \in \bar{l}^{\mathrm{p}}.\ \sigma^{\mathrm{p}}(l^{\mathrm{p}}_m) = (\omega^{\mathrm{p}}_m, ty, n, \mathrm{PermL}(\mathrm{Freeable}, ty, \mathrm{private}, n))\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\mathrm{MPC}_{free}([[\omega^1_0, ..., \omega^1_{\alpha-1}], ..., [\omega^{\mathrm{q}}_0, ..., \omega^{\mathrm{q}}_{\alpha-1}]], [\bar{j}^1, ...\bar{j}^{\mathrm{q}}]) = ([[\omega'^1_0, ..., \omega'^1_{\alpha-1}], ..., [\omega'^{\mathrm{q}}_0, ..., \omega'^{\mathrm{q}}_{\alpha-1}]], [\bar{j}'^1, ..., \bar{j}'^{\mathrm{q}}])$, $\{\mathrm{UpdateBytesFree}(\sigma^{\mathrm{p}}, \bar{l}^{\mathrm{p}}, [\omega'^{\mathrm{p}}_0, ..., \omega'^{\mathrm{p}}_{\alpha-1}]) = \sigma^{\mathrm{p}}_1\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{\sigma^{\mathrm{p}}_2 = \mathrm{UpdatePointerLocations}(\sigma^{\mathrm{p}}_1, \bar{l}^{\mathrm{p}}[1 : \alpha - 1], \bar{j}^{\mathrm{p}}[1 : \alpha - 1], \bar{l}^{\mathrm{p}}[0], \bar{j}^{\mathrm{p}}[0])\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\{[\alpha,\ \bar{l}^{\mathrm{p}}\ \bar{j}^{\mathrm{p}},\ i] \cong_{\psi} [1, [(\widehat{l}_1, 0)], [1], \widehat{i}]\}^{\mathrm{q}}_{\mathrm{p}=1}$, and $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})\}^{\mathrm{q}}_{\mathrm{p}=1}$,
then $\mathrm{Free}(\widehat{\sigma}, \widehat{l}_1) = \widehat{\sigma}_1$ and ψ_1 such that $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}_2) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Proof. Proof Sketch:
Given $\{\forall(l^{\mathrm{p}}_m, 0) \in \bar{l}.\ \sigma^{\mathrm{p}}(l^{\mathrm{p}}_m) = (\omega^{\mathrm{p}}_m, ty, n, \mathrm{PermL}(\mathrm{Freeable}, ty, \mathrm{private}, n))\}^{\mathrm{q}}_{\mathrm{p}=1}$, we have pulled all the byte representations for each location within $\bar{l}^{\mathrm{p}}$.

Given $\mathrm{MPC}_{free}([[\omega^1_0, ..., \omega^1_{\alpha-1}], ..., [\omega^{\mathrm{q}}_0, ..., \omega^{\mathrm{q}}_{\alpha-1}]], [\bar{j}^1, ...\bar{j}^{\mathrm{q}}]) = ([[\omega'^1_0, ..., \omega'^1_{\alpha-1}], ..., [\omega'^{\mathrm{q}}_0, ..., \omega'^{\mathrm{q}}_{\alpha-1}]], [\bar{j}'^1, ..., \bar{j}'^{\mathrm{q}}])$, we have that either tag 0 was not the true location and therefore the byte representation for a location j was swapped with the byte representation for 0 and all others remain the same, or 0 was the true location and all byte representations remain constant.

If the locations were swapped, we obtain ψ_1 by add a mapping to ψ indicating that location 0 was swapped with location j. If the locations were not swapped, $\psi_1 = \psi$.

Given $\{\mathrm{UpdateBytesFree}(\sigma^{\mathrm{p}}, \bar{l}^{\mathrm{p}}, [\omega'^{\mathrm{p}}_0, ..., \omega'^{\mathrm{p}}_{\alpha-1}]) = \sigma^{\mathrm{p}}_1\}^{\mathrm{q}}_{\mathrm{p}=1}$, by definition of UpdateBytesFree, we have that each of the updated byte representations are placed into memory at their corresponding locations, with the permissions at the first location marked as Freeable.

Given $\{\sigma^{\mathrm{p}}_2 = \mathrm{UpdatePointerLocations}(\sigma^{\mathrm{p}}_1, \bar{l}^{\mathrm{p}}[1 : \alpha - 1], \bar{j}^{\mathrm{p}}[1 : \alpha - 1], \bar{l}^{\mathrm{p}}[0], \bar{j}^{\mathrm{p}}[0])\}^{\mathrm{q}}_{\mathrm{p}=1}$, by definition of UpdatePointerLocations we will iterate through and find all private pointers. If the private pointer had location 0 in its location list, we will appropriately update the location list to store the union of what it was and what the location list of the pointer we just freed was, merging the lists and updating the tags so that, if the location we freed was it's true location and we swapped the byte data to a new location, the pointer will now refer it's true location to the location j that we swapped the data to.

Once we have ensured all pointers that could have been affected by the swapping of locations are properly updated, we obtain $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}_2) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)\}^{\mathrm{q}}_{\mathrm{p}=1}$. □

5.2.3 Multiparty Computation Axioms

Axiom 5.2.4 (MPC_b)**.** *Given bop* $\in \{+, -, \cdot, \div\}$*, values* $\{n_1^{\mathrm{p}}, n_2^{\mathrm{p}}, \widehat{n}_1, \widehat{n}_2\}_{\mathrm{p}=1}^{\mathrm{q}} \in \mathbb{N}$*,*
if $\mathrm{MPC}_b(bop, [n_1^1, ..., n_1^{\mathrm{q}}], [n_2^1, ..., n_2^{\mathrm{q}}]) = (n_3^1, ..., n_3^{\mathrm{q}})$*,* $\{n_1^{\mathrm{p}} \cong \widehat{n}_1\}_{\mathrm{p}=1}^{\mathrm{q}}$*, and* $\{n_2^{\mathrm{p}} \cong \widehat{n}_2\}_{\mathrm{p}=1}^{\mathrm{q}}$*,*
then $\{n_3^{\mathrm{p}} \cong \widehat{n}_3^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *such that* $\widehat{n}_1$ *bop* $\widehat{n}_2 = \widehat{n}_3$*.*

Axiom 5.2.5 (MPC_{cmp})**.** *Given bop* $\in \{==, !=, <\}$*, values* $\{n_1^{\mathrm{p}}, n_2^{\mathrm{p}}, \widehat{n}_1, \widehat{n}_2\}_{\mathrm{p}=1}^{\mathrm{q}} \in \mathbb{N}$*,*
if $\mathrm{MPC}_{cmp}(bop, [n_1^1, ..., n_1^{\mathrm{q}}], [n_2^1, ..., n_2^{\mathrm{q}}]) = (n_3^1, ..., n_3^{\mathrm{q}})$*,* $\{n_1^{\mathrm{p}} \cong \widehat{n}_1\}_{\mathrm{p}=1}^{\mathrm{q}}$*, and* $\{n_2^{\mathrm{p}} \cong \widehat{n}_2\}_{\mathrm{p}=1}^{\mathrm{q}}$*,*
then $\{n_3^{\mathrm{p}} \cong \widehat{n}_3\}_{\mathrm{p}=1}^{\mathrm{q}}$ *such that* $\widehat{n}_1$ *bop* $\widehat{n}_2 = \widehat{n}_3$*.*

Axiom 5.2.6 ($\mathrm{MPC}_{resolve}$ False Conditional)**.** *Given conditional result values* $\{n^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and branch result values* $\{[(v_{t1}^{\mathrm{p}}, v_{e1}^{\mathrm{p}}), ..., (v_{tm}^{\mathrm{p}}, v_{em}^{\mathrm{p}})]\}_{\mathrm{p}=1}^{\mathrm{q}}$*,*
if $\mathrm{MPC}_{resolve}([n^1, ..., n^{\mathrm{q}}],\ [[(v_{t1}^1, v_{e1}^1), ..., (v_{tm}^1, v_{em}^1)],\ ...,\ [(v_{t1}^{\mathrm{q}},\ v_{e1}^{\mathrm{q}}),\ ...,\ (v_{tm}^{\mathrm{q}}, v_{em}^{\mathrm{q}})]]) = [[v_1^1, ..., v_m^1], ..., [v_1^{\mathrm{q}},\ ..., v_m^{\mathrm{q}}]]$*, and* $\{n^{\mathrm{p}} \cong \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *such that* $\widehat{n} = 0$*, then* $\{\forall i \in \{1...m\}, v_i^{\mathrm{p}} = v_{ei}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*.*

Axiom 5.2.7 ($\mathrm{MPC}_{resolve}$ True Conditional)**.** *Given conditional result values* $\{n^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and branch result values* $\{[(v_{t1}^{\mathrm{p}}, v_{e1}^{\mathrm{p}}), ..., (v_{tm}^{\mathrm{p}}, v_{em}^{\mathrm{p}})]\}_{\mathrm{p}=1}^{\mathrm{q}}$*,*
if $\mathrm{MPC}_{resolve}([n^1, ..., n^{\mathrm{q}}],\ [[(v_{t1}^1, v_{e1}^1), ..., (v_{tm}^1, v_{em}^1)],\ ...,\ [(v_{t1}^{\mathrm{q}},\ v_{e1}^{\mathrm{q}}),\ ...,\ (v_{tm}^{\mathrm{q}}, v_{em}^{\mathrm{q}})]]) = [[v_1^1, ..., v_m^1], ..., [v_1^{\mathrm{q}},\ ..., v_m^{\mathrm{q}}]]$*, and* $\{n^{\mathrm{p}} \cong \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *such that* $\widehat{n} \neq 0$*,*
then $\{\forall i \in \{1...m\}, v_i^{\mathrm{p}} = v_{ti}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*.*

Axiom 5.2.8 (MPC_{ar})**.** *Given array size* $\alpha, \widehat{\alpha}$*, values* $\{[n_0^{\mathrm{p}}, ..., n_{\alpha-1}^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$*,* $\widehat{n}_{\widehat{i}}$*, and indices* $\{i^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*,* $\widehat{i}$*,*
if $\mathrm{MPC}_{ar}((i^1, [n_0^1, ..., n_{\alpha-1}^1]), ..., (i^{\mathrm{q}}, [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}])) = (n^1, ..., n^{\mathrm{q}})$*,* $0 \leq \widehat{i} < \widehat{\alpha}$*,* $\alpha = \widehat{\alpha}$*,* $\{i^{\mathrm{p}} \cong \widehat{i}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, and* $\{n_{\widehat{i}}^{\mathrm{p}} \cong \widehat{n}_{\widehat{i}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, then* $\{n^{\mathrm{p}} \cong \widehat{n}_{\widehat{i}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*.*

Axiom 5.2.9 (MPC_{aw})**.** *Given array size* $\alpha, \widehat{\alpha}$*, values* $\{[n_0^{\mathrm{p}}, ..., n_{\alpha-1}^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$*,* $\widehat{n}_{\widehat{i}}$*, and indices* $\{i^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*,* $\widehat{i}$*,*
if $\mathrm{MPC}_{aw}((i^1, n^1, [n_0^1, ..., n_{\alpha-1}^1]), ...,\ (i^{\mathrm{q}}, n^{\mathrm{q}}, [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}])) = ([n_0'^1, ..., n_{\alpha-1}'^1], ..., [n_0'^{\mathrm{q}}, ..., n_{\alpha-1}'^{\mathrm{q}}])$*,* $0 \leq \widehat{i} < \widehat{\alpha}$*,* $\alpha = \widehat{\alpha}$*,* $\{i^{\mathrm{p}} \cong \widehat{i}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, and* $\{n_{\widehat{i}}^{\mathrm{p}} \cong \widehat{n}_{\widehat{i}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*,*
then $\{n_{\widehat{i}}'^{\mathrm{p}} \cong \widehat{n}_{\widehat{i}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{\forall j \neq \widehat{i} \in \{0...\alpha-1\} n_j^{\mathrm{p}} = n_j'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *.*

Axiom 5.2.10 (MPC_u)**.** *Given array size* $\alpha, \widehat{\alpha}$*, unary operator* $uop \in \{++\}$*, and values* $\{n_1^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*,* $\widehat{n}_1$*,*
if $\mathrm{MPC}_u(++, n_1^1, ..., n_1^{\mathrm{q}}) = (n_2^1, ..., n_2^{\mathrm{q}})$ *and* $\{n_1^{\mathrm{p}} \cong \widehat{n}_1\}_{\mathrm{p}=1}^{\mathrm{q}}$*,*
then $\{n_2^{\mathrm{p}} \cong \widehat{n}_2\}_{\mathrm{p}=1}^{\mathrm{q}}$ *such that* $\widehat{n}_2 = \widehat{n}_1 + 1$*.*

Axiom 5.2.11 (MPC_{dv})**.** *Given map* ψ*, tag lists* $\{\overline{j}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, and values stored at each location referred to by the given private pointer* $\{[n_0^{\mathrm{p}}, ..., n_{\alpha-1}^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$*,*
if $\mathrm{MPC}_{dv}([[n_0^1, ..., n_{\alpha-1}^1], ..., [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}]], [\overline{j}^1, ..., \overline{j}^{\mathrm{q}}]) = (n^1, ..., n^{\mathrm{q}})$*,*
then $\{n^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *is the value stored in the true location referred to by the private pointer.*

Axiom 5.2.12 (MPC_{dp})**.** *Given map ψ, number of location α, tag lists $\{\bar{j}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and pointer data structures stored at each of the α location referred to by the given higher level private pointer $\{[\alpha_j,\ \bar{l}_j^{\mathrm{p}},\ \bar{j}_j^{\mathrm{p}}, i-1]\}_{\mathrm{p}=1}^{\mathrm{q}}$,*
if $\mathrm{MPC}_{dp}([[[\alpha_0,\ \bar{l}_0^1,\ \bar{j}_0^1], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^1, \bar{j}_{\alpha-1}^1]], ..., [[\alpha_0,\ \bar{l}_0^{\mathrm{q}},\ \bar{j}_0^{\mathrm{q}}], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^{\mathrm{q}},\ \bar{j}_{\alpha-1}^{\mathrm{q}}]]], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) = ([[\alpha_\alpha,\ \bar{l}_\alpha^1,\ \bar{j}_\alpha^1], ..., [\alpha_\alpha,\ \bar{l}_\alpha^{\mathrm{q}},\ \bar{j}_\alpha^{\mathrm{q}}]])$,
then $\{[\alpha_\alpha,\ \bar{l}_\alpha^{\mathrm{p}},\ \bar{j}_\alpha^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$ properly indicates the true location of the lower level private pointer that is the true location referred to by the higher level private pointer.

Axiom 5.2.13 (MPC_{wdv})**.** *Given map ψ, tag lists $\{\bar{j}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and values stored at each location referred to by the given private pointer $\{[n_0^{\mathrm{p}}, ..., n_{\alpha-1}^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$,*
if $\mathrm{MPC}_{wdv}([[n_0^1, ..., n_{\alpha-1}^1], ..., [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}]],\ [n^1, ..., n^{\mathrm{q}}], [\bar{j}^1,\ ...,\ \bar{j}^{\mathrm{q}}]) = ([n_0'^1, ..., n_{\alpha-1}'^1], ..., [n_0'^{\mathrm{q}}, ..., n_{\alpha-1}'^{\mathrm{q}}])$ *and* $\{\bar{j}^{\mathrm{p}}[j] = \mathrm{encrypt}(1)\}_{\mathrm{p}=1}^{\mathrm{q}}$, *then $\{n_j'^{\mathrm{p}} = n^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\{\forall i \neq j \in \{0...\alpha-1\}\ n_i'^{\mathrm{p}} = n_i^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.*

Axiom 5.2.14 (MPC_{wdp})**.** *Given map ψ, number of location α, tag lists $\{\bar{j}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and pointer data structures stored at each of the α location referred to by the given higher level private pointer $\{[\alpha_j,\ \bar{l}_j^{\mathrm{p}},\ \bar{j}_j^{\mathrm{p}}, i-1]\}_{\mathrm{p}=1}^{\mathrm{q}}$,*
if $\mathrm{MPC}_{wdp}([[1, [(l_e^1, \mu_e^1)], [1], i-1],\ [\alpha_0,\ \bar{l}_0^1,\ \bar{j}_0^1, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^1,\ \bar{j}_{\alpha-1}^1, i-1]], ..., [[1, [(l_e^{\mathrm{q}}, \mu_e^{\mathrm{q}})], [1], i-1],\ [\alpha_0,\ \bar{l}_0^{\mathrm{q}},\ \bar{j}_0^{\mathrm{q}}, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^{\mathrm{q}},\ \bar{j}_{\alpha-1}^{\mathrm{q}}, i-1]], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) = [[[\alpha_0',\ \bar{l}_0'^1,\ \bar{j}_0'^1, i-1], ..., [\alpha_{\alpha-1}',\ \bar{l}_{\alpha-1}'^1,\ \bar{j}_{\alpha-1}'^1, i-1]], ..., [\alpha_0',\ \bar{l}_0'^{\mathrm{q}},\ \bar{j}_0'^{\mathrm{q}}, i-1], ..., [\alpha_{\alpha-1}',\ \bar{l}_{\alpha-1}'^{\mathrm{q}},\ \bar{j}_{\alpha-1}'^{\mathrm{q}}, i-1]]$ *and* $\{\bar{j}^{\mathrm{p}}[j] = \mathrm{encrypt}(1)\}_{\mathrm{p}=1}^{\mathrm{q}}$,
then $[\alpha_j,\ \bar{l}_j^{\mathrm{p}},\ \bar{j}_j^{\mathrm{p}}]$ has the true location set as $(l_e^{\mathrm{p}}, \mu_e^{\mathrm{p}})$ and $\forall i \neq j \in \{0...\alpha-1\}[\alpha_i,\ \bar{l}_i^{\mathrm{p}},\ \bar{j}_i^{\mathrm{p}}]$, the true location remains the same as what it originally was.

Axiom 5.2.15 (MPC_{free})**.** *Given map ψ, byte representations $\{[\omega_0^{\mathrm{p}}, ..., \omega_{\alpha-1}^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$ and tag lists $\{\bar{j}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ such that*
$\mathrm{MPC}_{free}([[\omega_0^1, ..., \omega_{\alpha-1}^1], ..., [\omega_0^{\mathrm{q}}, ..., \omega_{\alpha-1}^{\mathrm{q}}]], [\bar{j}^1, ...\bar{j}^{\mathrm{q}}]) = ([[\omega_0'^1, ..., \omega_{\alpha-1}'^1], ..., [\omega_0'^{\mathrm{q}}, ..., \omega_{\alpha-1}'^{\mathrm{q}}]], [\bar{j}'^1, ..., \bar{j}'^{\mathrm{q}}])$,
if $\{\bar{j}'^{\mathrm{p}}[0] = \mathrm{encrypt}(1)\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\bar{j}'^{\mathrm{p}}[j] = \mathrm{encrypt}(1)\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{\forall i \neq j \in \{1...\alpha-1\}\bar{j}'^{\mathrm{p}}[i] = \mathrm{encrypt}(0)\}_{\mathrm{p}=1}^{\mathrm{q}}$
then $\{\omega_0^{\mathrm{p}} = \omega_j'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\omega_j^{\mathrm{p}} = \omega_0'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, *and* $\{\forall i \neq j \in \{1...\alpha-1\}, \omega_i^{\mathrm{p}} = \omega_i'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$
otherwise if $\{\bar{j}'^{\mathrm{p}}[0] = \mathrm{encrypt}(1)\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{\forall i \in \{1...\alpha-1\}\bar{j}'^{\mathrm{p}}[i] = \mathrm{encrypt}(0)\}_{\mathrm{p}=1}^{\mathrm{q}}$
then $\{\forall i \in \{0...\alpha-1\}, \omega_i^{\mathrm{p}} = \omega_i'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

5.2.4 Confluence

Definition 5.2.36 ($v^1 \sim v^2$)**.** Two values are *corresponding*, in symbols $v^1 \sim v^2$, if and only if either both v^1, v^2 are public (including locations) and $v^1 = v^2$, or v^1, v^2 are private and $\mathrm{Erase}(v^1) = \mathrm{Erase}(v^2)$.

Definition 5.2.37 ($\gamma^1 \sim \gamma^2$)**.** Two environments are *corresponding*, in symbols $\gamma^1 \sim \gamma^2$, if and only if $\gamma^1 = \gamma^2$.

Definition 5.2.38 ($\omega^1 \sim \omega^2$)**.** Two bytes are *corresponding*, in symbols $\omega^1 \sim \omega^2$, if and only if they are of the same type, and when decoded to values, $v^1 \sim v^2$.

Definition 5.2.39 ($\sigma^1 \sim \sigma^2$)**.** Two memories are *corresponding*, in symbols $\sigma^1 \sim \sigma^2$, if and only if $\forall l_1 \notin \sigma^1, l_1 \notin \sigma^2$, and $\forall l \in \sigma^1$ such that $\sigma^1(l) = (\omega^1, ty^1, \alpha^1, \mathrm{PermL}^1)$, $l \in \sigma^2$ such that $\sigma^2(l) = (\omega^2, ty^2, \alpha^2, \mathrm{PermL}^2)$ and $\omega^1 \sim \omega^2$, $ty^1 = ty^2$, $\alpha^1 = \alpha^2$, and $\mathrm{PermL}^1 = \mathrm{PermL}^2$.

Definition 5.2.40 ($\Delta^1 \sim \Delta^2$)**.** Two location maps are *corresponding*, in symbols $\Delta^1 \sim \Delta^2$, if and only if $\forall (l_1, \mu_1) \notin \Delta^1, (l_1, \mu_1) \notin \Delta^2$, and $\forall (l, \mu) \in \Delta^1$ such that $(l, \mu) \rightarrow (v_1^1, v_2^1, j^1, ty^1)$, $(l, \mu) \in \Delta^2$ such that $(l, \mu) \rightarrow (v_1^2, v_2^2, j^2, ty^2)$ and $v_1^1 \sim v_1^2$, $v_2^1 \sim v_2^2$, $j^1 \sim j^2$, and $ty^1 \sim ty^2$.

Definition 5.2.41 ($\mathrm{acc}^1 \sim \mathrm{acc}^2$)**.** Two accumulators are *corresponding*, in symbols $\mathrm{acc}^1 \sim \mathrm{acc}^2$, if and only if $\mathrm{acc}^1 = \mathrm{acc}^2$.

Definition 5.2.42 ($C^1 \sim C^2$)**.** Two configurations are *corresponding*, in symbols $C^1 \sim C^2$ or $(1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, s^1) \sim (2, \gamma^2, \sigma^2, \Delta^2, \mathrm{acc}^2, s^2)$, if and only if $\gamma^1 = \gamma^2$, $\sigma^1 \sim \sigma^2$, $\Delta^1 \sim \Delta^2$, $\mathrm{acc}^1 = \mathrm{acc}^2$, and $s^1 = s^2$.

Lemma 5.2.86 ($C^1 \sim C^2 \implies C^1 \cong_\psi \widehat{C} \wedge C^2 \cong_\psi \widehat{C}$)**.** *Given two configurations C^1, C^2 such that $C^1 = (1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, s^1)$ and $C^2 = (2, \gamma^2, \sigma^2, \Delta^2, \mathrm{acc}^2, s^2)$ and ψ, if $C^1 \sim C^2$ then $\{C^{\mathrm{p}} \cong_\psi (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, s)\}_{\mathrm{p}=1}^2$.*

Proof.

Proof Sketch:

Using the definition of Erase and Definition 5.2.42, there is only one possible Vanilla C configuration $\widehat{C}$ (modulo party ID) that can be obtained from both $\mathrm{Erase}(C^1)$ and $\mathrm{Erase}(C^2)$. □

Lemma 5.2.87 (Unique party-wise transitions)**.** *Given $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s) \parallel C)$ if $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s) \parallel C) \Downarrow_{\mathcal{D}}^{\mathcal{L}} ((\mathrm{p}, \gamma_1, \sigma_1, \Delta_1, \mathrm{acc}, v) \parallel C_1)$ then there exists no other rule by which $(\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s)$ can step.*

Proof.

Proof Sketch:

By induction on $(\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, s)$. We verify that for every configuration, given s, acc, and stored type information, there is only one corresponding semantic rule. □

Theorem 5.2.2 (Confluence)**.** *Given $C^1 \parallel ... \parallel C^{\mathrm{q}}$ such that $\{C^1 \sim C^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*
if $(C^1 \parallel ... \parallel C^{\mathrm{q}}) \Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} (C_1^1 \parallel ... \parallel C_1^{\mathrm{q}})$ such that $\exists \mathrm{p} \in \{1...\mathrm{q}\} C_1^1 \not\sim C_1^{\mathrm{p}}$,
then $\exists\, (C_1^1 \parallel ... \parallel C_1^{\mathrm{q}}) \Downarrow_{\mathcal{D}_2}^{\mathcal{L}_2} (C_2^1 \parallel ... \parallel C_2^{\mathrm{q}})$
such that $\{C_2^1 \sim C_2^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{(\mathcal{L}_1^1 :: \mathcal{L}_2^1) = (\mathcal{L}_1^{\mathrm{p}} :: \mathcal{L}_2^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, and $\{(\mathcal{D}_1^1 :: \mathcal{D}_2^1) = (\mathcal{D}_1^{\mathrm{p}} :: \mathcal{D}_2^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Proof.

Proof Sketch:

By Lemma 5.2.87, we have that there is only one possible execution trace for any given party based on the starting configuration.

By definition of $\{C^1 \sim C^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, we have that the starting states of all parties are corresponding, with identical statements.

Therefore, all parties must follow the same execution trace and will eventually reach another set of corresponding states. $\square$

5.2.5 Proof of Correctness

Theorem 5.2.3 (Semantic Correctness)**.**

For every configuration $\{(\mathrm{p},\ \gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}},\ \Delta^{\mathrm{p}},\ \mathrm{acc}^{\mathrm{p}},\ s^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{(\mathrm{p},\ \widehat{\gamma}^{\mathrm{p}},\ \widehat{\sigma}^{\mathrm{p}},\ \square,\ \square,\ \widehat{s}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and map* ψ

such that $\{(\mathrm{p}, \gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}},\ \Delta^{\mathrm{p}},\ \mathrm{acc}^{\mathrm{p}},\ s^{\mathrm{p}}) \cong_{\psi} (\mathrm{p},\ \widehat{\gamma}^{\mathrm{p}},\ \widehat{\sigma}^{\mathrm{p}},\ \square,\ \square,\ \widehat{s}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$,

if $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \mathrm{acc}^1,\ s^1)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}},\ \sigma^{\mathrm{q}},\ \Delta^{\mathrm{q}},\ \mathrm{acc}^{\mathrm{q}},\ s^{\mathrm{q}}))$

$\Downarrow_{\mathcal{D}}^{\mathcal{L}} ((1, \gamma_1^1,\ \sigma_1^1,\ \Delta_1^1,\ \mathrm{acc}_1^1,\ v^1)\ \|\ ...\ \|\ (\mathrm{q}, \gamma_1^{\mathrm{q}},\ \sigma_1^{\mathrm{q}},\ \Delta_1^{\mathrm{q}},\ \mathrm{acc}_1^{\mathrm{q}},\ v^{\mathrm{q}}))$

for codes $\mathcal{D} \in SmcC$*, then there exists a derivation*

$\Sigma \triangleright\ ((1, \widehat{\gamma}^1,\ \widehat{\sigma}^1,\ \square,\ \square,\ \widehat{s}^1)\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma}^{\mathrm{q}},\ \widehat{\sigma}^{\mathrm{q}},\ \square,\ \square,\ \widehat{s}^{\mathrm{q}}))$

$\Downarrow_{\widehat{\mathcal{D}}} ((1, \widehat{\gamma}_1^1,\ \widehat{\sigma}_1^1,\ \square,\ \square,\ \widehat{v}^1)\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma}_1^{\mathrm{q}},\ \widehat{\sigma}_1^{\mathrm{q}},\ \square,\ \square,\ \widehat{v}^{\mathrm{q}}))$

for codes $\widehat{\mathcal{D}} \in VanC$ *and a map* ψ_1 *such that*

$\mathcal{D} \cong \widehat{\mathcal{D}}$, $\{(\mathrm{p}, \gamma_1^{\mathrm{p}},\ \sigma_1^{\mathrm{p}},\ \Delta_1^{\mathrm{p}},\ \mathrm{acc}_1^{\mathrm{p}},\ v^{\mathrm{p}}) \cong_{\psi_1} (\mathrm{p},\ \widehat{\gamma}_1^{\mathrm{p}},\ \widehat{\sigma}_1^{\mathrm{p}},\ \square,\ \square,\ \widehat{v}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, *and* $\Pi \cong_{\psi_1} \Sigma$.

Proof.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1\ bop\ e_2)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1\ bop\ e_2)) \Downarrow_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpb])}^{\mathcal{L}_1::\mathcal{L}_2}$
$((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n_3^1)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n_3^{\mathrm{q}}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1\ bop\ e_2)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1\ bop\ e_2)) \Downarrow_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpb])}^{\mathcal{L}_1::\mathcal{L}_2}$ $((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n_3^1)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n_3^{\mathrm{q}}))$, by SMC^2 rule Multiparty Binary Operation we have $\{(e_1, e_2) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (B) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1)) \Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n_1^1)\ \|\ ...\ \|$ $(\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n_1^{\mathrm{q}}))$, (C) $((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, e_2)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, e_2)) \Downarrow_{\mathcal{D}_2}^{\mathcal{L}_2} ((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n_2^1)$ $\|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n_2^{\mathrm{q}}))$, (D) $\mathrm{MPC}_b(bop, [n_1^1, ..., n_1^{\mathrm{q}}], [n_2^1, ..., n_2^{\mathrm{q}}]) = (n_3^1, ..., n_3^{\mathrm{q}})$, and (E) $bop \in \{\cdot, +, -, \div\}$.

Given (A), $((1, \widehat{\gamma}^1,\ \widehat{\sigma}^1,\ \square, \square,\ \widehat{e}_1 bop\ \widehat{e}_2)\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma}^{\mathrm{q}},\ \widehat{\sigma}^{\mathrm{q}},\ \square, \square,\ \widehat{e}_1 bop\ \widehat{e}_2))$ and ψ such that $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, e_1\ bop\ e_2) \cong_{\psi} (\mathrm{p}, \widehat{\gamma}^{\mathrm{p}},\ \widehat{\sigma}^{\mathrm{p}},\ \square, \square,\ \widehat{e}_1 bop\ \widehat{e}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$, by Definition 5.2.22 we have $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}^{\mathrm{p}},\ \widehat{\sigma}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (F) $e_1 bop\ e_2 \cong_{\psi} \widehat{e}_1 bop\ \widehat{e}_2$. By Definition 5.2.20 we have $bop = bop$, (G) $e_1 \cong_{\psi} \widehat{e}_1$ and (H) $e_2 \cong_{\psi} \widehat{e}_2$.

Given Axiom 5.2.1, by Theorem 5.2.2 we have $\{(1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1\ bop\ e_2) \sim (\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, e_1\ bop\ e_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$. By Lemma 5.2.86, we have $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, e_1\ bop\ e_2) \cong_{\psi} (\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma},\ \square, \square,\ \widehat{e}_1 bop\ \widehat{e}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$. and therefore (I) $((1, \widehat{\gamma},\ \widehat{\sigma},\ \square, \square,\ \widehat{e}_1 bop\ \widehat{e}_2)\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma},\ \widehat{\sigma},\ \square, \square,\ \widehat{e}_1 bop\ \widehat{e}_2))$. By Definition 5.2.22 we have (J) $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}) \cong_{\psi}$

$(\widehat{\gamma},\ \widehat{\sigma})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (B), (J), (G), and ψ, by Lemma 5.2.76 we have $((1,\widehat{\gamma},\ \widehat{\sigma},\ \Box,\Box,\ \widehat{e}_1) \parallel ... \parallel\ (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma},\ \Box,\Box,\ \widehat{e}_1))$ such that $\{(\mathrm{p},\gamma^{\mathrm{p}},\sigma^{\mathrm{p}},\Delta^{\mathrm{p}},\mathrm{acc},e_1) \cong_{\psi} (\mathrm{p},\widehat{\gamma},\ \widehat{\sigma},\ \Box,\Box,\ \widehat{e}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$. By the inductive hypothesis, we have (K) $((1,\widehat{\gamma},\ \widehat{\sigma},\ \Box,\Box,\ \widehat{e}_1) \parallel ... \parallel\ (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma},\ \Box,\Box,\ \widehat{e}_1)) \Downarrow'_{\widehat{\mathcal{D}}_1} ((1,\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\Box,\ \widehat{n}_1) \parallel ... \parallel\ (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\Box,\ \widehat{n}_1))$ and ψ_1 such that $\{(\mathrm{p},\gamma^{\mathrm{p}},\sigma_1^{\mathrm{p}},\Delta_1^{\mathrm{p}},\mathrm{acc},n_1^{\mathrm{p}}) \cong_{\psi_1} (\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\Box,\ \widehat{n}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. By Definition 5.2.22 we have (L) $\{(\gamma^{\mathrm{p}},\ \sigma_1^{\mathrm{p}}) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\{n_1^{\mathrm{p}} \cong_{\psi_1} \widehat{n}_1\}_{\mathrm{p}=1}^{\mathrm{q}}$. By Definition 5.2.19 we have (M) $\{n_1^{\mathrm{p}} \cong \widehat{n}_1\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given Axiom 5.2.1, we have $(l,\mu) \notin e_2$. Given (H), by Lemma 5.2.55 we have $e_2 \cong_{\psi_1} \widehat{e}_2$. Therefore, given (C), (L), and ψ_1, by Lemma 5.2.76 we have $((1,\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\Box,\ \widehat{e}_2) \parallel ... \parallel\ (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\Box,\ \widehat{e}_2))$ such that $\{(\mathrm{p},\gamma^{\mathrm{p}},\sigma_1^{\mathrm{p}},\Delta_1^{\mathrm{p}},\mathrm{acc},e_2) \cong_{\psi_1} (\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\Box,\ \widehat{e}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$. By the inductive hypothesis, we have (N) $((1,\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\Box,\ \widehat{e}_2) \parallel ... \parallel\ (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_1,\ \Box,\Box,\ \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_2} ((1,\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\Box,\ \widehat{n}_2) \parallel ... \parallel\ (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\Box,\ \widehat{n}_2))$ and ψ_2 such that $\{(\mathrm{p},\gamma^{\mathrm{p}},\sigma_2^{\mathrm{p}},\Delta_2^{\mathrm{p}},\ \mathrm{acc},\ n_2^{\mathrm{p}}) \cong_{\psi_2} (\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\Box,\ \widehat{n}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$. By Definition 5.2.22 we have (O) $\{(\gamma^{\mathrm{p}},\ \sigma_2^{\mathrm{p}}) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\{n_2^{\mathrm{p}} \cong_{\psi_2} \widehat{n}_2\}_{\mathrm{p}=1}^{\mathrm{q}}$. By Definition 5.2.19 we have (P) $\{n_2^{\mathrm{p}} \cong \widehat{n}_2\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (D), (M), and (P), by Axiom 5.2.4 we have (Q) $\{n_3^{\mathrm{p}} \cong \widehat{n}_3\}_{\mathrm{p}=1}^{\mathrm{q}}$ such that (R) $\widehat{n}_1\ bop\ \widehat{n}_2 = \widehat{n}_3$.

Given (I), (K), (N), (R), (E) and $bop = bop$, we have $\Sigma \triangleright ((1,\widehat{\gamma},\ \widehat{\sigma},\ \Box,\Box,\ \widehat{e}_1\ bop\ \widehat{e}_2) \parallel ... \parallel\ (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma},\ \Box,\Box,\ \widehat{e}_1\ bop\ \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::[(\mathrm{ALL},[\widehat{mpb}])]} ((1,\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\Box,\ \widehat{n}_3) \parallel ... \parallel\ (\mathrm{q},\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\Box,\ \widehat{n}_3))$ by Vanilla C rule Multiparty Binary Operation.

Given (O) and (Q), by Definition 5.2.22 we have $\{(\mathrm{p},\gamma^{\mathrm{p}},\sigma_2^{\mathrm{p}},\Delta_2^{\mathrm{p}},\mathrm{acc},n_3^{\mathrm{p}}) \cong_{\psi_2} (\mathrm{p},\widehat{\gamma},\ \widehat{\sigma}_2,\ \Box,\Box,\ \widehat{n}_3)\}_{\mathrm{p}=1}^{\mathrm{q}}$.
By Definition 5.2.23 we have $mpb \cong \hat{mpb}$. Given $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1, \mathcal{D}_2 \cong \widehat{\mathcal{D}}_2, \mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{ALL},[mpb])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: [(\mathrm{ALL},[\widehat{mpb}])]$ by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{ALL},[mpb]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: [(\mathrm{ALL},[\widehat{mpb}])]$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_2} \Sigma$.

Case $\Pi \triangleright ((1,\gamma^1,\sigma^1,\Delta^1,\mathrm{acc}^1,e_1\ bop\ e_2) \parallel ... \parallel\ (\mathrm{q},\gamma^{\mathrm{q}},\sigma^{\mathrm{q}},\Delta^{\mathrm{q}},\mathrm{acc}^{\mathrm{q}},e_1\ bop\ e_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpcmp])}$
$((1,\gamma^1,\sigma_2^1,\Delta_2^1,\mathrm{acc}^1,n_3^1) \parallel ... \parallel\ (\mathrm{q},\gamma^{\mathrm{q}},\sigma_2^{\mathrm{q}},\Delta_2^{\mathrm{q}},\mathrm{acc}^{\mathrm{q}},n_3^{\mathrm{q}}))$

Given (A) $\Pi \triangleright ((1,\gamma^1,\sigma^1,\Delta^1,\mathrm{acc}^1,e_1\ bop\ e_2) \parallel ... \parallel\ (\mathrm{q},\gamma^{\mathrm{q}},\sigma^{\mathrm{q}},\Delta^{\mathrm{q}},\mathrm{acc}^{\mathrm{q}},e_1\ bop\ e_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpcmp])} ((1,\gamma^1,\sigma_2^1,\ \Delta_2^1,\mathrm{acc}^1,n_3^1) \parallel ... \parallel\ (\mathrm{q},\gamma^{\mathrm{q}},\sigma_2^{\mathrm{q}},\Delta_2^{\mathrm{q}},\mathrm{acc}^{\mathrm{q}},n_3^{\mathrm{q}}))$ by SMC^2 rule Multiparty Comparison Operation, we have

$\{(e_1, e_2) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (B) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1)) \Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n_1^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n_1^{\mathrm{q}}))$, (C) $((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, e_2)) \Downarrow_{\mathcal{D}_2}^{\mathcal{L}_2} ((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n_2^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n_2^{\mathrm{q}}))$, (D) $\mathrm{MPC}_{cmp}(bop, [n_1^1, ..., n_1^{\mathrm{q}}], [n_2^1, ..., n_2^{\mathrm{q}}]) = (n_3^1, ..., n_3^{\mathrm{q}})$, and (E) $bop \in \{==, !=, <\}$.

Given (A), $((1, \widehat{\gamma}^1, \widehat{\sigma}^1, \Box, \Box, \widehat{e}_1 \, bop \, \widehat{e}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}^{\mathrm{q}}, \widehat{\sigma}^{\mathrm{q}}, \Box, \Box, \widehat{e}_1 \, bop \, \widehat{e}_2))$ and ψ such that $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, e_1 \; bop \; e_2) \cong_\psi (\mathrm{p}, \widehat{\gamma}^{\mathrm{p}}, \widehat{\sigma}^{\mathrm{p}}, \Box, \Box, \widehat{e}_1 \, bop \, \widehat{e}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$, by Definition 5.2.22 we have $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \cong_\psi (\widehat{\gamma}^{\mathrm{p}}, \widehat{\sigma}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (F) $e_1 \, bop \, e_2 \cong_\psi \widehat{e}_1 \, bop \, \widehat{e}_2$. Given (F), by Definition 5.2.20 we have (G) $e_1 \cong_\psi \widehat{e}_1$, (H) $e_2 \cong_\psi \widehat{e}_2$, and $bop = bop$.

Given Axiom 5.2.1, by Theorem 5.2.2 we have $\{(1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1 \; bop \; e_2) \sim (\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, e_1 \; bop \; e_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$. By Lemma 5.2.86, we have $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, e_1 \; bop \; e_2) \cong_\psi (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1 \, bop \, \widehat{e}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and therefore (I) $((1, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1 \, bop \, \widehat{e}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1 \, bop \, \widehat{e}_2))$. By Definition 5.2.22 we have (J) $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (B), (J), (G), and ψ, by Lemma 5.2.76 we have $((1, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1))$ such that $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, e_1) \cong_\psi (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$. By the inductive hypothesis, we have (K) $((1, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1)) \Downarrow'_{\widehat{\mathcal{D}}_1} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}_1) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}_1))$ and ψ_1 such that $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, \Delta_1^{\mathrm{p}}, \mathrm{acc}, n_1^{\mathrm{p}}) \cong_{\psi_1} (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. By Definition 5.2.22 we have (L) $\{(\gamma^{\mathrm{p}}, \sigma_1^{\mathrm{p}}) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\{n_1^{\mathrm{p}} \cong_{\psi_1} \widehat{n}_1\}_{\mathrm{p}=1}^{\mathrm{q}}$. By Definition 5.2.19 we have (M) $\{n_1^{\mathrm{p}} \cong \widehat{n}_1\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given Axiom 5.2.1, we have $(l, \mu) \notin e_2$. Given (H), by Lemma 5.2.55 we have $e_2 \cong_{\psi_1} \widehat{e}_2$. Therefore, given (C), (E), (L), and ψ_1, by Lemma 5.2.76 we have $((1, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{e}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{e}_2))$ such that $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, \Delta_1^{\mathrm{p}}, \mathrm{acc}, e_2) \cong_\psi (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{e}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$. By the inductive hypothesis, we have (N) $((1, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{e}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_2} ((1, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \widehat{n}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \widehat{n}_2))$ and ψ_2 such that $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma_2^{\mathrm{p}}, \Delta_2^{\mathrm{p}}, \mathrm{acc}, n_2^{\mathrm{p}}) \cong_{\psi_2} (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \widehat{n}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$. By Definition 5.2.22 we have (O) $\{(\gamma^{\mathrm{p}}, \sigma_2^{\mathrm{p}}) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\{n_2^{\mathrm{p}} \cong_{\psi_2} \widehat{n}_2\}_{\mathrm{p}=1}^{\mathrm{q}}$. By Definition 5.2.19 we have (P) $\{n_2^{\mathrm{p}} \cong \widehat{n}_2\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (D), (M), and (P), by Axiom 5.2.5 we have (Q) $\{n_3^{\mathrm{p}} \cong \widehat{n}_3\}_{\mathrm{p}=1}^{\mathrm{q}}$ such that (R) $(\widehat{n}_1 \; bop \; \widehat{n}_2) = \widehat{n}_3$.

Subcase (S1) $\widehat{n}_3 = 1$

Given (I), (K), (N), (R), (S1), (E), and $bop = bop$, we have $\Sigma \triangleright ((1, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1 \; bop \; \widehat{e}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1 \; bop \; \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::[(\mathrm{ALL}, [\widehat{mpcmpt}])]} ((1, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \widehat{n}_3) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \widehat{n}_3))$ by Vanilla C rule Multiparty

Comparison True Operation.

Given (O) and (Q), by Definition 5.2.22 we have $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma_2^{\mathrm{p}}, \Delta_2^{\mathrm{p}}, \mathrm{acc}, n_3^{\mathrm{p}}) \cong_\psi (\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma}_2,\ \Box, \Box,\ \widehat{n}_3)\}_{\mathrm{p}=1}^{\mathrm{q}}$.
By Definition 5.2.23 we have $mpcmp \cong \hat{mpcmpt}$.
Given $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$, $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$, $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{ALL}, [mpcmp])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: [(\mathrm{ALL}, [\widehat{mpcmpt}])]$ by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{ALL}, [mpcmp]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: [(\mathrm{ALL}, [\widehat{mpcmpt}])]$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Subcase (S2) $\widehat{n}_3 = 0$

Given (I), (K), (N), (R), (S2), (E), and $bop = bop$, we have $\Sigma \triangleright ((1, \widehat{\gamma},\ \widehat{\sigma},\ \Box, \Box,\ \widehat{e}_1\ bop\ \widehat{e}_2)\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma},\ \widehat{\sigma},\ \Box, \Box, \widehat{e}_1\ bop\ \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: [(\mathrm{ALL}, [\widehat{mpcmpf}])]} ((1, \widehat{\gamma},\ \widehat{\sigma}_2,\ \Box, \Box,\ \widehat{n}_3)\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma},\ \widehat{\sigma}_2,\ \Box, \Box,\ \widehat{n}_3))$ by Vanilla C rule Multiparty Comparison False Operation.

Given (O) and (Q), by Definition 5.2.22 we have $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma_2^{\mathrm{p}}, \Delta_2^{\mathrm{p}}, \mathrm{acc}, n_3^{\mathrm{p}}) \cong_\psi (\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma}_2,\ \Box, \Box,\ \widehat{n}_3)\}_{\mathrm{p}=1}^{\mathrm{q}}$.
By Definition 5.2.23 we have $mpcmp \cong \hat{mpcmpf}$.
Given $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$, $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$, $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{ALL}, [mpcmp])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: [(\mathrm{ALL}, [\widehat{mpcmpf}])]$ by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{ALL}, [mpcmp]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: [(\mathrm{ALL}, [\widehat{mpcmpf}])]$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)\ \|\ C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [iet])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip})\ \|\ C_2)$

Given $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)\ \|\ C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [iet])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip})\ \|\ C_2)$ by SMC^2 rule Public If Else True, we have $(e) \nvdash \gamma$, (A) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e)\ \|\ C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n)\ \|\ C_1)$, (B) $n \neq 0$, and (C) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ s_1)\ \|\ C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma_1,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip})\ \|\ C_2)$.

Given $(\Box, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2)$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)\ \|\ C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2)\ \|\ \widehat{C})$, by Definition 5.2.22 we have (D) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and $\mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2 \cong_\psi \mathrm{if}\ (\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2$ and (E) $C \cong_\psi \widehat{C}$. By Definition 5.2.20, we have (F) $e \cong_\psi \widehat{e}$, (G) $s_1 \cong_\psi \widehat{s}_1$, and (H) $s_2 \cong_\psi \widehat{s}_2$.

Given (D), ψ, (E), and (F), by Lemma 5.2.50 we have $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e)\ \|\ C) \cong_\psi ((\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma}, \Box, \Box, \widehat{e})\ \|\ \widehat{C})$. Given

(A), by the inductive hypothesis we have (I) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \hat{n}) \parallel \widehat{C}_1)$ and ψ_1 such that $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \hat{n}) \parallel \widehat{C}_1)$ and (J) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. By Definition 5.2.22 we have (K) $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$, (L) $C_1 \cong_{\psi_1} \widehat{C}_1$, and $n \cong_{\psi_1} \hat{n}$. By Definition 5.2.19 we have $n \cong \hat{n}$.

Given $(e) \nvdash \gamma$, we have $(n) \nvdash \gamma$ and therefore $n = \hat{n}$. Given (B), we have (M) $\hat{n}\ \neq\ 0$.

Given Axiom 5.2.1, we have $(l, \mu) \notin s_1$. Given (G), by Lemma 5.2.55 we have $s_1 \cong_{\psi_1} \widehat{s}_1$. Therefore, given (K), ψ_1, and (L), by Lemma 5.2.50 we have $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ s_1) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_1) \parallel \widehat{C}_1)$. Given (C), by the inductive hypothesis, we have (N) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_1) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$ and ψ_2 such that $((\mathrm{p}, \gamma_1,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$ and (O) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$. By Definition 5.2.22, we have $(\gamma_1, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$ and (P) $C_2 \cong_{\psi_2} \widehat{C}_2$. By Lemma 5.2.57, we have (Q) $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2) \parallel \widehat{C})$ and (I), (M), and (N), we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2) \parallel \widehat{C})$ $\Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{iet}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$ by Vanilla C rule If Else True.

Given (P) and (Q), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$. By Definition 5.2.23 we have $iet \cong \hat{iet}$. Given (J), (O), $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [iet])$ and
$\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{iet}])$ by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [iet]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{iet}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p},\ \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ief])} ((\mathrm{p},\ \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[iet])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{while}\ (e)\ s) \parallel C) \Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wle])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{while}\ (e)\ s) \parallel C) \Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wle])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule While End, we have $(e) \nvdash \gamma$, (A) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C)$, $\Downarrow^{\mathcal{L}}_{\mathcal{D}} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1)$, (B) $n\ ==\ 0$.

Given $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{while}(\widehat{e})\ \widehat{s}) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{while}\ (e)\ s) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square,$

$\text{while}(\widehat{e})\ \widehat{s}) \parallel \widehat{C})$, by Definition 5.2.22 we have (C) $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, (D) $C \cong_{\psi} \widehat{C}$ and while $(e)\ s \cong_{\psi}$ while $(\widehat{e})\ \widehat{s}$. By Definition 5.2.20 we have (E) $e \cong_{\psi} \widehat{e}$ and $s \cong_{\psi} \widehat{s}$

Given (C), (D), (E), and ψ, by Lemma 5.2.50 we have $((\text{p}, \gamma, \sigma, \Delta, \text{acc}, e) \parallel C) \cong_{\psi} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}) \parallel \widehat{C})$. By the inductive hypothesis, we have (F) $((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}) \parallel \widehat{C})$ such that $((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, n)$-$\parallel C_1) \cong_{\psi_1} ((\Box, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}) \parallel \widehat{C}_1)$ and (G) $\mathcal{D} \cong \widehat{\mathcal{D}}$.

By Definition 5.2.22 we have (H) $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_{\psi_1} \widehat{n}$. By Definition 5.2.19 we have $n \cong \widehat{n}$. Given $(e) \nvdash \gamma$, we have $(n) \nvdash \gamma$ and therefore (I) $n = \widehat{n}$.

Given (B) and (I), we have (J) $\widehat{n}\ =\ 0$.

Given $((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \text{while}(\widehat{e})\ \widehat{s}) \parallel \widehat{C})$, (F), and (J), we have $\Sigma \triangleright ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \text{while}(\widehat{e})\ \widehat{s}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}::(\text{p}, [\widehat{wle}])}$ $((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \text{skip}) \parallel \widehat{C}_1)$ by Vanilla C rule While End.

Given (H), by Definition 5.2.22 we have $((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1) \cong_{\psi_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \text{skip}) \parallel \widehat{C}_1)$.
By Definition 5.2.23 we have $wle \cong \widehat{wle}$. Given (G), $\mathcal{D}_1 :: (\text{p}, [wle])$ and
$\widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{wle}])$ by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\text{p}, [wle]) \cong \widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{wle}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{while}\ (e)\ s) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p}, [wlc])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{while}\ (e)\ s) \parallel C_2)$

Given $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{while}\ (e)\ s) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p}, [wlc])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{while}\ (e)\ s) \parallel C_2)$ by SMC^2 rule While Continue, we have $(e) \nvdash \gamma$, (A) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ n) \parallel C_1)$, (B) $n\ \neq\ 0$, and (C) $((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ s) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\text{p}, \gamma_1,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{skip})\ \parallel C_2)$.

Given (D) $((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \text{while}(\widehat{e})\ \widehat{s}) \parallel \widehat{C})$ and ψ such that $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{while}\ (e)\ s) \parallel C) \cong_{\psi} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box,$ $\text{while}(\widehat{e})\ \widehat{s}) \parallel \widehat{C})$, by Definition 5.2.22 we have (E) $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, $C \cong_{\psi} \widehat{C}$, and (F) while $(e)\ s \cong_{\psi}$ while $(\widehat{e})\ \widehat{s}$. By Definition 5.2.20 we have (G) $e \cong_{\psi} \widehat{e}$ and (H) $s \cong_{\psi} \widehat{s}$.

Given (D), ψ, (E), $C \cong_{\psi} \widehat{C}$, and (H), by Lemma 5.2.50 we have $((\text{p}, \gamma, \sigma, \Delta, \text{acc}, e) \parallel C) \cong_{\psi} ((\text{p}, \widehat{\gamma},\ \widehat{\sigma}, \Box, \Box, \widehat{e}) \parallel \widehat{C})$. Given (A), by the inductive hypothesis we have (I) $((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \hat{n}) \parallel \widehat{C}_1)$ and ψ_1

such that $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$ and (J) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$.

By Definition 5.2.22 we have (K) $C_1 \cong_{\psi_1} \widehat{C}_1$, (L) $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$ and $n \cong_{\psi_1} \widehat{n}$. By Definition 5.2.19 we have $n \cong \widehat{n}$. Given $(e) \nvdash \gamma$, we have $(n) \nvdash \gamma$ and therefore (M) $n = \widehat{n}$.

Given (B) and (M), we have (N) $\widehat{n} \neq 0$.

Given Axiom 5.2.1, we have $(l, \mu) \notin s$. Given (H), by Lemma 5.2.55 we have $s \cong_{\psi_1} \widehat{s}$. Therefore, given (L), ψ_1, and (K), by Lemma 5.2.50 we have $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ s) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}) \parallel \widehat{C}_1)$. Given (C), by the inductive hypothesis, we have (O) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$ and ψ_2 such that $((\mathrm{p}, \gamma_1,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$ and (P) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$. By Definition 5.2.22, we have $(\gamma_1, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$ and (Q) $C_2 \cong_{\psi_2} \widehat{C}_2$. By Lemma 5.2.57, we have (R) $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given (D), (I), (N), and (O), we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{while}(\widehat{e})\widehat{s}) \parallel \widehat{C})$
$\Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{wlc}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{while}(\widehat{e})\widehat{s}) \parallel \widehat{C}_2)$ by Vanilla C rule While Continue.

Given Axiom 5.2.1, we have $(l, \mu) \notin \mathrm{while}\ (e)\ s$. Therefore, given (F), by Lemma 5.2.55 we have (S) $\mathrm{while}\ (e)\ s \cong_{\psi_2} \mathrm{while}\ (\widehat{e})\ \widehat{s}$.

Given (S), (R), and (Q), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{while}\ (e)\ s) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{while}\ (\widehat{e})\ \widehat{s}) \parallel \widehat{C}_2)$.
By Definition 5.2.23 we have $wlc \cong \hat{wlc}$. Given (J) and (O), $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wlc])$ and
$\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{wlc}])$ by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wlc]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{wlc}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p}, [(l,0)])}_{(\mathrm{p}, [dp])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

Given $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p}, [(l,0)])}_{(\mathrm{p}, [dp])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ by SMC^2 rule Public Pointer Declaration, we have (A) $(ty = \mathrm{public}\ bty*)$, $\mathrm{acc} = 0$, (B) $l = \phi()$, (C) $\mathrm{GetIndirection}(*) = i$, (D) $\omega = \mathrm{EncodePtr}(\mathrm{public}\ bty*, [1, [(l_{default}, 0)], [1], i])$, (E) $\gamma_1 = \gamma[x \rightarrow (l,\ \mathrm{public}\ bty*)]$, and (F) $\sigma_1 = \sigma[l \rightarrow (\omega,\ \mathrm{public}\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{public}\ bty*, \mathrm{public}, 1))]$.

Given (G) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x}) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x}) \parallel \widehat{C})$, by Definition 5.2.22 we have (H) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, $C \cong_\psi \widehat{C}$, and (I) $ty\ x \cong_\psi \widehat{ty}\ \widehat{x}$. By Definition 5.2.20 we have (J) $ty \cong_\psi \widehat{ty}$ such that (K) $* = *$ and $x \cong_\psi \widehat{x}$. Therefore, we have (L) $x = \widehat{x}$.

Given (C) and (K), by Lemma 5.2.1 we have (M) $\mathrm{GetIndirection}(*) = \widehat{i}$ such that (N) $i = \widehat{i}$.

Given (B), by Axiom 5.2.2 we have (O) $\widehat{l} = \phi()$ and (P) $l = \widehat{l}$.

Given (D), (J), (N), and $[1, [(l_{default}, 0)], [1], i] \cong_\psi [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}]$ by Definition 5.2.15, by Lemma 5.2.4 we have (Q) $\widehat{\omega} = \mathrm{EncodePtr}(\widehat{bty}*, [1, [(\widehat{l}_{default}, 0)], [1], \widehat{i}])$ such that (R) $\omega \cong_\psi \widehat{\omega}$.

Given (E), (L), (P), (H), (A), and (I), by Lemma 5.2.60 we have (S) $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l},\ \widehat{ty})]$ such that (T) $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given (F), (T), (P), (R), and (J), by Lemma 5.2.61 we have (U) $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega},\ \widehat{ty},\ 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}, \mathrm{public}, 1))]$ such that (V) $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given (A) and (J), by Definition 5.2.8 we have (W) $(\widehat{ty} = \widehat{bty}*)$.

Given (G), (M), (O), (Q), (S), (U), and (W) we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p}, [\widehat{dp}])} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \parallel \widehat{C})$ by Vanilla C rule Pointer Declaration.

Given (U) and $C \cong_\psi \widehat{C}$, by Definition 5.2.22 we have $((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \parallel \widehat{C})$.
By Definition 5.2.23 we have $dp \cong \hat{dp}$ and by Definition 5.2.25 we have $(\mathrm{p}, [dp]) \cong (\mathrm{p}, [\widehat{dp}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p}, [(l,0)])}_{(\mathrm{p}, [dp1])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p}, [(l,0)])}_{(\mathrm{p}, [dp])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 + e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bp])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2)$

Given $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 + e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bp])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2)$ by SMC^2 rule Public Addition, we have (A) $(e_1, e_2) \nvdash \gamma$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1)$, (C) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2)$, and (D) $n_1 + n_2 = n_3$.

Given $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1 + \widehat{e}_2) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 + e_2) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1 + \widehat{e}_2)$-$\parallel \widehat{C})$, by Definition 5.2.22 we have (E) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and (F) $C \cong_\psi \widehat{C}$. $e_1 + e_2 \cong_\psi \widehat{e}_1 + \widehat{e}_2$. By Definition 5.2.20 we have (G) $e_1 \cong_\psi \widehat{e}_1$ and (H) $e_2 \cong_\psi \widehat{e}_2$.

Given (E), ψ, (F), and (G), by Lemma 5.2.50 we have $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e_1) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma}, \Box, \Box, \widehat{e}_1) \parallel \widehat{C})$. Given (B), by the inductive hypothesis we have (I) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \hat{n}_1) \parallel \widehat{C}_1)$ and ψ_1 such that $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \hat{n}_1) \parallel \widehat{C}_1)$ and (J) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. By Definition 5.2.22 we have (K) $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$, $n_1 \cong_{\psi_1} \widehat{n}_1$, and (L) $C_1 \cong_{\psi_1} \widehat{C}_1$. Given (A), we have $(n_1) \nvdash \gamma$ and therefore by Definition 5.2.19 (M) $n_1 = \widehat{n}_1$.

Given Axiom 5.2.1, we have $(l, \mu) \notin e_2$. Given (H), by Lemma 5.2.55 we have $e_2 \cong_{\psi_1} \widehat{e}_2$. Therefore, given (K), ψ_1, and (L), by Lemma 5.2.50 we have $((\mathrm{p}, \gamma, \sigma_1, \Delta, \mathrm{acc}, e_2) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma}_1, \Box, \Box, \widehat{e}_2) \parallel \widehat{C})$. Given (C), by the inductive hypothesis we have (N) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \hat{n}_2) \parallel \widehat{C}_2)$ and ψ_2 such that $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \hat{n}_2) \parallel \widehat{C}_2)$ and (O) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$. By Definition 5.2.22 we have (P) $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$, (Q) $C_2 \cong_{\psi_2} \widehat{C}_2$ and $n_2 \cong_{\psi_2} \widehat{n}_2$. Given (A), we have $(n_2) \nvdash \gamma$ and therefore by Definition 5.2.19 (R) $n_2 = \widehat{n}_2$.

Given (D), (M), and (R), we have (S) $\widehat{n}_1 + \widehat{n}_2 = \widehat{n}_3$ such that $n_3 = \widehat{n}_3$ and therefore by Definition 5.2.19 (T) $n_3 \cong_{\psi_2} \widehat{n}_3$.

Given $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1 + \widehat{e}_2) \parallel \widehat{C})$, (I), (N), and (S), by Vanilla C rule Addition we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma},\ \Box, \Box, \widehat{e}_1 + \widehat{e}_2)$ $\parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{bp}])} ((\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma}_2,\ \Box, \Box,\ \widehat{n}_3) \parallel \widehat{C}_2)$.

Given (P), (Q), and (T), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \widehat{n}_3) \parallel \widehat{C}_2)$. By Definition 5.2.23 we have $bp \cong \hat{bp}$. Given (J), (O), $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [bp])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{bp}])$ by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [bp]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{bp}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 - e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bs])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e_1 + e_2) \ \parallel\ C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bp])} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, n_3) \ \parallel\ C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 \cdot e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bm])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e_1 + e_2) \ \parallel\ C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bp])} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, n_3) \ \parallel\ C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 \div e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bd])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e_1 + e_2) \ \parallel\ C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bp])} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, n_3) \ \parallel\ C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$

Given $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$ by SMC^2 rule Public Less Than True, we have (A) $(e_1, e_2) \nvdash \gamma$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1)$, (C) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2)$, and (D) $(n_1 < n_2) = 1$.

Given $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1 < \widehat{e}_2) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1 < \widehat{e}_2) \parallel \widehat{C})$, by Definition 5.2.22 we have (E) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (F) $C \cong_\psi \widehat{C}$ and $e_1 < e_2 \cong_\psi \widehat{e}_1 < \widehat{e}_2$. By Definition 5.2.20 we have (G) $e_1 \cong_\psi \widehat{e}_1$ and (H) $e_2 \cong_\psi \widehat{e}_2$.

Given (E), ψ, (F), and (G), by Lemma 5.2.50 we have $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e_1) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma}, \Box, \Box, \widehat{e}_1) \parallel \widehat{C})$. Given (B), by the inductive hypothesis we have (I) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}_1) \parallel \widehat{C}_1)$ and ψ_1 such that $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}_1) \parallel \widehat{C}_1)$ and (J) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. (K) $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$, (L) $C_1 \cong_{\psi_1} \widehat{C}_1$, and $n_1 \cong_\psi \widehat{n}_1$. Given (A), we have $(n_1) \nvdash \gamma$ and therefore by Definition 5.2.19 we have (M) $n_1 = \widehat{n}_1$.

Given Axiom 5.2.1, we have $(l, \mu) \notin e_2$. Given (H), by Lemma 5.2.55 we have $e_2 \cong_{\psi_1} \widehat{e}_2$. Therefore, given (K), ψ, and (L), by Lemma 5.2.50 we have $((\mathrm{p}, \gamma, \sigma_1, \Delta, \mathrm{acc}, e_2) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma}_1, \Box, \Box, \widehat{e}_2) \parallel \widehat{C})$. Given (C), by

the inductive hypothesis we have (N) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \hat{n}_2) \parallel \widehat{C}_2)$ and ψ_2 such that $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \hat{n}_2) \parallel \widehat{C}_2)$ and (O) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$. By Definition 5.2.22 we have (P) $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$, (Q) $C_2 \cong_{\psi_2} \widehat{C}_2$, and $n_2 \cong_{\psi_2} \widehat{n}_2$. Given (A), we have $(n_2) \nvdash \gamma$ and therefore by Definition 5.2.19 we have (R) $n_2 = \widehat{n}_2$.

Given (D), (M), and (R), we have (S) $(\widehat{n}_1 < \widehat{n}_2) = 1$.

Given $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}_1 < \widehat{e}_2) \parallel \widehat{C})$, (I), (N), and (S), by Vanilla C rule Less Than True we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma},\ \Box, \Box,\ \widehat{e}_1 < \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{ltt}])} ((\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma}_2,\ \Box, \Box,\ 1) \parallel \widehat{C}_2)$.

Given (P), (Q), and $1 = 1$, by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, 1) \parallel \widehat{C}_2)$. By Definition 5.2.23 we have $ltt \cong \hat{ltt}$. Given (J), (O), $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ltt])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{ltt}])$ by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ltt]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{ltt}])$. Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ltf])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 == e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [eqt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 == e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [eqf])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 ! = e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [net])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 ! = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[nef])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x(\overline{p})\{s\}) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[fd])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x(\overline{p})\{s\}) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[fd])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ by SMC^2 rule Function Definition, we have $\mathrm{acc} = 0$, (B) $x \notin \gamma$, (C) $l = \phi()$, (D) $\mathrm{GetFunTypeList}(\overline{p}) = \overline{ty}$, (E) $\gamma_1 = \gamma[x \rightarrow (l,\ \overline{ty} \rightarrow ty)]$, (F) $\mathrm{CheckPublicEffects}(s,\ x,\ \gamma, \sigma) = n$, (G) $\mathrm{EncodeFun}(s,\ n,\ \overline{p}) = \omega$, and (H) $\sigma_1 = \sigma[l \rightarrow (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \mathrm{PermL_Fun(public)})]$.

Given (I) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x(\overline{p})\{s\}) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}) \parallel \widehat{C})$, by Definition 5.2.22 we have (J) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (K) $C \cong_\psi \widehat{C}$ and $ty\ x(\overline{p})\{s\} \cong_\psi \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\}$. By Definition 5.2.20 we have (L) $ty \cong_\psi \widehat{ty}$, $x \cong_\psi \widehat{x}$ and therefore (M) $x = \widehat{x}$, (N) $\overline{p} \cong_\psi \widehat{\overline{p}}$, and (O) $s \cong_\psi \widehat{s}$.

Given (B), (M), and (J), by Lemma 5.2.59 we have (P) $\widehat{x} \notin \widehat{\gamma}$.

Given (C) by Axiom 5.2.2 we have (Q) $\widehat{l} = \phi()$ such that (R) $l = \widehat{l}$.

Given (D) and (N), by Lemma 5.2.2 we have (S) $\mathrm{GetFunTypeList}(\widehat{\overline{p}}) = \widehat{\overline{ty}}$ such that (T) $\overline{ty} \cong_\psi \widehat{\overline{ty}}$. Given (L) and (T), by Definition 5.2.7 we have (U) $\overline{ty} \rightarrow ty \cong_\psi \widehat{\overline{ty}} \rightarrow \widehat{ty}$.

Given (E), (J), (M), (R), and (U), by Lemma 5.2.60 we have (V) $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \widehat{\overline{ty}} \rightarrow \widehat{ty})]$ such that (W) $(\gamma_1,\ \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given (G), (N), and (O), by Lemma 5.2.7 we have (X) $\mathrm{EncodeFun}(\widehat{s}, \square, \widehat{\overline{p}}) = \widehat{\omega}$ such that (Y) $\omega \cong_\psi \widehat{\omega}$.

Given (H), (W), (R), (Y), and (U), by Lemma 5.2.61 we have (Z) $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{\overline{ty}} \rightarrow \widehat{ty}, 1, \mathrm{PermL_Fun(public)})]$ such that (A1) $(\gamma_1,\ \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given (I), (P), (Q), (S), (V), (X), and (Z), by Vanilla C rule Function Definition we have $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{ty}\ \widehat{x}(\widehat{\overline{p}})\{\widehat{s}\})$ $\parallel \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{fd}])} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \Box, \mathrm{skip}) \parallel \widehat{C})$.

Given (A1) and (K), by Definition 5.2.22 we have $((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \Box, \mathrm{skip}) \parallel C)$. By Definition 5.2.23 we have $fd \cong \hat{fd}$. Given $(\mathrm{p}, [fd])$ and $(\mathrm{p}, [\widehat{fd}])$, by Definition 5.2.25 we have $(\mathrm{p}, [fd]) \cong (\mathrm{p}, [\widehat{fd}])$. Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x(\overline{p})) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[df])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x(\overline{p})\{s\}) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[fd])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$. The main difference is that we are creating the function data as a NULL placeholder, to be defined later.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x(\overline{p})\{s\}) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[fpd])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x(\overline{p})\{s\}) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[fd])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$. The main difference is that we taking out the NULL placeholder data and replacing it with the function data.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x(\overline{e})) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])::\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[fc])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x(\overline{e})) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])::\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[fc])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$ by SMC^2 rule Function Call With Public Side Effects, we have (B) $\gamma(x) = (l,\ \overline{ty} \rightarrow ty)$, (C) $\sigma(l) = (\omega,\ \overline{ty} \rightarrow ty,\ 1,$ PermL_Fun(public)), (D) $\mathrm{DecodeFun}(\omega) = (s,\ n,\ \overline{p})$, (E) $\mathrm{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$, (F) $\mathrm{acc} = 0$, (G) $((\mathrm{p},\ \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, s_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma_1, \sigma_1, \Delta_1, \mathrm{acc}, \mathrm{skip}) \parallel C_1)$, (H) $n = 1$, and (I) $((\mathrm{p}, \gamma_1, \sigma_1, \Delta_1, \mathrm{acc}, s) \parallel C_1)$ $\Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma_2, \sigma_2, \Delta_2, \mathrm{acc}, \mathrm{skip}) \parallel C_2)$.

Given (J) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{x}(\widehat{\overline{e}})) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x(\overline{e})) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{x}(\widehat{\overline{e}})) \parallel \widehat{C})$, by Definition 5.2.22 we have (K) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (L) $C \cong_\psi \widehat{C}$, and (M) $x(\overline{e}) \cong_\psi \widehat{x}(\widehat{\overline{e}})$. By Definition 5.2.20 we have (N) $\overline{e} \cong_\psi \widehat{\overline{e}}$ and $x \cong_\psi \widehat{x}$. Therefore we have (O) $x = \widehat{x}$.

Given (B), (K), and (O), by Lemma 5.2.62 we have (P) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{\overline{ty}} \rightarrow \widehat{ty})$ such that (Q) $\overline{ty} \rightarrow ty \cong_{\psi} \widehat{\overline{ty}} \rightarrow \widehat{ty}$ and (R) $l = \widehat{l}$.

Given (C), (K), and (R), by Lemma 5.2.63 we have (S) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{\overline{ty}} \rightarrow \widehat{ty}, 1, \text{PermL_Fun}(\text{public}))$ such that (T) $\omega \cong_{\psi} \widehat{\omega}$.

Given (D) and (T), by Lemma 5.2.13 we have (U) $\text{DecodeFun}(\widehat{\omega}) = (\widehat{s}, \Box, \widehat{\overline{p}})$ such that (V) $s \cong_{\psi} \widehat{s}$ and (W) $\overline{p} \cong_{\psi} \widehat{\overline{p}}$.

Given (E), (W), and (N), by Lemma 5.2.3 we have (X) $\text{GetFunParamAssign}(\widehat{\overline{p}}, \widehat{\overline{e}}) = \widehat{s}_1$ such that (Y) $s_1 \cong_{\psi} \widehat{s}_1$.

Given (G), (K), (L), and (Y), by Lemma 5.2.50 we have (Z) $((\text{p},\ \gamma,\ \sigma,\ \Delta,\ \text{acc}, s_1) \parallel C) \cong_{\psi} ((\text{p}, \widehat{\gamma},\ \widehat{\sigma},\ \Box, \Box, \widehat{s}_1) \parallel \widehat{C})$. Given (Z), by the inductive hypothesis we have (A1) $((\text{p}, \widehat{\gamma},\ \widehat{\sigma},\ \Box, \Box, \widehat{s}_1) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\text{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \Box, \text{skip}) \parallel \widehat{C}_1)$ and ψ_1 such that (B1) $((\text{p}, \gamma_1, \sigma_1, \Delta_1, \text{acc}, \text{skip}) \parallel C_1) \cong_{\psi_1} ((\text{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \Box, \text{skip}) \parallel \widehat{C}_1)$ and (C1) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (B1), by Definition 5.2.22 we have (D1) $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1, \widehat{\sigma}_1)$ and (E1) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given Axiom 5.2.1, we have $(l, \mu) \notin s$. Therefore, given (V), by Lemma 5.2.55 we have (F1) $s \cong_{\psi_2} \widehat{s}$.

Given (I), (D1), (E1), and (F1), by Lemma 5.2.50 we have (G1) $((\text{p}, \gamma_1, \sigma_1, \Delta_1, \text{acc}, s) \parallel C_1) \cong_{\psi_1} ((\text{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \Box, \widehat{s}) \parallel \widehat{C}_1)$. Given (G1), by the inductive hypothesis we have (H1) $((\text{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \Box, \widehat{s}) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\text{p}, \widehat{\gamma}_2, \widehat{\sigma}_2, \Box, \Box, \text{skip}) \parallel \widehat{C}_2)$ and ψ_2 such that (I1) $((\text{p}, \gamma_2, \sigma_2, \Delta_2, \text{acc}, \text{skip}) \parallel C_2) \cong_{\psi_2} ((\text{p}, \widehat{\gamma}_2, \widehat{\sigma}_2, \Box, \Box, \text{skip}) \parallel \widehat{C}_2)$ and (J1) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$. Given (I1), by Definition 5.2.22 we have (K1) $(\gamma_2, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_2, \widehat{\sigma}_2)$ and (L1) $C_2 \cong_{\psi_2} \widehat{C}_2$.

Given (J1), by Lemma 5.2.57, we have (M1) $(\gamma,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given (J), (P), (S), (U), (X), (A1), and (H1), by Vanilla C rule Function Call we have $((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{x}(\widehat{\overline{e}})) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\text{p}, [\widehat{fc}])} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \text{skip}) \parallel \widehat{C}_2)$.

Given (M1) and (L1), by Definition 5.2.22 we have $((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{skip}) \parallel C_2) \cong_{\psi_2} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \text{skip})$- $\parallel \widehat{C}_2)$.

By Definition 5.2.23 we have $fc \cong \hat{fc}$. Given (E1) and (J1), $\mathcal{D}_1 :: \mathcal{D}_2 :: (\text{p}, [fc])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\text{p}, [\widehat{fc}])$ by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\text{p}, [fc]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\text{p}, [\widehat{fc}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi\triangleright$ $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x(\overline{e})) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])::\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[fc1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi\triangleright$ $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x(\overline{e})) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])::\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[fc])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Case $\Pi\triangleright$ $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[w])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi\triangleright$ $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[w])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Write Public Variable, we have (B) $(e) \nvdash \gamma$, (C) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1)$, (D) $\gamma(x) = (l,\ \mathrm{public}\ bty)$, and (E) $\mathrm{UpdateVal}(\sigma_1,\ l,\ n,\ \mathrm{public}\ bty) = \sigma_2$.

Given (F) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{x} = \widehat{e}) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{x} = \widehat{e}) \parallel \widehat{C})$, by Definition 5.2.22 we have (G) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (H) $C \cong_\psi \widehat{C}$, and (I) $x = e \cong_\psi \widehat{x} = \widehat{e}$. Given (I), by Definition 5.2.20 we have (J) $e \cong_\psi \widehat{e}$ and $x \cong_\psi \widehat{x}$. Therefore we have (K) $x = \widehat{x}$.

Given (C), (G), (H), by Lemma 5.2.50 we have (L) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}) \parallel \widehat{C})$ Given (L), by the inductive hypothesis we have (M) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}) \parallel \widehat{C}_1)$ and ψ_1 such that (N) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}) \parallel \widehat{C}_1)$ and (O) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (N), by Definition 5.2.22 we have (P) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$, (Q) $n \cong_{\psi_1} \widehat{n}$ and (R) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (B), (C) and (Q), by Definition 5.2.19 we have (S) $n = \widehat{n}$.

Given (D), (P), and (K), by Lemma 5.2.62 we have (T) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that (U) $\mathrm{public}\ bty \cong_{\psi_1} \widehat{bty}$ and (V) $l = \widehat{l}$.

Given (E), (P), (V), (Q), and (U), by Lemma 5.2.14 we have (W) $\mathrm{UpdateVal}(\widehat{\sigma}_1, \widehat{l}, \widehat{n}, \widehat{bty}) = \widehat{\sigma}_2$ such that (X) $(\gamma, \sigma_2) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given (F), (M), (T), and (W), by Vanilla C rule Write we have $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{x} = \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::(\mathrm{p},[\widehat{w}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \mathrm{skip}) \parallel \widehat{C}_1)$.

Given (X) and (R), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \mathrm{skip}) \parallel \widehat{C}_1)$. By Definition 5.2.23 we have $w \cong \hat{w}$.

Given (O), $\mathcal{D}_1 :: (\mathrm{p}, [w])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{w}])$ by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [w]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{w}])$. Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [w1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [w])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [w2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [w2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Write Private Variable Public Value, we have (B) $(e) \nvdash \gamma$, (C) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1)$, (D) $\gamma(x) = (l,\ \mathrm{private}\ bty)$, and (E) $\mathrm{UpdateVal}(\sigma_1,\ l,\ \mathrm{encrypt}(n),\ \mathrm{private}\ bty) = \sigma_2$.

Given (F) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x} = \widehat{e}) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x} = \widehat{e}) \parallel \widehat{C})$, by Definition 5.2.22 we have (G) $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, (H) $C \cong_{\psi} \widehat{C}$, and (I) $x = e \cong_{\psi} \widehat{x} = \widehat{e}$. Given (I), by Definition 5.2.20 we have (J) $e \cong_{\psi} \widehat{e}$ and $x \cong_{\psi} \widehat{x}$. Therefore we have (K) $x = \widehat{x}$.

Given (C), (G), and (H), by Lemma 5.2.50 we have (L) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C})$ Given (C) and (L), by the inductive hypothesis we have (M) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$ and ψ_1 such that (N) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$ and (O) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (N), by Definition 5.2.22 we have (P) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$, (Q) $n \cong_{\psi_1} \widehat{n}$ and (R) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (B), (C) and (Q), by Definition 5.2.19 we have $n = \widehat{n}$ and therefore (S) $\mathrm{encrypt}(n) \cong_{\psi_1} \widehat{n}$.

Given (D), (P), and (K), by Lemma 5.2.62 we have (T) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that (U) $\mathrm{private}\ bty \cong_{\psi_1} \widehat{bty}$ and (V) $l = \widehat{l}$.

Given (E), (P), (V), (S), and (U), by Lemma 5.2.14 we have (W) $\mathrm{UpdateVal}(\widehat{\sigma}_1, \widehat{l}, \widehat{n}, \widehat{bty}) = \widehat{\sigma}_2$ such that (X) $(\gamma, \sigma_2) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given (F), (M), (T), and (W), by Vanilla C rule Write we have $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x} = \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{w}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square,$

$\square$, skip) $\parallel \widehat{C}_1$).

Given (X) and (R), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_1)$.
By Definition 5.2.23 we have $w2 \cong \hat{w}$.
Given (O), $\mathcal{D}_1 :: (\mathrm{p}, [w2])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{w}])$ by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [w2]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{w}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[r1])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[r1])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$ by SMC^2 rule Read Private Variable, we have (B) $\gamma(x) = (l,\ \mathrm{private}\ bty)$, (C) $\sigma(l) = (\omega,\ \mathrm{private}\ bty,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty, \mathrm{private}, 1))$, and (D) $\mathrm{DecodeVal}(\mathrm{private}\ bty,\ \omega) = n$.

Given (E) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C})$ by Definition 5.2.22 we have (F) $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, (G) $C \cong_{\psi} \widehat{C}$, and (H) $x \cong_{\psi} \widehat{x}$. Given (H), by Definition 5.2.20 we have (I) $x = \widehat{x}$.

Given (B), (F), and (I), by Lemma 5.2.62 we have (J) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that (K) $\mathrm{private}\ bty \cong_{\psi_1} \widehat{bty}$ and (L) $l = \widehat{l}$.

Given (C), (F), and (L), by Lemma 5.2.63 we have (M) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, 1))$ such that (N) $\omega \cong_{\psi} \widehat{\omega}$.

Given (D), (K), and (N), by Lemma 5.2.8 we have (O) $\mathrm{DecodeVal}(\widehat{bty}, \widehat{\omega}) = \widehat{n}$ such that (P) $n \cong_{\psi} \widehat{n}$.

Given (E), (J), (M), and (O), by Vanilla C rule Read we have $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{r}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}) \parallel \widehat{C})$.

Given (F), (G), and (P), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}) \parallel \widehat{C})$.
By Definition 5.2.23 we have $r1 \cong \hat{r}$, and by Definition 5.2.25 we have $(\mathrm{p}, [r1]) \cong (\mathrm{p}, [\widehat{r}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[r])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$

This case is similar to Case $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[r1])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$.

Case $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[dv])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

Given (A) $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[dv])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ by SMC^2 rule Public Declaration, we have (B) $(ty = \mathrm{public}\ bty)$, $\mathrm{acc} = 0$, (C) $l = \phi()$, (D) $\gamma_1 = \gamma[x\ \rightarrow\ (l,\ ty)]$, (E) $\omega = \mathrm{EncodeVal}(ty, \mathrm{NULL})$, and (F) $\sigma_1 = \sigma[l\ \rightarrow\ (\omega,\ ty,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, ty, \mathrm{public}, 1))]$.

Given (G) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{bty}\ \widehat{x}) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{bty}\ \widehat{x}) \parallel \widehat{C})$, by Definition 5.2.22 we have (H) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$ and (I) $ty\ x \cong_\psi \widehat{bty}\ \widehat{x}$. Given (B) and (I), by Definition 5.2.20 we have (J) $\mathrm{public}\ bty \cong_\psi \widehat{bty}$ such that (K) $bty = \widehat{bty}$ and $x \cong_\psi \widehat{x}$ such that (L) $x = \widehat{x}$.

Given (C), by Axiom 5.2.2 we have (M) $\widehat{l} = \phi()$ and (N) $l = \widehat{l}$.

Given (D), (H), (L), (N), and (I), by Lemma 5.2.60 we have (O) $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \widehat{bty})]$ such that (P) $(\gamma_1, \sigma) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma})$.

Given (E) and (I), by Lemma 5.2.5 we have (Q) $\widehat{\omega} = \mathrm{EncodeVal}(\widehat{bty}, \mathrm{NULL})$ such that (R) $\omega \cong_\psi \widehat{\omega}$.

Given (F), (N), (R), (I), and (P), by Lemma 5.2.61 we have (S) $\widehat{\sigma}_1 = \widehat{\sigma}[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{bty}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, 1))]$ such that (T) $(\gamma_1, \sigma_1) \cong_\psi (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$.

Given (G), (M), (O), (Q), and (S), by Vanilla C rule Declaration we have $\Sigma\triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{bty}\ \widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{dv}])}$ $((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \Box, \mathrm{skip}) \parallel \widehat{C})$.

Given (T), by Definition 5.2.22 we have $((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \Box, \mathrm{skip}) \parallel \widehat{C})$.
By Definition 5.2.23 we have $dv \cong \hat{dv}$, and by Definition 5.2.25 we have $(\mathrm{p}, [dv]) \cong (\mathrm{p}, [\widehat{dv}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[d1])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

This case is similar to Case $\Pi\triangleright$ $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[dv])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$.

Case $\Pi\triangleright$ $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ s_1;\ s_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ss])} ((\mathrm{p}, \gamma_2,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ v) \parallel C_2)$

Given (A) $\Pi\triangleright$ $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ s_1;\ s_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ss])} ((\mathrm{p}, \gamma_2,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ v) \parallel C_2)$ by SMC^2 rule Statement Sequencing, we have (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ s_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v_1) \parallel C_1)$ and (C) $((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ s_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma_2,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ v_2) \parallel C_2)$.

Given (D) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}_1; \widehat{s}_2) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ s_1;\ s_2) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}_1; \widehat{s}_2)$-$\parallel \widehat{C})$, by Definition 5.2.22 we have (E) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (F) $C \cong_\psi \widehat{C}$ and (G) $s_1;\ s_2 \cong_\psi \widehat{s}_1; \widehat{s}_2$. By Definition 5.2.12 we have (H) $s_1 \cong_\psi \widehat{s}_1$ and (I) $s_2 \cong_\psi \widehat{s}_2$.

Given ψ, (E), (F), and (H), by Lemma 5.2.50 we have (J) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ s_1) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}_1) \parallel \widehat{C})$. Given (B) and (J), by the inductive hypothesis we have (K) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}_1) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \widehat{v}_1) \parallel \widehat{C}_1)$ and ψ_1 such that (L) $((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v_1) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \widehat{v}_1) \parallel \widehat{C}_1)$ and (M) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$.

Given (L), by Definition 5.2.22 we have (N) $(\gamma_1,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, (O) $v_1 \cong_{\psi_1} \widehat{v}_1$, and (P) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given Axiom 5.2.1, we have $(l, \mu) \notin s_2$. Therefore, given (I), by Lemma 5.2.55 we have (Q) $s_2 \cong_{\psi_1} \widehat{s}_2$.

Given ψ_1, (N), (P), and (Q), by Lemma 5.2.50 we have (R) $((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ s_2) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \widehat{s}_2)$-$\parallel \widehat{C}_1)$. Given (C) and (R), by the inductive hypothesis we have (S) $((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \widehat{s}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}_2, \widehat{\sigma}_2, \square, \square, \widehat{v}_2)$ $\parallel \widehat{C}_2)$ and ψ_2 such that (T) $((\mathrm{p}, \gamma_2,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ v_2)\ \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}_2, \widehat{\sigma}_2, \square, \square, \widehat{v}_2)\ \parallel \widehat{C}_2)$ and (U) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$.

Given (T), by Definition 5.2.22 we have (V) $(\gamma_2,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_2,\ \widehat{\sigma}_2)$, (W) $v_2 \cong_{\psi_2} \widehat{v}_2$, and (X) $C_2 \cong_{\psi_2} \widehat{C}_2$.

Given (D), (K), and (S), by Vanilla C rule Statement Sequencing we have $\Sigma\triangleright$ $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{s}_1; \widehat{s}_2) \parallel \widehat{C})$ $\Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{ss}])} ((\mathrm{p}, \widehat{\gamma}_2, \widehat{\sigma}_2, \square, \square, \widehat{v}_2) \parallel \widehat{C}_2)$.

Given (V), (W), and (X), by Definition 5.2.22 we have $((\mathrm{p}, \gamma_2,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ v_2) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}_2, \widehat{\sigma}_2, \square, \square, \widehat{v}_2)$-$\parallel \widehat{C}_2)$.

By Definition 5.2.23 we have $ss \cong \widehat{ss}$. Given (M), (U), $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ss])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{ss}])$, by Lemma 5.2.58

we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ss]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{s}\widehat{s}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_2} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \{s\}) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\mathrm{p},[sb])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \{s\}) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\mathrm{p},[sb])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Statement Block, we have (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ s) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v) \parallel C_1)$.

Given (C) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{s}_1; \widehat{s}_2) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \{s\}) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \{\widehat{s}\}) \parallel \widehat{C})$, by Definition 5.2.22 we have (D) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (E) $C \cong_\psi \widehat{C}$ and (F) $\{s\} \cong_\psi \{\widehat{s}\}$. Given (F), by Definition 5.2.20 we have (G) $s \cong_\psi \widehat{s}$.

Given ψ, (D), (E), and (G), by Lemma 5.2.50 we have (H) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ s) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{s}) \parallel \widehat{C})$. Given (B) and (H), by the inductive hypothesis we have (I) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{s}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \Box, \widehat{v}) \parallel \widehat{C}_1)$ and ψ_1 such that (J) $((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \Box, \Box, \widehat{v}) \parallel \widehat{C}_1)$ and (K) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$.

Given (J), by Definition 5.2.22 we have (L) $(\gamma_1,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, (M) $v \cong_{\psi_1} \widehat{v}$, and (N) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (B), (I), and (J), by Lemma 5.2.57 we have (O) $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$

Given (C) and (I), by Vanilla C rule Statement Block we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \{\widehat{s}\}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::(\mathrm{p},[\widehat{sb}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \mathrm{skip}) \parallel \widehat{C}_1)$.

Given (O) and (N), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \mathrm{skip}) \parallel \widehat{C}_1)$. By Definition 5.2.23 we have $sb \cong \hat{sb}$. Given (K), $\mathcal{D}_1 :: (\mathrm{p}, [sb])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{sb}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [sb]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{sb}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (e)) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\mathrm{p},[ep])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (e)) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\mathrm{p},[ep])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v) \parallel C_1)$ by SMC^2 rule Parentheses, we

have (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v) \parallel C_1)$.

Given (C) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, (\widehat{e})) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (e)) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, (\widehat{e})) \parallel \widehat{C})$, by Definition 5.2.22 we have (D) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (E) $C \cong_\psi \widehat{C}$ and (F) $(e) \cong_\psi (\widehat{e})$. Given (F), by Definition 5.2.20 we have (G) $e \cong \widehat{e}$.

Given ψ, (D), (E), and (G), by Lemma 5.2.50 we have (H) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}) \parallel \widehat{C})$. Given (B) and (H), by the inductive hypothesis we have (I) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{v}) \parallel \widehat{C}_1)$ and ψ_1 such that (J) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{v}) \parallel \widehat{C}_1)$ and (K) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$.

Given (J), by Definition 5.2.22 we have (L) $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$, (M) $v \cong_{\psi_1} \widehat{v}$, and (N) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (C) and (I), by Vanilla C rule Parentheses we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, (\widehat{e})) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{ep}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{v}) \parallel \widehat{C}_1)$.

Given (L), (M), and (N), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{v}) \parallel \widehat{C}_1)$. By Definition 5.2.23 we have $ep \cong \widehat{ep}$. Given (L), $\mathcal{D}_1 :: (\mathrm{p}, [ep])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{ep}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [ep]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{ep}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ds])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ds])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$ by SMC^2 rule Declaration Assignment, we have (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ and (C) $((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ x = e) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma_1,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Given (D) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{ty}\ \widehat{x} = \widehat{e}) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x = e) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{ty}\ \widehat{x} = \widehat{e}) \parallel \widehat{C})$, by Definition 5.2.22 we have (E) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (F) $C \cong_\psi \widehat{C}$ and (G) $ty\ x = e \cong_\psi \widehat{ty}\ \widehat{x} = \widehat{e}$. By Definition 5.2.20 we have (H) $ty \cong_\psi \widehat{ty}$, (I) $x \cong_\psi \widehat{x}$, such that (J) $x = \widehat{x}$, and (K) $e \cong_\psi \widehat{e}$.

Given (H) and (J), by Definition 5.2.20 we have (L) $ty\ x \cong_\psi \widehat{ty}\ \widehat{x}$.

Given ψ, (E), (F), and (L), by Lemma 5.2.50 we have (M) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ x) \parallel \widehat{C})$. Given (B) and (M), by the inductive hypothesis we have (N) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ x) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \parallel \widehat{C}_1)$ and ψ_1 such that (O) $((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \parallel \widehat{C}_1)$ and (P) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$.

Given (O), by Definition 5.2.22 we have (Q) $(\gamma_1,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1,\ \widehat{\sigma}_1)$, and (R) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given Axiom 5.2.1, we have $(l, \mu) \notin e$. Therefore, given (K), by Lemma 5.2.55 we have (S) $e \cong_{\psi_1} \widehat{e}$.

Given (J) and (S), by Definition 5.2.20 we have (T) $x = e \cong_{\psi_1} \widehat{x} = \widehat{e}$.

Given ψ_1, (Q), (T), and (R), by Lemma 5.2.50 we have (U) $((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ x = e) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \widehat{x} = \widehat{e}) \parallel \widehat{C}_1)$. Given (C) and (U), by the inductive hypothesis we have (V) $((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_1, \square, \square, \widehat{x} = \widehat{e}) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$ and ψ_2 such that (W) $((\mathrm{p}, \gamma_1,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$ and (X) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$.

Given (W), by Definition 5.2.22 we have (Y) $(\gamma_1,\ \sigma_2) \cong_{\psi_2} (\widehat{\gamma}_1,\ \widehat{\sigma}_2)$ and (Z) $C_2 \cong_{\psi_2} \widehat{C}_2$.

Given (D), (N), and (V), by Vanilla C rule Declaration Assignment we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{ty}\ \widehat{x} = \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{ds}])} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$.

Given (Y) and (Z), by Definition 5.2.22 we have $((\mathrm{p}, \gamma_1,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$. By Definition 5.2.23 we have $ds \cong \hat{ds}$. Given (P), (X), $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ds])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{ds}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ds]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{ds}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_2} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x[e_1] = e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [das])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ds])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_1,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [da1])} ((\mathrm{p}, \gamma_1,\ \sigma_3,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,0)])}_{\mathcal{D}_1::(\mathrm{p},[da1])} ((\mathrm{p}, \gamma_1,\ \sigma_3,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Private Array Declaration, we have (B) $(e) \nvdash \gamma$, (C) $((ty = \text{private } bty) \vee (ty = bty)) \wedge ((bty = \mathrm{int}) \vee (bty = \mathrm{float}))$, (D) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \alpha) \parallel C_1)$, (E) $\alpha > 0$, (F) $l = \phi()$, (G) $l_1 = \phi()$, (H) $\gamma_1 = \gamma[x \rightarrow (l,\ \text{private const } bty*)]$, (I) $\omega = \mathrm{EncodePtr}(\text{private const } bty*,\ [1,\ [(l_1, 0)],\ [1],\ 1])$, (J) $\omega_1 = \mathrm{EncodeArr}(\text{private } bty, 0, \alpha, \mathrm{NULL})$, (K) $\sigma_2 = \sigma_1[l \rightarrow (\omega,\ \text{private const } bty*,\ 1,\ \mathrm{PermL}(\text{Freeable, private const } bty*, \text{private}, 1))]$, and (L) $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1,\ \text{private } bty,\ \alpha, \mathrm{PermL}(\text{Freeable, private } bty, \text{private}, \alpha))]$.

Given (M) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{bty}\ \widehat{x}[\widehat{e}]) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x[e]) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{bty}\ \widehat{x}[\widehat{e}]) \parallel \widehat{C})$, by Definition 5.2.22 we have (N) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (O) $C \cong_\psi \widehat{C}$ and (P) $ty\ x[e] \cong_\psi \widehat{bty}\ \widehat{x}[\widehat{e}]$.

Given (P), by Definition 5.2.20 we have (Q) $ty \cong_\psi \widehat{bty}$, (R) $x \cong_\psi \widehat{x}$ such that (S) $x = \widehat{x}$ and (T) $e \cong_\psi \widehat{e}$. Given (C) and (Q), by Definition 5.2.8 we have (U) $bty = \widehat{bty}$.

Given ψ, (N), (O), and (T), by Lemma 5.2.50 we have (V) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C})$. Given (D) and (V), by the inductive hypothesis we have (W) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{\alpha}) \parallel \widehat{C}_1)$ and ψ_1 such that (X) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \alpha) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{\alpha}) \parallel \widehat{C}_1)$ and (Y) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$.

Given (X), by Definition 5.2.22 we have (Z) $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$, (A1) $\alpha \cong_{\psi_1} \widehat{\alpha}$, and (B1) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (F) and (G), by Axiom 5.2.2 we have (C1) $\widehat{l} = \phi()$, (D1) $l = \widehat{l}$, (E1) $\widehat{l}_1 = \phi()$, and (F1) $l_1 = \widehat{l}_1$.

Given (C), (Q), and (U), by Definition 5.2.8 we have (G1) $\text{private const } bty* \cong_\psi \text{const } \widehat{bty}*$ Given (H), (Z), (D1), and (G1), by Lemma 5.2.60 we have (H1) $\widehat{\gamma}_1 = \widehat{\gamma}[\widehat{x} \rightarrow (\widehat{l}, \text{const } \widehat{bty}*)]$ such that (I1) $(\gamma_1, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}_1, \widehat{\sigma}_1)$.

Given (F1), by Definition 5.2.15 we have (J1) $[1,\ [(l_1, 0)],\ [1],\ 1] \cong_{\psi_1} [1, [(\widehat{l}_1, 0)], [1], 1]$. Given (I), (G1), and (J1), by Lemma 5.2.4 we have (K1) $\widehat{\omega} = \mathrm{EncodePtr}(\text{const } \widehat{bty}*, [1, [(\widehat{l}_1, 0)], [1], 1])$ such that (L1) $\omega \cong_{\psi_1} \widehat{\omega}$.

Given (C), (Q), and (U), by Definition 5.2.8 we have (M1) $\text{private } bty \cong_\psi \widehat{bty}$ Given (J), (M1), and (A1), by Lemma 5.2.6 we have (N1) $\widehat{\omega}_1 = \mathrm{EncodeArr}(\widehat{bty}, 0, \widehat{\alpha}, \mathrm{NULL})$ such that (O1) $\omega_1 \cong_{\psi_1} \widehat{\omega}_1$.

Given (A1) and (B), by Lemma 5.2.51 we have (P1) $\alpha = \widehat{\alpha}$. Given (E) and (P1), we have (Q1) $\widehat{\alpha} > 0$.

Given (K), (I1), (C1), (K1), and (G1), by Lemma 5.2.61 we have (R1) $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\widehat{\omega}, \text{const } \widehat{bty}*, 1,$

$\text{PermL}(\text{Freeable, const } \widehat{bty}*, \text{public}, 1))]$ such that (S1) $(\gamma_1, \sigma_2) \cong_{\psi_1} (\widehat{\gamma}_1, \widehat{\sigma}_2)$.

Given (L), (S1), (F1), (O1), (P1), and (M1), by Lemma 5.2.61 we have (T1) $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l_1} \rightarrow (\widehat{\omega}_1, \widehat{bty}, \alpha, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \alpha))]$ such that (U1) $(\gamma_1, \sigma_3) \cong_{\psi_1} (\widehat{\gamma}_1, \widehat{\sigma}_3)$.

Given (M), (W), (C1), (E1), (H1), (K1), (N1), (Q1), (R1), and (T1), by Vanilla C rule Array Declaration we have $\Sigma \triangleright ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{bty}\ \widehat{x}[\widehat{e}]) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{da}])} ((\text{p}, \widehat{\gamma}_1, \widehat{\sigma}_3, \square, \square, \text{skip}) \parallel \widehat{C}_1)$.

Given (U1) and (B1), by Definition 5.2.22 we have $((\text{p}, \gamma_1,\ \sigma_3,\ \Delta,\ \text{acc},\ \text{skip}) \parallel C_1) \cong_{\psi} ((\text{p}, \widehat{\gamma}_1, \widehat{\sigma}_3, \square, \square, \text{skip}) \parallel \widehat{C}_1)$. By Definition 5.2.23 we have $da1 \cong \hat{da}$. Given (Y), $\mathcal{D}_1 :: (\text{p}, [da1])$ and $\widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{da}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\text{p}, [da1]) \cong \widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{da}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0),(l_1,0)])}_{\mathcal{D}_1 :: (\text{p}, [da])} ((\text{p}, \gamma_1,\ \sigma_3,\ \Delta,\ \text{acc},\ \text{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0),(l_1,0)])}_{\mathcal{D}_1 :: (\text{p}, [da1])} ((\text{p}, \gamma_1,\ \sigma_3,\ \Delta,\ \text{acc},\ \text{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0),(l_1,i)])}_{\mathcal{D}_1 :: (\text{p}, [ra])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ n_i) \parallel C_1)$

Given (A) $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0),(l_1,i)])}_{\mathcal{D}_1 :: (\text{p}, [ra])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ n_i) \parallel C_1)$ by SMC^2 rule Public Array Read Public Index, we have (B) $(e) \nvdash \gamma$, (C) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ i) \parallel C_1)$, (D) $\gamma(x) = (l,\ \text{public const } bty*)$, (E) $\sigma_1(l) = (\omega,\ \text{public const } bty*, 1,\ \text{PermL}(\text{Freeable}, \text{public const } bty*, \text{public},\ 1))$, (F) $\text{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$, (G) $\sigma_1(l_1) = (\omega_1, \text{public } bty,\ \alpha, \text{PermL}(\text{Freeable}, \text{public } bty, \text{public}, \alpha))$, (H) $0 \leq i \leq \alpha - 1$, and (I) $\text{DecodeArr}(\text{public } bty,\ i,\ \omega_1) = n_i$.

Given (J) $((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}]) \parallel \widehat{C})$ and ψ such that $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e]) \parallel C) \cong_{\psi} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}]) \parallel \widehat{C})$, by Definition 5.2.22 we have (K) $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})$, (L) $x[e] \cong_{\psi} \widehat{x}[\widehat{e}]$, and (M) $C \cong_{\psi} \widehat{C}$. Given (L), by Definition 5.2.20 we have (N) $e \cong_{\psi} \widehat{e}$ and $x \cong_{\psi} \widehat{x}$ such that (O) $x = \widehat{x}$.

Given ψ, (K), (N), and (M), by Lemma 5.2.50 we have (P) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C) \cong_{\psi} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C})$. Given (C) and (P), by the inductive hypothesis we have (Q) $((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{i}) \parallel \widehat{C}_1)$

and ψ_1 such that $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ i) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{i}) \parallel \widehat{C}_1)$. By Definition 5.2.22 we have (R) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$ (S) $i \cong_{\psi_1} \widehat{i}$, (T) $C_1 \cong_{\psi_1} \widehat{C}_1$, and (U) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$.

Given (D), (R), and (O), by Lemma 5.2.62 we have (V) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \mathrm{const}\ \widehat{bty}*)$ such that (W) $\mathrm{public\ const}\ bty* \cong_{\psi_1} \mathrm{const}\ \widehat{bty}*$ and (X) $l = \widehat{l}$.

Given (E), (R), and (X), by Lemma 5.2.63 we have (Y) $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable},\ \mathrm{const}\ \widehat{bty}*, \mathrm{public}, 1))$ such that (Z) $\omega \cong_{\psi_1} \widehat{\omega}$.

Given (F), (W), and (Z), by Lemma 5.2.12 we have (A1) $\mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ such that (B1) $l_1 = \widehat{l}_1$.

Given (G), (R), and (B1), by Lemma 5.2.63 we have (C1) $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, \widehat{\alpha}))$ such that (D1) $\omega_1 \cong_{\psi_1} \widehat{\omega}_1$, (E1) $\alpha = \widehat{\alpha}$, and (F1) $\mathrm{public}\ bty \cong_{\psi_1} \widehat{bty}$.

Given (S) and (B), by Lemma 5.2.51 we have (G1) $i = \widehat{i}$. Given (H), (G1), and (E1), we have (H1) $0 \leq \widehat{i} \leq \widehat{\alpha} - 1$.

Given (I), (F1), (G1), and (D1), by Lemma 5.2.9 we have (I1) $\mathrm{DecodeArr}(\widehat{bty},\ \widehat{i},\ \widehat{\omega}_1) = \widehat{n}_{\widehat{i}}$ such that (J1) $n_i \cong_{\psi_1} \widehat{n}_{\widehat{i}}$.

Given (J), (Q), (V), (Y), (A1), (C1), (H1), and (I1), by Vanilla C rule Array Read we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}]) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{ra}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}_{\widehat{i}}) \parallel \widehat{C}_1)$.

Given (R), (J1), and (T), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_i) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}_{\widehat{i}}) \parallel \widehat{C}_1)$. By Definition 5.2.23 we have $ra \cong \widehat{ra}$. Given (U), $\mathcal{D}_1 :: (\mathrm{p}, [ra])$, and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{ra}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [ra]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{ra}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_1,i)])}_{\mathcal{D}_1 :: (\mathrm{p}, [ra1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_i) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_1,i)])}_{\mathcal{D}_1 :: (\mathrm{p}, [ra])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_i) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wa2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wa2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$ by SMC^2 rule Private Array Write Private Value Public Index, we have (B) $(e_1) \nvdash \gamma$, (C) $(e_2) \vdash \gamma$, (D) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e_1)$ - $\parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, i) \parallel C_1)$, (E) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, n) \parallel C_2)$, (F) $\gamma(x) = (l,\ \text{private const}\ bty*)$, (G) $\sigma_2(l)\ =\ (\omega, \text{private const}\ bty*, 1, \mathrm{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private},\ 1))$, (H) $\mathrm{DecodePtr}(\text{private const}\ bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$, (I) $\sigma_2(l_1) = (\omega_1, \text{private}\ bty, \alpha,$ $\mathrm{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, \alpha))$, (J) $0 \leq i \leq \alpha - 1$, (K) $\mathrm{DynamicUpdate}(\Delta_2, \sigma_2, [(l_1, i)],\ \mathrm{acc}, \text{private}$ $bty) = \Delta_3$, and (L) $\mathrm{UpdateArr}(\sigma_2,\ (l_1, i),\ n,\ \text{private}\ bty) = \sigma_3$.

Given (M) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box,$ $\widehat{x}[\widehat{e}_1]\ =\ \widehat{e}_2) \parallel \widehat{C})$, by Definition 5.2.22 we have (N) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (O) $x[e_1]\ = e_2 \cong_\psi \widehat{x}[\widehat{e}_1]\ =\ \widehat{e}_2$, and (P) $C \cong_\psi \widehat{C}$. By Definition 5.2.20 we have (Q) $e_1 \cong_\psi \widehat{e}_1$, (R) $e_2 \cong_\psi \widehat{e}_2$, and $x \cong_\psi \widehat{x}$ such that (S) $x = \widehat{x}$.

Given ψ, (N), (Q), and (P), by Lemma 5.2.50 we have (T) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e_1)\ \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma},\ \Box, \Box, \widehat{e}_1) \parallel \widehat{C})$ Given (D) and (T), by the inductive hypothesis we have (U) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma},\ \Box, \Box, \widehat{e}_1) \parallel \widehat{C})\ \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \hat{i}) \parallel \widehat{C}_1)$ and ψ_1 such that (V) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ i) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \hat{i}) \parallel \widehat{C}_1)$. Given (V), by Definition 5.2.22 we have (W) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$, (X) $i \cong_{\psi_1} \widehat{i}$, (Y) $C_1 \cong_{\psi_1} \widehat{C}_1$, and (Z) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$.

Given (B) and (X), by Lemma 5.2.51 we have (A1) $i = \widehat{i}$.

Given Axiom 5.2.1, we have $(l, \mu) \notin e_2$. Given (R), by Lemma 5.2.55 we have (B1) $e_2 \cong_{\psi_1} \widehat{e}_2$.

Given ψ_1, (W), (B1), and (Y), by Lemma 5.2.50 we have (C1) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, e_2) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{e}_2)$ $\parallel \widehat{C}_1)$. Given (E) and (C1), by the inductive hypothesis we have (D1) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box,$ $\Box, \widehat{n})\ \parallel \widehat{C}_2)$ and ψ_2 such that (E1) $((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, n) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \widehat{n}) \parallel \widehat{C}_2)$. Given (E1), by Definition 5.2.22 we have (F1) $(\gamma, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2)$, (G1) $n \cong_{\psi_2} \widehat{n}$, and (H1) $C_2 \cong_{\psi_2} \widehat{C}_2$, and (I1) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$.

Given (F), (F1), and (S), by Lemma 5.2.62 we have (J1) $\widehat{\gamma}(\widehat{x})\ =\ (\widehat{l}, \text{const}\ \widehat{bty*})$ such that (K1) $l\ =\ \widehat{l}$ and (L1) $\text{private const}\ bty* \cong_{\psi_2} \text{const}\ \widehat{bty*}$. Given (L1), by Definition 5.2.8 we have (M1) $\text{private}\ bty \cong_{\psi_2} \widehat{bty}$.

Given (G), (F1), and (K1), by Lemma 5.2.63 we have (N1) $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \text{const}\ \widehat{bty*}, 1, \mathrm{PermL}(\text{Freeable}, \text{const}\ \widehat{bty*},$ $\text{public}, 1))$ such that (O1) $\omega \cong_{\psi_2} \widehat{\omega}$.

Given (H), (L1), and (O1), by Lemma 5.2.12 we have (P1) $\mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ such that (Q1) $l_1 = \widehat{l}_1$.

Given (I), (Q1), and (F1), by Lemma 5.2.63 we have (R1) $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, \widehat{\alpha}))$ such that (S1) $\omega_1 \cong_{\psi_2} \widehat{\omega}_1$, (T1) $\alpha = \widehat{\alpha}$.

Given (J), (A1), and (T1), we have (U1) $0 \leq \hat{i} \leq \widehat{\alpha} - 1$.

Given (L), (E1), (O1), (Z), (R1), and (K1), by Lemma 5.2.15 we have (V1) $\mathrm{UpdateArr}(\widehat{\sigma}_2,\ (\widehat{l}_1, \widehat{i}),\ \widehat{n},\ \widehat{bty}) = \widehat{\sigma}_3$ such that (W1) $(\gamma, \sigma_3) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given (M), (U), (D1), (J1), (N1), (P1), (R1), (U1), and (V1), by Vanilla C rule Array Write we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{wa}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_3, \Box, \Box, \mathrm{skip}) \parallel \widehat{C}_2)$.

Given (W1) and (H1), by Definition 5.2.22 we have $((\mathrm{p}, \gamma, \sigma_3, \Delta_3, \mathrm{acc}, \mathrm{skip}) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_3, \Box, \Box, \mathrm{skip})$ - $\parallel \widehat{C}_2)$.

By Definition 5.2.23 we have $wa2 \cong \hat{wa}$. Given (Z), (I1), $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wa2])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{wa}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wa2]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{wa}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_2} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2 :: (\mathrm{p}, [(l,0),(l_1,i)])}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wa1])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2 :: (\mathrm{p}, [(l,0),(l_1,i)])}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wa2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc}, \mathrm{skip})$ $\parallel C_2)$. Given $n = \widehat{n}$, we use Definition 5.2.19 to prove that $\mathrm{encrypt}(n) \cong \widehat{n}$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1] = e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2 :: (\mathrm{p}, [(l,0),(l_1,i)])}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wa])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2 :: (\mathrm{p}, [(l,0),(l_1,i)])}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wa2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc}, \mathrm{skip})$ $\parallel C_2)$.

Case $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{(\mathrm{p},[rea])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [n_0, ..., n_{\alpha-1}]) \parallel C)$

Given (A) $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{(\mathrm{p},[rea])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [n_0, ..., n_{\alpha-1}]) \parallel C)$ by SMC^2 rule Read Entire Array, we have (B) $\gamma(x) = (l, a \text{ const } bty*)$, (C) $\sigma(l) = (\omega, a \text{ const } bty*, 1, \mathrm{PermL}(\text{Freeable}, a \text{ const } bty*, a, 1))$, (D) $\mathrm{DecodePtr}(a \text{ const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$, (E) $\sigma(l_1) = (\omega_1, a\ bty, \alpha, \mathrm{PermL}(\text{Freeable}, a\ bty, a, \alpha))$, and (F) $\forall i \in \{0...\alpha-1\}\ \mathrm{DecodeArr}(a\ bty, i, \omega_1) = n_i$.

Given (G) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C})$, by Definition 5.2.22 we have (H) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, (I) $x \cong_\psi \widehat{x}$, and (J) $C \cong_\psi \widehat{C}$. Given (I), by Definition 5.2.20 we have (K) $x = \widehat{x}$.

Given (B), (H), and (K), by Lemma 5.2.62 we have (L) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that (M) $l = \widehat{l}$ and (N) $a \text{ const } bty* \cong_\psi \text{const } \widehat{bty}*$.

Given (C), (H), and (M), by Lemma 5.2.63 we have (O) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \mathrm{PermL}(\text{Freeable}, \text{const } \widehat{bty}*, \text{public}, 1))$ such that (P) $\omega \cong_\psi \widehat{\omega}$.

Given (D), (N), and (P), by Lemma 5.2.12 we have (Q) $\mathrm{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ such that (R) $l_1 = \widehat{l}_1$.

Given (E), (H), (R), by Lemma 5.2.63 we have (T) $\widehat{\sigma}(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\text{Freeable}, bty, \text{public}, \widehat{\alpha}))$ such that (U) $\omega_1 \cong_\psi \widehat{\omega}_1$, (V) $bty \cong_\psi \widehat{bty}$, and (W) $\alpha = \widehat{\alpha}$.

Given (F) and (W), we have (X) $i = \widehat{i}$. Given (F), (X), (W), (V), and (U), by Lemma 5.2.10 we have (Y) $\forall \widehat{i} \in \{0...\widehat{\alpha}-1\}$ $\mathrm{DecodeArr}(\widehat{bty}, \widehat{i}, \widehat{\omega}_1) = \widehat{n}_{\widehat{i}}$ such that (Z) $\forall i \in \{0...\alpha-1\}\ n_i \cong_\psi \widehat{n}_i$.

Given (G), (L), (O), (Q), (T), and (Y), by Vanilla C rule Read Entire Array we have $\Sigma\triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{rea}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, [\widehat{n}_0, ..., \widehat{n}_{\widehat{\alpha}-1}]) \parallel \widehat{C})$.

Given (H), (J), (W), and (Z), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [n_0, ..., n_{\alpha-1}]) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, [\widehat{n}_0, ..., \widehat{n}_{\widehat{\alpha}-1}]) \parallel \widehat{C})$.

By Definition 5.2.23 we have $rea \cong \widehat{rea}$, and by Definition 5.2.25 we have $(\mathrm{p}, [rea]) \cong (\mathrm{p}, [\widehat{rea}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::(\mathrm{p},[wea1])} ((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::(\mathrm{p},[wea1])} ((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Write Entire Private Array, we have (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ [n_0, ..., n_{\alpha_e-1}]) \parallel C_1)$, (C) $\gamma(x) = (l,\ \text{private const } bty*)$, (D) $(e) \vdash \gamma$, (E) $\sigma_1(l) = (\omega, \text{private const } bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, \text{private const } bty*, \mathrm{private}, 1))$, (F) $\mathrm{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)], [1], 1]$, (G) $\sigma_1(l_1) = (\omega_1, \text{private } bty, \alpha, \mathrm{PermL}(\mathrm{Freeable}, \text{private } bty, \mathrm{private}, \alpha))$, (H) $\alpha_e = \alpha$, and (I) $\forall i \in \{0...\alpha - 1\}$ $\mathrm{UpdateArr}(\sigma_{1+i},\ (l_1, i),\ n_i,\ \text{private } bty) = \sigma_{2+i}$.

Given (J) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x} = \widehat{e}) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x} = \widehat{e})$-$\parallel \widehat{C})$, by Definition 5.2.22 we have (K) $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, (L) $C \cong_{\psi} \widehat{C}$, and (M) $x = e \cong_{\psi} \widehat{x} = \widehat{e}$. Given (M), by Definition 5.2.20 we have (N) $e \cong_{\psi} \widehat{e}$ and $x \cong_{\psi} \widehat{x}$ such that (O) $x = \widehat{x}$.

Given ψ, (K), (L), and (N), by Lemma 5.2.50 we have (P) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C})$ Given (B) and (P), by the inductive hypothesis we have (Q) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, [\widehat{n}_0, ..., \widehat{n}_{\widehat{\alpha}_e-1}]) \parallel \widehat{C}_1)$ and ψ_1 such that (R) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, [n_0, ..., n_{\alpha_e-1}]) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, [\widehat{n}_0, ..., \widehat{n}_{\widehat{\alpha}_e-1}]) \parallel \widehat{C}_1)$ and (S) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (R), by Definition 5.2.22 we have (T) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$, (U) $[n_0, ..., n_{\alpha_e-1}] \cong_{\psi_1} [\widehat{n}_0, ..., \widehat{n}_{\widehat{\alpha}_e-1}]$, and (V) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (C), (T), and (O), by Lemma 5.2.62 we have (W) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that (X) $l = \widehat{l}$ and (Y) $\text{private const } bty* \cong_{\psi_1} \text{const } \widehat{bty}*$. By Definition 5.2.8 we have (Z) $bty = \widehat{bty}$.

Given (E), (T), and (X), by Lemma 5.2.63 we have (A1) $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \text{const } \widehat{bty}*, \mathrm{public}, 1))$ such that (B1) $\omega \cong_{\psi_1} \widehat{\omega}$.

Given (F), (Y), and (B1), by Lemma 5.2.12 we have (C1) $\mathrm{DecodePtr}(\text{const } \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ such that (D1) $l_1 = \widehat{l}_1$.

Given (G), (T), and (D1), by Lemma 5.2.63 we have (E1) $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, bty, \mathrm{public}, \widehat{\alpha}))$ such that (F1) $\omega_1 \cong_{\psi_1} \widehat{\omega}_1$ and (G1) $\alpha = \widehat{\alpha}$.

Given (U), by Definition 5.2.20 we have (H1) $\alpha_e = \widehat{\alpha}_e$. Given (H), (H1), and (G1), we have (I1) $\widehat{\alpha}_e = \widehat{\alpha}$.

Given (I) and (G1), we have (J1) $i = \widehat{i} \in \{0...\alpha-1\}$. Given (I), (T), (D1), (U), (Z), (I1), (G1), and (J1), by Lemma 5.2.16 we have (K1) $\forall \widehat{i} \in \{0...\widehat{\alpha}-1\}$ $\mathrm{UpdateArr}(\widehat{\sigma}_{1+\widehat{i}},\ (\widehat{l}_1, \widehat{i}),\ \widehat{n}_{\widehat{i}},\ \widehat{bty}) = \sigma_{2+\widehat{i}}$ such that (L1) $(\gamma, \sigma_{2+i}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_{2+\widehat{i}})$.

Given (J), (Q), (W), (A1), (C1), (E1), (H1), and (K1), by Vanilla C rule Write Entire Array we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{x} = \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{p},[\widehat{wea}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_{2+\widehat{\alpha}-1}, \Box, \Box, \mathrm{skip}) \parallel \widehat{C}_1)$.

Given (L1) and (V), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_{2+\widehat{\alpha}-1}, \Box, \Box, \mathrm{skip}) \parallel \widehat{C}_1)$.
By Definition 5.2.23 we have $wea1 \cong \widehat{wea}$. Given (S), $\mathcal{D}_1 :: (\mathrm{p}, [wea1])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{wea}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [wea1]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{wea}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::(\mathrm{p},[wea2])} ((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::(\mathrm{p},[wea1])} ((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1)$. Given $n = \widehat{n}$, we use Definition 5.2.18 to prove that $\mathrm{encrypt}(n) \cong \widehat{n}$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::(\mathrm{p},[wea])} ((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::(\mathrm{p},[wea1])} ((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::(\mathrm{p},[rao])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::(\mathrm{p},[rao])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$ by SMC^2 rule Public Array Read Out of Bounds Public Index, we have (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ i) \parallel C_1)$, (C) $\gamma(x) = (l,\ \text{public const}\ bty*)$, (D) $(e) \nvdash \gamma$, (E) $\sigma_1(l) = (\omega,\ \text{public const}\ bty*, 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{public}$

const $bty*$, public, 1)), (F) DecodePtr(public const $bty*$, 1, ω) $= [1,\ [(l_1, 0)],\ [1],\ 1]$, (G) $\sigma_1(l_1) = (\omega_1,$ public $bty, \alpha,$ PermL(Freeable, public bty, public, α)), (H) $(i < 0) \vee (i \geq \alpha)$, and (I) ReadOOB$(i, \alpha, l_1,$ public $bty, \sigma_1) = (n, 1, (l_2, \mu))$.

Given (J) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}]) \parallel \widehat{C})$ and ψ such that $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}]) \parallel \widehat{C})$, by Definition 5.2.22 we have (K) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (L) $C \cong_\psi \widehat{C}$, and (M) $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. Given (M), by Definition 5.2.20 we have (N) $e \cong_\psi \widehat{e}$ and $x \cong_\psi \widehat{x}$ such that (O) $x = \widehat{x}$.

Given ψ, (K), (N), and (L), by Lemma 5.2.50 we have (P) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C})$. Given (B) and (P), by the inductive hypothesis we have (Q) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{i}) \parallel \widehat{C}_1)$ and ψ_1 such that (R) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ i) \parallel C_1) \cong_{\psi_1} (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{i}) \parallel \widehat{C}_1)$ and (S) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$.

Given (R), by Definition 5.2.22 we have (T) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$, (U) $i \cong_{\psi_1} \widehat{i}$, and (V) $C_1 \cong_{\psi_1} \widehat{C}_1$. Given (D) and (U) by Lemmas 5.2.52 and 5.2.51, we have (W) $i = \widehat{i}$.

Given (C), (T), and (O), by Lemma 5.2.62 we have (X) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty}*)$ such that (Y) $l = \widehat{l}$ and (Z) public const $bty* \cong_{\psi_1}$ const $\widehat{bty}*$. Given (Z), by Definition 5.2.8 we have (A1) public $bty \cong_{\psi_1} \widehat{bty}$.

Given (E), (T), and (Y), by Lemma 5.2.63 we have (B1) $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty}*,$ public, 1)) such that (C1) $\omega \cong_{\psi_1} \widehat{\omega}$.

Given (F), (Z), and (C1), by Lemma 5.2.12 we have (D1) DecodePtr(const $\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ such that (E1) $l_1 = \widehat{l}_1$.

Given (G), (T), and (E1), by Lemma 5.2.63 we have (F1) $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \text{PermL}(\text{Freeable}, \widehat{bty}, \text{public}, \widehat{\alpha}))$ such that (G1) $\omega_1 \cong_{\psi_1} \widehat{bty}_1$, and (H1) $\alpha = \widehat{\alpha}$.

Given (H), (W), and (H1), we have (I1) $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{\alpha})$.

Given (I), (W), (H1), (E1), (A1), and (T), by Lemma 5.2.18 we have (J1) ReadOOB$(\widehat{i}, \widehat{\alpha}, \widehat{l}_1, \widehat{bty}, \widehat{\sigma}_1) = (\widehat{n}, 1)$ such that (K1) $n \cong_{\psi_1} \widehat{n}$.

Given (J), (Q), (X), (B1), (D1), (F1), (I1), and (J1), by Vanilla C rule Array Read Out of Bounds we have $\Sigma \triangleright$

$((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}]) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::(\mathrm{p},[\widehat{rao}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$.

Given (T), (K1), and (V), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$. By Definition 5.2.23 we have $rao \cong \hat{rao}$. Given (S), $\mathcal{D}_1 :: (\mathrm{p}, [rao])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{rao}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [rao]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{rao}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::(\mathrm{p},[rao1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::(\mathrm{p},[rao])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wao2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wao2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$ by SMC^2 rule Private Array Write Out of Bounds Public Index Private Value, we have (B) $(e_1) \nvdash \gamma$, (C) $(e_2) \vdash \gamma$, (D) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e_1)\ \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, i) \parallel C_1)$, (E) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2,$ $\mathrm{acc}, n)\ \parallel C_2)$, (F) $\gamma(x) = (l, \text{private const } bty*)$, (G) $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \mathrm{PermL}(\text{Freeable, private const } bty*, \text{private}$ (H) $\mathrm{DecodePtr}(\text{private const } bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1], 1]$, (I) $\sigma_2(l_1) = (\omega_1, \text{private } bty,\ \alpha, \mathrm{PermL}(\text{Freeable, private } bty, \text{privat}$ (J) $(i < 0) \vee (i \geq \alpha)$, and (K) $\mathrm{WriteOOB}(n, i, \alpha, l_1, \text{private } bty,\ \sigma_2,\ \Delta_2, \mathrm{acc}) = (\sigma_3,\ \Delta_3, 1, (l_2, \mu))$.

Given (L) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \parallel \widehat{C})$ and ψ such that (M) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma},$ $\square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \parallel \widehat{C})$, by Definition 5.2.22 we have (N) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (O) $x[e_1]\ = e_2 \cong_\psi \widehat{x}[\widehat{e}_1] = \widehat{e}_2$, and (P) $C \cong_\psi \widehat{C}$. By Definition 5.2.20 we have (Q) $e_1 \cong_\psi \widehat{e}_1$, (R) $e_2 \cong_\psi \widehat{e}_2$, and $x \cong_\psi \widehat{x}$ such that (S) $x = \widehat{x}$.

Given ψ, (N), (Q), and (P), by Lemma 5.2.50 we have (T) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e_1)\ \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma},\ \square, \square, \widehat{e}_1) \parallel \widehat{C})$ Given (D) and (T), by the inductive hypothesis we have (U) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma},\ \square, \square, \widehat{e}_1) \parallel \widehat{C})\ \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \hat{i}) \parallel \widehat{C}_1)$ and ψ_1 such that (V) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ i) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \hat{i}) \parallel \widehat{C}_1)$. Given (V), by Definition 5.2.22 we have (W) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$, (X) $i \cong_{\psi_1} \widehat{i}$, (Y) $C_1 \cong_{\psi_1} \widehat{C}_1$, and (Z) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$.

Given (B) and (X), by Lemma 5.2.51 we have (A1) $i = \widehat{i}$.

Given Axiom 5.2.1, we have $(l, \mu) \notin e_2$. Given (R), by Lemma 5.2.55 we have (B1) $e_2 \cong_{\psi_1} \widehat{e}_2$.

Given ψ_1, (W), (B1), and (Y), by Lemma 5.2.50 we have (C1) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, e_2) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{e}_2) \parallel \widehat{C}_1)$. Given (E) and (C1), by the inductive hypothesis we have (D1) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{e}_2) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \widehat{n}) \parallel \widehat{C}_2)$ and ψ_2 such that (E1) $((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, n) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \widehat{n}) \parallel \widehat{C}_2)$. Given (E1), by Definition 5.2.22 we have (F1) $(\gamma, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2)$, (G1) $n \cong_{\psi_2} \widehat{n}$, and (H1) $C_2 \cong_{\psi_2} \widehat{C}_2$, and (I1) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$.

Given (F), (F1), and (S), by Lemma 5.2.62 we have (J1) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \mathrm{const}\ \widehat{bty*})$ (K1) $l = \widehat{l}$ and (L1) $\mathrm{private\ const}\ bty* \cong_{\psi_2} \mathrm{const}\ \widehat{bty*}$. Given (L1), by Definition 5.2.8 we have (M1) $\mathrm{private}\ bty \cong_{\psi_2} \widehat{bty}$.

Given (G), (F1), and (K1), by Lemma 5.2.63 we have (N1) $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty*}, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty*}, \mathrm{public}, 1))$ such that (O1) $\omega \cong_{\psi_2} \widehat{\omega}$.

Given (H), (L1), and (O1), by Lemma 5.2.12 we have (P1) $\mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ (Q1) $l_1 = \widehat{l}_1$.

Given (I), (Q1), and (F1), by Lemma 5.2.63 we have (R1) $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, \widehat{\alpha}))$ such that (S1) $\omega_1 \cong_{\psi_2} \widehat{\omega}_1$, (T1) $\alpha = \widehat{\alpha}$.

Given (J), (A1), and (T1), we have (U1) $(\widehat{i} < 0) \vee (\widehat{i} \geq \widehat{\alpha})$.

Given (K), (G1), (A1), (T1), (Q1), (M1), and (F1), by Lemma 5.2.19 we have (V1) $\mathrm{WriteOOB}(\widehat{n}, \widehat{i}, \widehat{\alpha}, \widehat{l}_1, \widehat{bty}, \widehat{\sigma}_2) = (\widehat{\sigma}_3, 1)$ such that (W1) $(\gamma, \sigma_3) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_3)$.

Given (L), (U), (D1), (J1), (N1), (P1), (R1), (U1), and (V1), by Vanilla C rule Array Write Out of Bounds we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{wao}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$.

Given (W1) and (H1), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$.

By Definition 5.2.23 we have $wao2 \cong \widehat{wao}$. Given (Z), (I1), $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wao2])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{wao}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wao2]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{wao}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_2} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wao])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wao2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wao1])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wao2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3::\mathcal{L}_4::\mathcal{L}_5::\mathcal{L}_6::\mathcal{L}_7}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[iep])}$ $((1, \gamma^1, \sigma^1_6, \Delta^1_3, \mathrm{acc}, \mathrm{skip})\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_6, \Delta^{\mathrm{q}}_3, \mathrm{acc}, \mathrm{skip}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2))$ $\Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3::\mathcal{L}_4::\mathcal{L}_5::\mathcal{L}_6::\mathcal{L}_7}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[iep])}$ $((1, \gamma^1, \sigma^1_6, \Delta^1_3, \mathrm{acc}, \mathrm{skip})\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_6, \Delta^{\mathrm{q}}_3, \mathrm{acc}, \mathrm{skip}))$ by SMC^2 rule Private If Else (Variable Tracking), we have (B) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e)\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((1, \gamma^1, \sigma^1_1, \Delta^1_1, \mathrm{acc}, n^1)$ $\parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_1, \Delta^{\mathrm{q}}_1, \mathrm{acc}, n^{\mathrm{q}}))$, (C) $\{(e) \vdash \gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, (D) $\{\mathrm{Extract}(s_1,\ s_2, \gamma^{\mathrm{p}}) = (x_{list}, 0)\}^{\mathrm{q}}_{\mathrm{p}=1}$,
(E) $\{\mathrm{InitializeVariables}(x_{list}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_1, n^{\mathrm{p}}, \mathrm{acc}+1) = (\gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_2, \bar{l}^{\mathrm{p}}_2)\}^{\mathrm{q}}_{\mathrm{p}=1}$, (F) $((1, \gamma^1_1, \sigma^1_2, \Delta^1_1, \mathrm{acc}+1, s_1)\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}_1,$ $\sigma^{\mathrm{q}}_2, \Delta^{\mathrm{q}}_1, \mathrm{acc}+1, s_1)) \Downarrow^{\mathcal{L}_3}_{\mathcal{D}_2} ((1, \gamma^1_2, \sigma^1_3, \Delta^1_2, \mathrm{acc}+1, \mathrm{skip})\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}_2, \sigma^{\mathrm{q}}_3, \Delta^{\mathrm{q}}_2, \mathrm{acc}+1, \mathrm{skip}))$,
(G) $\{\mathrm{RestoreVariables}(x_{list}, \gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_3, \mathrm{acc}+1) = (\sigma^{\mathrm{p}}_4, \bar{l}^{\mathrm{p}}_4)\}^{\mathrm{q}}_{\mathrm{p}=1}$, (H) $((1, \gamma^1_1, \sigma^1_4, \Delta^1_2, \mathrm{acc}+1, s_2)\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}_1, \sigma^{\mathrm{q}}_4,$ $\Delta^{\mathrm{q}}_2, \mathrm{acc}+1, s_2)) \Downarrow^{\mathcal{L}_5}_{\mathcal{D}_3} ((1, \gamma^1_3, \sigma^1_5, \Delta^1_3, \mathrm{acc}+1, \mathrm{skip})\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}_3, \sigma^{\mathrm{q}}_5, \Delta^{\mathrm{q}}_3, \mathrm{acc}+1, \mathrm{skip}))$
(I) $\{\mathrm{ResolveVariables_Retrieve}(x_{list},\ \mathrm{acc}+1, \gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_5) = ([(v^{\mathrm{p}}_{t1}, v^{\mathrm{p}}_{e1}), ..., (v^{\mathrm{p}}_{tm}, v^{\mathrm{p}}_{em})], n^{\mathrm{p}}, \bar{l}^{\mathrm{p}}_6)\}^{\mathrm{q}}_{\mathrm{p}=1}$,
(J) $\mathrm{MPC}_{resolve}([n^1,\ ..., n^{\mathrm{q}}],\ [[(v^1_{t1}, v^1_{e1}), ..., (v^1_{tm}, v^1_{em})],\ ...,\ [(v^{\mathrm{q}}_{t1},\ v^{\mathrm{q}}_{e1}),\ ...,\ (v^{\mathrm{q}}_{tm}, v^{\mathrm{q}}_{em})]]) = [[v^1_1, ..., v^1_m], ..., [v^{\mathrm{q}}_1,\ ...,$ $v^{\mathrm{q}}_m]]$, (K) $\{\mathrm{ResolveVariables_Store}(x_{list},\ \mathrm{acc}+1, \gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_5,\ [v^{\mathrm{p}}_1, ..., v^{\mathrm{p}}_m]) = (\sigma^{\mathrm{p}}_6, \bar{l}^{\mathrm{p}}_7)\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\mathcal{L}_2 = (1, \bar{l}^1_2) \parallel ... \parallel (\mathrm{q}, \bar{l}^{\mathrm{q}}_2)$, $\mathcal{L}_4 = (1, \bar{l}^1_4) \parallel ... \parallel (\mathrm{q}, \bar{l}^{\mathrm{q}}_4)$, $\mathcal{L}_6 = (1, \bar{l}^1_6) \parallel ... \parallel (\mathrm{q}, \bar{l}^{\mathrm{q}}_6)$, and $\mathcal{L}_7 = (1, \bar{l}^1_7) \parallel ... \parallel (\mathrm{q}, \bar{l}^{\mathrm{q}}_7)$.

Given Axiom 5.2.1, by Theorem 5.2.2 we have (L) $\{(1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \sim (\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1$ $\mathrm{else}\ s_2)\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Given (L), (M) $((1, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2)\ \parallel ... \parallel\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2))$ and ψ such that (N) $((1, \gamma^1, \sigma^1,$

Δ^1, acc, if (e) s_1 else $s_2)$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, \text{if } (e)\ s_1 \text{ else } s_2))$ $\cong_\psi$ $((1, \widehat{\gamma}^1, \widehat{\sigma}^1, \Box, \Box, \text{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2)$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \widehat{\gamma}^{\mathrm{q}}, \widehat{\sigma}^{\mathrm{q}}, \Box, \Box, \text{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2))$, by Lemma 5.2.86, we have (O) $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \cong_\psi (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \text{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$. and therefore (P) $((1, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \text{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2)$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \text{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2))$.

Given (O), by Definition 5.2.22 we have (Q) $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (R) if (e) s_1 else $s_2 \cong_\psi$ if$(\widehat{e})$ $\widehat{s}_1$ else $\widehat{s}_2$. Given (R), by Definition 5.2.20 we have (S) $e \cong_\psi \widehat{e}$ such that (T) $s_1 \cong_\psi \widehat{s}_1$ and (U) $s_2 \cong_\psi \widehat{s}_2$.

Given ψ, (Q), and (S), by Lemma 5.2.50 we have (V) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e)$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \cong_\psi$ $((1, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e})$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}))$. Given (B) and (V), by the inductive hypothesis we have (W) $((1, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e})$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}_1}$ $((1, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n})$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}))$ and ψ_1 such that (X) $((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n^1)$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}})) \cong_{\psi_1}$ $((1, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n})$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}))$ and (Y) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$.

Given (X), by Definition 5.2.22 we have (Z) $\{(\gamma^{\mathrm{p}}, \sigma_1^{\mathrm{p}}) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (A1) $\{n^{\mathrm{p}} \cong_{\psi_1} \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given Axiom 5.2.1, we have $(l, \mu) \notin s_1$. Given (T), by Lemma 5.2.55 we have (B1) $s_1 \cong_{\psi_1} \widehat{s}_1$.

Given (D), by Lemma 5.2.32 we have (C1) that all updates to memory in either branch will be caught by variables $x \in x_{list}$.

Given (E) and (C1), by Lemma 5.2.33 we have (D1) $\forall x_i \in x_{list}, \mathrm{p} \in \{1...\mathrm{q}\}, (\gamma_1^{\mathrm{p}}, \sigma_2^{\mathrm{p}}) \models (x_i_else_\mathrm{acc} \equiv v_orig_i^{\mathrm{p}})$.

Given (E) and (Z), by Lemma 5.2.28 we have (E1) $\{(\gamma_1^{\mathrm{p}}, \sigma_2^{\mathrm{p}}) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$ such that (F1) $\{\sigma_2^{\mathrm{p}} = \sigma_1^{\mathrm{p}} :: \sigma_{temp1}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (E1) and (B1), by Lemma 5.2.50 we have (G1) $((1, \gamma_1^1, \sigma_2^1, \Delta_1^1, \mathrm{acc}+1, s_1)$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \gamma_1^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}+1, s_1))$ $\cong_{\psi_1}$ $((1, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{s}_1)$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{s}_1))$. Given (F) and (G1), by the inductive hypothesis we have (H1) $((1, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{s}_1)$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{s}_1)) \Downarrow'_{\widehat{\mathcal{D}}_2}$ $((1, \widehat{\gamma}_1, \widehat{\sigma}_2, \Box, \Box, \text{skip})$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \widehat{\gamma}_1, \widehat{\sigma}_2, \Box, \Box, \text{skip}))$ and ψ_2 such that (I1) $((1, \gamma_2^1, \sigma_3^1, \Delta_2^1, \mathrm{acc}+1, \text{skip})$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \gamma_2^{\mathrm{q}}, \sigma_3^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}+1, \text{skip})) \cong_{\psi_2}$ $((1, \widehat{\gamma}_1, \widehat{\sigma}_2, \Box, \Box, \text{skip})$ $\parallel$ ... $\parallel$ $(\mathrm{q}, \widehat{\gamma}_1, \widehat{\sigma}_2, \Box, \Box, \text{skip}))$ and (J1) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$.

Given (I1), by Definition 5.2.22 we have (K1) $\{(\gamma_2^{\mathrm{p}}, \sigma_3^{\mathrm{p}}) \cong_{\psi_2} (\widehat{\gamma}_1, \widehat{\sigma}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (K1), (E1), (F), and (H1), by Lemma 5.2.57 we have (L1) $\{(\gamma_1^{\mathrm{p}}, \sigma_3^{\mathrm{p}}) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (F) and (F1), by Lemma 5.2.34 we have (M1) $\{\sigma_3^{\mathrm{p}} = \sigma_3'^{\mathrm{p}} :: \sigma_{temp1}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ such that (N1) $\{\sigma_{temp1}'^{\mathrm{p}} = \sigma_{temp1}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (F1), (M1), (N1), and (D1), we have (O1) $\forall x_i \in x_{list}, \mathrm{p} \in \{1...\mathrm{q}\}, (\gamma_1^{\mathrm{p}}, \sigma_3^{\mathrm{p}}) \models (x_i_else_\mathrm{acc} \equiv v_orig_i^{\mathrm{p}})$.

Given (G), (C1), (O1), (E1), (L1), and (F1), by Lemma 5.2.35 we have (P1) $\{\forall x_i \in x_{list}, (\gamma_1^{\mathrm{p}}, \sigma_3^{\mathrm{p}}) \models (x_i \equiv v_{ti}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, (Q1) $\{\forall x_i \in x_{list}\ (\gamma_1^{\mathrm{p}}, \sigma_4^{\mathrm{p}}) \models (x_i_then_\mathrm{acc} \equiv v_{ti}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, (R1) $\{\sigma_4^{\mathrm{p}} = \sigma_1^{\mathrm{p}} :: \sigma_{temp2}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (S1) $\{(\gamma_1^{\mathrm{p}}, \sigma_4^{\mathrm{p}}) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given Axiom 5.2.1, we have $(l, \mu) \notin s_2$. Given (U), by Lemma 5.2.55 we have (T1) $s_2 \cong_{\psi_2} \widehat{s}_2$.

Given (S1) and (T1), by Lemma 5.2.50 we have (U1) $((1, \gamma_1^1, \sigma_4^1, \Delta_2^1, \mathrm{acc}+1, s_2) \parallel ... \parallel (\mathrm{q}, \gamma_1^{\mathrm{q}}, \sigma_4^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}+1, s_2)) \cong_{\psi_2} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_2))$. Given (H) and (U1), by the inductive hypothesis we have (V1) $((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_2)) \Downarrow'_{\widehat{\mathcal{D}}_3} ((1, \widehat{\gamma}_2, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}_2, \widehat{\sigma}_3, \square, \square, \mathrm{skip}))$ and ψ_3 such that (W1) $((1, \gamma_3^1, \sigma_5^1, \Delta_3^1, \mathrm{acc}+1, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma_3^{\mathrm{q}}, \sigma_5^{\mathrm{q}}, \Delta_3^{\mathrm{q}}, \mathrm{acc}+1, \mathrm{skip})) \cong_{\psi_3} ((1, \widehat{\gamma}_2, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}_2, \widehat{\sigma}_3, \square, \square, \mathrm{skip}))$ and (X1) $\mathcal{D}_3 \cong \widehat{\mathcal{D}}_3$.

Given (W1), by Definition 5.2.22 we have (Y1) $\{(\gamma_3^{\mathrm{p}}, \sigma_5^{\mathrm{p}}) \cong_{\psi_3} (\widehat{\gamma}_2, \widehat{\sigma}_3)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (Y1), (S1), (H), and (V1), by Lemma 5.2.57 we have (Z1) $\{(\gamma_1^{\mathrm{p}}, \sigma_5^{\mathrm{p}}) \cong_{\psi_3} (\widehat{\gamma}, \widehat{\sigma}_3)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (A1), (B), and (I), by Definition 5.2.19 and Lemma 5.2.36 we have (A2) $\{n^{\mathrm{p}} \cong \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (H) and (R1), by Lemma 5.2.34 we have (B2) $\{\sigma_5^{\mathrm{p}} = \sigma_5'^{\mathrm{p}} :: \sigma_{temp2}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ such that (C2) $\{\sigma_{temp2}'^{\mathrm{p}} = \sigma_{temp2}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (R1), (B2), (C2), and (Q1), we have (D2) $\forall x_i \in x_{list}, \mathrm{p} \in \{1...\mathrm{q}\}, (\gamma_1^{\mathrm{p}}, \sigma_5^{\mathrm{p}}) \models (x_i_then_\mathrm{acc} \equiv v_{ti}^{\mathrm{p}})$.

Given (I), (H), (A2), (Z1), (C1), and (D2), by Lemma 5.2.37 (E2) $\{\forall x_i \in x_{list}, (\gamma_1^{\mathrm{p}}, \sigma_5^{\mathrm{p}}) \models (x_i \equiv v_{ei}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (F2) $\{\forall x_i \in x_{list}, (\gamma_1^{\mathrm{p}}, \sigma_5^{\mathrm{p}}) \models (x_i_then_\mathrm{acc} \equiv v_{ti}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Subcase (G2) $\widehat{n} = 0$

Given (J), (A2), (E2), (F2), and (G2), by Axiom 5.2.6 we have (H2) $\{\forall i \in \{1...m\}, v_i^{\mathrm{p}} = v_{ei}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (K), (H2), (C1), (E2), and (Z1), by Lemma 5.2.38 we have (I2) $\{\forall x \in x_{list}, (\gamma^{\mathrm{p}}, \sigma_f^{\mathrm{p}}) \models (x \equiv v_{ei}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (J2) $\{(\gamma_1^{\mathrm{p}}, \sigma_6^{\mathrm{p}}) \cong_{\psi_3} (\widehat{\gamma}, \widehat{\sigma}_3)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (J2) and (Q), by Lemma 5.2.57 we have (K2) $\{(\gamma^{\mathrm{p}}, \sigma_6^{\mathrm{p}}) \cong_{\psi_3} (\widehat{\gamma}, \widehat{\sigma}_3)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (P), (W), (H1), (V1), and (G2), by Vanilla C rule Multiparty If Else False we have $\Sigma \triangleright ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \parallel ... \parallel\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: \widehat{\mathcal{D}}_3 :: (\mathrm{p}, [\widehat{mpief}])} ((1, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \parallel ... \parallel\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}))$.

Given (K2), by Definition 5.2.22 we have $((1, \gamma^1, \sigma_6^1, \Delta_3^1, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_6^{\mathrm{q}}, \Delta_3^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip})) \cong_{\psi_3} ((1, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \parallel ... \parallel\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}))$.
By Definition 5.2.23 we have $iep \cong \hat{mpief}$.
Given (Y), (J1), (X1), $\mathcal{D}_1 :: \mathcal{D}_2 :: \mathcal{D}_3 :: (\mathrm{p}, [iep])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: \widehat{\mathcal{D}}_3 :: (\mathrm{p}, [\widehat{mpief}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: \mathcal{D}_3 :: (\mathrm{p}, [iep]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: \widehat{\mathcal{D}}_3 :: (\mathrm{p}, [\widehat{mpief}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_3} \Sigma$.

Subcase (G3) $\widehat{n} \neq 0$

Given (J), (A2), (E2), (F2), and (G3), by Axiom 5.2.7 we have (H3) $\{\forall i \in \{1...m\}, v_i^{\mathrm{p}} = v_{ti}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (K), (C1), (H3), (L1), and (P1), by Lemma 5.2.39 we have (I3) $\{\forall x \in x_{list}, (\gamma^{\mathrm{p}}, \sigma_f^{\mathrm{p}}) \models (x \equiv v_{ti}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (J3) $\{(\gamma_1^{\mathrm{p}}, \sigma_6^{\mathrm{p}}) \cong_{\psi_3} (\widehat{\gamma}, \widehat{\sigma}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (J3) and (Q), by Lemma 5.2.57 we have (K3) $\{(\gamma^{\mathrm{p}}, \sigma_6^{\mathrm{p}}) \cong_{\psi_3} (\widehat{\gamma}, \widehat{\sigma}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (P), (W), (H1), (V1), and (G3), by Vanilla C rule Multiparty If Else True we have $\Sigma \triangleright ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \parallel ... \parallel\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: \widehat{\mathcal{D}}_3 :: (\mathrm{p}, [\widehat{mpiet}])} ((1, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel ... \parallel\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}))$.

Given (K3), by Definition 5.2.22 we have $((1, \gamma^1, \sigma_6^1, \Delta_3^1, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_6^{\mathrm{q}}, \Delta_3^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip})) \cong_{\psi_3} ((1, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel ... \parallel\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}))$.
By Definition 5.2.23 we have $iep \cong \hat{mpiet}$.
Given (Y), (J1), (X1), $\mathcal{D}_1 :: \mathcal{D}_2 :: \mathcal{D}_3 :: (\mathrm{p}, [iep])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: \widehat{\mathcal{D}}_3 :: (\mathrm{p}, [\widehat{mpiet}])$, by Lemma 5.2.58 we have

$\mathcal{D}_1 :: \mathcal{D}_2 :: \mathcal{D}_3 :: (\mathrm{p}, [iep]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: \widehat{\mathcal{D}}_3 :: (\mathrm{p}, [\widehat{mpiet}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_3} \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3::\mathcal{L}_4::\mathcal{L}_5::\mathcal{L}_6::\mathcal{L}_7}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[iepd])}$ $((1, \gamma^1, \sigma^1_6, \Delta^1_6, \mathrm{acc}, \mathrm{skip})\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_6, \Delta^{\mathrm{q}}_6, \mathrm{acc}, \mathrm{skip}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2))$ $\Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3::\mathcal{L}_4::\mathcal{L}_5::\mathcal{L}_6::\mathcal{L}_7}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[iepd])}$ $((1, \gamma^1, \sigma^1_6, \Delta^1_6, \mathrm{acc}, \mathrm{skip})\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_6, \Delta^{\mathrm{q}}_6, \mathrm{acc}, \mathrm{skip}))$ by SMC^2 rule Private If Else (Location Tracking), we have (B) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((1, \gamma^1, \sigma^1_1, \Delta^1_1, \mathrm{acc}, n^1)$ $\|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_1, \Delta^{\mathrm{q}}_1, \mathrm{acc}, n^{\mathrm{q}}))$, (C) $\{(e) \vdash \gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, (D) $\{\mathrm{Extract}(s_1, s_2, \gamma^{\mathrm{p}}) = (x_{list}, 1)\}^{\mathrm{q}}_{\mathrm{p}=1}$, (E) $\{\mathrm{Initialize}(\Delta^{\mathrm{p}}_1,$ $x_{list},\ \gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}_1,\ n^{\mathrm{p}}, \mathrm{acc}+1) = (\gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_2, \Delta^{\mathrm{p}}_2, \bar{l}^{\mathrm{p}}_2)\}^{\mathrm{q}}_{\mathrm{p}=1}$, (F) $((1, \gamma^1_1, \sigma^1_2, \Delta^1_2, \mathrm{acc}+1, s_1)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}_1, \sigma^{\mathrm{q}}_2, \Delta^{\mathrm{q}}_2, \mathrm{acc}+$ $1, s_1)) \Downarrow^{\mathcal{L}_3}_{\mathcal{D}_2} ((1, \gamma^1_2, \sigma^1_3, \Delta^1_3, \mathrm{acc}+1, \mathrm{skip})\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}_2, \sigma^{\mathrm{q}}_3, \Delta^{\mathrm{q}}_3, \mathrm{acc}+1, \mathrm{skip}))$, (G) $\{\mathrm{Restore}(\sigma^{\mathrm{p}}_3, \Delta^{\mathrm{p}}_3, \mathrm{acc}+1)$ $= (\sigma^{\mathrm{p}}_4, \Delta^{\mathrm{p}}_4, \bar{l}^{\mathrm{p}}_4)\}^{\mathrm{q}}_{\mathrm{p}=1}$, (H) $((1, \gamma^1_1, \sigma^1_4, \Delta^1_4, \mathrm{acc}+1, s_2)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}_1, \sigma^{\mathrm{q}}_4, \Delta^{\mathrm{q}}_4, \mathrm{acc}+1, s_2)) \Downarrow^{\mathcal{L}_5}_{\mathcal{D}_3} ((1, \gamma^1_3, \sigma^1_5, \Delta^1_5, \mathrm{acc}+$ $1, \mathrm{skip})\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}_3, \sigma^{\mathrm{q}}_5, \Delta^{\mathrm{q}}_5, \mathrm{acc}+1, \mathrm{skip}))$, (I) $\{\mathrm{Resolve_Retrieve}(\gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_5, \Delta^{\mathrm{p}}_5, \mathrm{acc}+1) = ([(v^{\mathrm{p}}_{t1}, v^{\mathrm{p}}_{e1}), ..., (v^{\mathrm{p}}_{tm},$ $v^{\mathrm{p}}_{em})],\ n^{\mathrm{p}}, \bar{l}^{\mathrm{p}}_6)\}^{\mathrm{q}}_{\mathrm{p}=1}$, (J) $\mathrm{MPC}_{resolve}([n^1, ..., n^{\mathrm{q}}],\ [[(v^1_{t1}, v^1_{e1}), ..., (v^1_{tm}, v^1_{em})], ..., [(v^{\mathrm{q}}_{t1}, v^{\mathrm{q}}_{e1}), ..., (v^{\mathrm{q}}_{tm}, v^{\mathrm{q}}_{em})]]) = [[v^1_1,$ $..., v^1_m], ...[v^{\mathrm{q}}_1, ..., v^{\mathrm{q}}_m]]$, (K) $\{\mathrm{Resolve_Store}(\Delta^{\mathrm{p}}_5, \sigma^{\mathrm{p}}_5, \mathrm{acc}+1, [v^{\mathrm{p}}_1, ..., v^{\mathrm{p}}_m]) = (\sigma^{\mathrm{p}}_6, \Delta^{\mathrm{p}}_6, \bar{l}^{\mathrm{p}}_7)\}^{\mathrm{q}}_{\mathrm{p}=1}$, $\mathcal{L}_2 = (1, \bar{l}^1_2)\ \|\ ...\ \|$ $(\mathrm{q}, \bar{l}^{\mathrm{q}}_2)$, $\mathcal{L}_4 = (1, \bar{l}^1_4) \| ... \| (\mathrm{q}, \bar{l}^{\mathrm{q}}_4)$, $\mathcal{L}_6 = (1, \bar{l}^1_6) \| ... \| (\mathrm{q}, \bar{l}^{\mathrm{q}}_6)$, and $\mathcal{L}_7 = (1, \bar{l}^1_7)\ \|\ ...\ \| (\mathrm{q}, \bar{l}^{\mathrm{q}}_7)$.

Given Axiom 5.2.1, by Theorem 5.2.2 we have (L) $\{(1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \sim (\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, \mathrm{if}\ (e)$ $s_1\ \mathrm{else}\ s_2)\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Given (L), (M) $((1, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2)\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2))$ and ψ such that (N) $((1, \gamma^1, \sigma^1,$ $\Delta^1, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)) \cong_{\psi} ((1, \widehat{\gamma}^1, \widehat{\sigma}^1, \Box, \Box, \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2)\ \|$ $...\ \|\ (\mathrm{q}, \widehat{\gamma}^{\mathrm{q}}, \widehat{\sigma}^{\mathrm{q}}, \Box, \Box, \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2))$, by Lemma 5.2.86, we have (O) $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \cong_{\psi}$ $(\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma},\ \Box, \Box,\ \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2)\}^{\mathrm{q}}_{\mathrm{p}=1}$. and therefore (P) $((1, \widehat{\gamma},\ \widehat{\sigma},\ \Box, \Box,\ \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2)\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma},\ \widehat{\sigma},\ \Box, \Box,\ \mathrm{if}(\widehat{e})$ $\widehat{s}_1\ \mathrm{else}\ \widehat{s}_2))$.

Given (O), by Definition 5.2.22 we have (Q) $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})\}^{\mathrm{q}}_{\mathrm{p}=1}$, and (R) $\mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2 \cong_{\psi} \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2$. Given (R), by Definition 5.2.20 we have (S) $e \cong_{\psi} \widehat{e}$ such that (T) $s_1 \cong_{\psi} \widehat{s}_1$ and (U) $s_2 \cong_{\psi} \widehat{s}_2$.

Given ψ, (Q), and (S), by Lemma 5.2.50 we have (V) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \cong_{\psi}$ $((1, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e})\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}))$. Given (B) and (V), by the inductive hypothesis we have (W) $((1, \widehat{\gamma}, \widehat{\sigma},$

$\square, \square, \widehat{e}) \ \| \ldots \| \ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}_1} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \ \| \ldots \| \ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}))$ and ψ_1 such that (X) $((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n^1) \ \| \ldots \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}})) \cong_{\psi_1} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \ \| \ldots \| \ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}))$ and (Y) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$.

Given (X), by Definition 5.2.22 we have (Z) $\{(\gamma^{\mathrm{p}}, \sigma_1^{\mathrm{p}}) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (A1) $\{n^{\mathrm{p}} \cong_{\psi_1} \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given Axiom 5.2.1, we have $(l, \mu) \notin s_1$. Given (T), by Lemma 5.2.55 we have (B1) $s_1 \cong_{\psi_1} \widehat{s}_1$.

Given (E) and (Z), by Lemma 5.2.41 we have (C1) $\{(\gamma_1^{\mathrm{p}}, \sigma_2^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (D), (E), by Lemma 5.2.40 we have (D1) all updates to a *constant location* dictated by variable x will have their original value stored within $\{\Delta_2^{\mathrm{p}}[\mathrm{acc}+1]\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (E1) $\{(\gamma_1^{\mathrm{p}}, \sigma_2^{\mathrm{p}}) \models (res_\mathrm{acc} \equiv n^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (F1) $\{\sigma_2^{\mathrm{p}} = \sigma_1^{\mathrm{p}} :: \sigma_{temp1}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (C1) and (B1), by Lemma 5.2.50 we have (G1) $((1, \gamma_1^1, \sigma_2^1, \Delta_1^1, \mathrm{acc}+1, s_1) \ \| \ldots \| \ (\mathrm{q}, \gamma_1^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}+1, s_1)) \cong_{\psi_1} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_1) \ \| \ldots \| \ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_1))$. Given (F) and (G1), by the inductive hypothesis we have (H1) $((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_1) \ \| \ldots \| \ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_1)) \Downarrow'_{\widehat{\mathcal{D}}_2} ((1, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \ \| \ldots \| \ (\mathrm{q}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}))$ and ψ_2 such that (I1) $((1, \gamma_2^1, \sigma_3^1, \Delta_2^1, \mathrm{acc}+1, \mathrm{skip}) \ \| \ldots \| \ (\mathrm{q}, \gamma_2^{\mathrm{q}}, \sigma_3^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}+1, \mathrm{skip})) \cong_{\psi_2} ((1, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \ \| \ldots \| \ (\mathrm{q}, \widehat{\gamma}_1, \widehat{\sigma}_2, \square, \square, \mathrm{skip}))$ and (J1) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$.

Given (I1), by Definition 5.2.22 we have (K1) $\{(\gamma_2^{\mathrm{p}}, \sigma_3^{\mathrm{p}}) \cong_{\psi_2} (\widehat{\gamma}_1, \widehat{\sigma}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (K1), (C1), (F), and (H1), by Lemma 5.2.57 we have (L1) $\{(\gamma_1^{\mathrm{p}}, \sigma_3^{\mathrm{p}}) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (F), by Lemma 5.2.42 we have (M1) $\{\Delta_3^{\mathrm{p}}[\mathrm{acc}+1]\}_{\mathrm{p}=1}^{\mathrm{q}}$ is *complete*.

Given (G), (M1), (L1), and (C1), by Lemma 5.2.43 we have (N1) $\{\Delta_4^{\mathrm{p}}[\mathrm{acc}+1]\}_{\mathrm{p}=1}^{\mathrm{q}}$ is *then-complete*, and (O1) $\{(\gamma_1^{\mathrm{p}}, \sigma_4^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given Axiom 5.2.1, we have (P1) $(l, \mu) \notin s_2$. Given (U) and (P1), by Lemma 5.2.55 we have (Q1) $s_2 \cong_{\psi_2} \widehat{s}_2$.

Given (O1) and (Q1), by Lemma 5.2.50 we have (R1) $((1, \gamma_1^1, \sigma_4^1, \Delta_2^1, \mathrm{acc}+1, s_2) \ \| \ldots \| \ (\mathrm{q}, \gamma_1^{\mathrm{q}}, \sigma_4^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}+1, s_2)) \cong_{\psi_2} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_2) \| \ldots \| \ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{s}_2))$. Given (H) and (R1), by the inductive hypothesis we have (S1)

$((1, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{s}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{s}_2)) \Downarrow'_{\widehat{\mathcal{D}}_3} ((1, \widehat{\gamma}_2, \widehat{\sigma}_3, \Box, \Box, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}_2, \widehat{\sigma}_3, \Box, \Box, \mathrm{skip}))$ and ψ_3 such that (T1) $((1, \gamma_3^1, \sigma_5^1, \Delta_3^1, \mathrm{acc}+1, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma_3^{\mathrm{q}}, \sigma_5^{\mathrm{q}}, \Delta_3^{\mathrm{q}}, \mathrm{acc}+1, \mathrm{skip})) \cong_{\psi_3} ((1, \widehat{\gamma}_2, \widehat{\sigma}_3, \Box, \Box, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}_2, \widehat{\sigma}_3, \Box, \Box, \mathrm{skip}))$ and (U1) $\mathcal{D}_3 \cong \widehat{\mathcal{D}}_3$.

Given (T1), by Definition 5.2.22 we have (V1) $\{(\gamma_3^{\mathrm{p}}, \sigma_5^{\mathrm{p}}) \cong_{\psi_3} (\widehat{\gamma}_2, \widehat{\sigma}_3)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (V1), (O1), (H), and (S1), by Lemma 5.2.57 we have (W1) $\{(\gamma_1^{\mathrm{p}}, \sigma_5^{\mathrm{p}}) \cong_{\psi_3} (\widehat{\gamma}, \widehat{\sigma}_3)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (A1), (B), (D), (E), (F), (G), (H), and (I), by Definition 5.2.19 and Lemma 5.2.44 we have (X1) $\{n^{\mathrm{p}} \cong \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (H), by Lemma 5.2.42 we have (Y1) $\{\Delta_5^{\mathrm{p}}[\mathrm{acc}+1]\}_{\mathrm{p}=1}^{\mathrm{q}}$ is *complete*. Given (N1), (H), and (Y1), by Lemma 5.2.45 we have that (Z1) $\{\Delta_5^{\mathrm{p}}[\mathrm{acc}+1]\}_{\mathrm{p}=1}^{\mathrm{q}}$ is *else-complete*.

Given (Z1), (F), (H), and (I), by Lemma 5.2.46 we have (A2) $\{\forall (l_i, \mu_i) = (v_{oi}^{\mathrm{p}}, v_{ti}^{\mathrm{p}}, 1, ty_i) \in \Delta_5^{\mathrm{p}}[\mathrm{acc}], (\sigma_3^{\mathrm{p}}) \models_l ((l_i, \mu_i) \equiv_{ty} v_{ti}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, (B2) $\{\forall (l_i, \mu_i) = (v_{ti}^{\mathrm{p}}, \mathrm{NULL}, 0, ty_i) \in \Delta_5^{\mathrm{p}}[\mathrm{acc}], (\sigma_3^{\mathrm{p}}) \models_l ((l_i, \mu_i) \equiv_{ty_i} v_{ti}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (C2) $\{\forall (l_i, \mu_i) = (v_{oi}^{\mathrm{p}}, v_{ti}^{\mathrm{p}}, j, ty_i) \in \Delta_5^{\mathrm{p}}[\mathrm{acc}], (\sigma_5^{\mathrm{p}}) \models_l ((l_i, \mu_i) \equiv_{ty_i} v_{ei}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Subcase (D2) $\widehat{n} = 0$

Given (J), (X1), (A2), (B2), (D2), and (C2), by Axiom 5.2.6 we have (E2) $\{\forall i \in \{1...m\}, v_i^{\mathrm{p}} = v_{ei}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (K), (W1), (Z1), (C2), and (E2), by Lemma 5.2.47 we have (F2) $\{(\gamma_1^{\mathrm{p}}, \sigma_6^{\mathrm{p}}) \cong_{\psi_3} (\widehat{\gamma}, \widehat{\sigma}_3)\}_{\mathrm{p}=1}^{\mathrm{q}}$ (G2) $\{\forall (l_i, \mu_i) = (v_{oi}^{\mathrm{p}}, v_{ti}^{\mathrm{p}}, j, ty_i) \in \Delta_1^{\mathrm{p}}[\mathrm{acc}], (\sigma_f^{\mathrm{p}}) \models_l ((l_i, \mu_i) \equiv_{ty_i} v_{ei}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (F2) and (Q), by Lemma 5.2.57 we have (H2) $\{(\gamma^{\mathrm{p}}, \sigma_6^{\mathrm{p}}) \cong_{\psi_3} (\widehat{\gamma}, \widehat{\sigma}_3)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (P), (W), (H1), (S1), and (D2), by Vanilla C rule Multiparty If Else False we have $\Sigma \triangleright ((1, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \mathrm{if}(\widehat{e})\ \widehat{s}_1\ \mathrm{else}\ \widehat{s}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::\widehat{\mathcal{D}}_3::(\mathrm{p}, [\widehat{mpief}])} ((1, \widehat{\gamma}, \widehat{\sigma}_3, \Box, \Box, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_3, \Box, \Box, \mathrm{skip}))$.

Given (H2), by Definition 5.2.22 we have $((1, \gamma^1, \sigma_6^1, \Delta_6^1, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_6^{\mathrm{q}}, \Delta_6^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip})) \cong_{\psi_3} ((1, \widehat{\gamma}, \widehat{\sigma}_3, \Box, \Box, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_3, \Box, \Box, \mathrm{skip}))$.
By Definition 5.2.23 we have $iepd \cong \widehat{mpief}$.

Given (Y), (J1), (U1), $\mathcal{D}_1 :: \mathcal{D}_2 :: \mathcal{D}_3 :: (\mathrm{p}, [iepd])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: \widehat{\mathcal{D}}_3 :: (\mathrm{p}, [\widehat{mpief}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: \mathcal{D}_3 :: (\mathrm{p}, [iepd]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: \widehat{\mathcal{D}}_3 :: (\mathrm{p}, [\widehat{mpief}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_3} \Sigma$.

Subcase (D3) $\widehat{n} \neq 0$

Given (J), (X1), (A2), (B2), (D3), and (C2), by Axiom 5.2.7 we have (E3) $\{\forall i \in \{1...m\}, v_i^{\mathrm{p}} = v_{ti}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (K), (L1), (Z1), (A2), (B2), and (E3), by Lemma 5.2.48 we have (F3) $\{(\gamma_1^{\mathrm{p}}, \sigma_6^{\mathrm{p}}) \cong_{\psi_3} (\widehat{\gamma}, \widehat{\sigma}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$ (G3) $\{\forall (l_i, \mu_i) = (v_{oi}^{\mathrm{p}}, v_{ti}^{\mathrm{p}}, 1, ty_i) \in \Delta_5^{\mathrm{p}}[\mathrm{acc}], (\sigma_6^{\mathrm{p}}) \models_l ((l_i, \mu_i) \equiv_{ty_i} v_{ti}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (H3) $\{\forall (l_i, \mu_i) = (v_{ti}^{\mathrm{p}}, \mathrm{NULL}, 0, ty_i) \in \Delta_5^{\mathrm{p}}[\mathrm{acc}], (\sigma_6^{\mathrm{p}}) \models_l ((l_i, \mu_i) \equiv_{ty_i} v_{ti}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (F3) and (Q), by Lemma 5.2.57 we have (I3) $\{(\gamma^{\mathrm{p}}, \sigma_6^{\mathrm{p}}) \cong_{\psi_3} (\widehat{\gamma}, \widehat{\sigma}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (P), (W), (H1), (S1), and (D3), by Vanilla C rule Multiparty If Else True we have $\Sigma \triangleright ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{if}(\widehat{e})\ \widehat{s}_1 \text{ else } \widehat{s}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: \widehat{\mathcal{D}}_3 :: (\mathrm{p}, [\widehat{mpiet}])} ((1, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}))$.

Given (I3), by Definition 5.2.22 we have $((1, \gamma^1, \sigma_6^1, \Delta_6^1, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_6^{\mathrm{q}}, \Delta_6^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip})) \cong_{\psi_3} ((1, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}))$.

By Definition 5.2.23 we have $iepd \cong \hat{mpiet}$.

Given (Y), (J1), (U1), $\mathcal{D}_1 :: \mathcal{D}_2 :: \mathcal{D}_3 :: (\mathrm{p}, [iepd])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: \widehat{\mathcal{D}}_3 :: (\mathrm{p}, [\widehat{mpiet}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: \mathcal{D}_3 :: (\mathrm{p}, [iepd]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: \widehat{\mathcal{D}}_3 :: (\mathrm{p}, [\widehat{mpiet}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_3} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, ++\ x) \parallel C) \Downarrow_{(\mathrm{p}, [pin3])}^{(\mathrm{p}, [(l,0)])} ((\mathrm{p}, \gamma, \sigma_1, \Delta, \mathrm{acc}, n_2) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, ++\ x) \parallel C) \Downarrow_{(\mathrm{p}, [pin3])}^{(\mathrm{p}, [(l,0)])} ((\mathrm{p}, \gamma, \sigma_1, \Delta, \mathrm{acc}, n_2) \parallel C)$ by SMC^2 rule Pre-Increment Private Int Variable, we have (B) $\gamma(x) = (l, \text{private int})$, (C) $\sigma(l) = (\omega, \text{private int}, 1, \mathrm{PermL}(\text{Freeable}, \text{private int}, \text{private}, 1))$, (D) $\mathrm{DecodeVal}(\text{private int}, \omega) = n_1$, (E) $n_2 = n_1 + \mathrm{encrypt}(1)$, and (F) $\mathrm{UpdateVal}(\sigma, l, n_2, \text{private int}) = \sigma_1$.

Given (G) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, ++\ \widehat{x}) \parallel \widehat{C})$ and ψ such that (H) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, ++\ x)\ \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, ++\ \widehat{x}) \parallel \widehat{C})$ by Definition 5.2.22 we have (I) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (J) $C \cong_\psi \widehat{C}$, and (K) $++\ x \cong_\psi ++\ \widehat{x}$. Given (K), by Definition 5.2.20 we have (L) $x = \widehat{x}$.

Given (B), (I), and (L), by Lemma 5.2.62 we have (M) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that (N) $l = \widehat{l}$ and (O) $\mathrm{private\ int} \cong_\psi \widehat{bty}$.

Given (C), (I), and (N), by Lemma 5.2.63 we have (P) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, 1))$ such that (Q) $\omega \cong_\psi \widehat{\omega}$.

Given (D), (O), and (Q), by Lemma 5.2.8 we have (R) $\mathrm{DecodeVal}(\widehat{bty}, \widehat{\omega}) = \widehat{n}_1$ such that (S) $n_1 \cong_\psi \widehat{n}_1$.

Given (E), by Definition 5.2.19 we have (T) $\mathrm{encrypt}(1) \cong_\psi 1$. Given (E) and (T), we have (U) $\widehat{n}_2 = \widehat{n}_1 + 1$ such that (V) $n_2 \cong_\psi \widehat{n}_2$.

Given (F), (I), (N), (V), and (O), by Lemma 5.2.14 we have (W) $\mathrm{UpdateVal}(\widehat{\sigma}, \widehat{l}, \widehat{n}_2, \widehat{bty}) = \widehat{\sigma}_1$ such that (X) $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given (G), (M), (P), (R), (U), and (W), by Vanilla C rule Pre-Increment Variable we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, ++\ \widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p}, [\widehat{pin}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}_2) \parallel \widehat{C})$.

Given (X), (V), and (J), by Definition 5.2.22 we have $((\mathrm{p}, \gamma, \sigma_1, \Delta, \mathrm{acc}, n_2) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}_2) \parallel \widehat{C})$.
By Definition 5.2.23 we have $pin3 \cong \hat{pin}$, and by Definition 5.2.25 we have $(\mathrm{p}, [pin3]) \cong (\mathrm{p}, [\widehat{pin}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ n_1) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin3])} ((\mathrm{p}, \gamma, \sigma_1, \Delta, \mathrm{acc}, n_2) \parallel C)$. The main difference is the value of x is equal instead of congruent, and we add 1 without encryption.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$

Given (A) $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$ by SMC^2 rule Pre-Increment Public Pointer Single Location, we have (B) $\gamma(x) = (l,\ \text{public}\ bty*)$, (C) $\sigma(l) = (\omega,\ \text{public}\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \text{public}\ bty*, \text{public}, 1))$, (D) $\mathrm{DecodePtr}(\text{public}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1]$, (E) $((l_2, \mu_2), 1) = \mathrm{GetLocation}((l_1, \mu_1), \tau(\text{public}\ bty), \sigma)$, and (F) $\mathrm{UpdatePtr}(\sigma,\ (l, 0),\ [1,\ [(l_2, \mu_2)],\ [1],\ 1], \text{public}\ bty*) = (\sigma_1,\ 1)$.

Given (G) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, ++\ \widehat{x}) \parallel \widehat{C})$ and ψ such that (H) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, ++\ \widehat{x}) \parallel \widehat{C})$, by Definition 5.2.22 we have (I) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (J) $C \cong_\psi \widehat{C}$, and (K) $++\ x \cong_\psi ++\ \widehat{x}$. Given (K), by Definition 5.2.20 we have (L) $x = \widehat{x}$.

Given (B), (I), and (L), by Lemma 5.2.62 we have (M) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that (N) $l = \widehat{l}$ and (O) $\text{public}\ bty* \cong_\psi \widehat{bty*}$.

Given (C), (I), and (N), by Lemma 5.2.63 we have (P) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty*}, \text{public}, 1))$ such that (Q) $\omega \cong_\psi \widehat{\omega}$.

Given (D), (O), and (Q) by Lemma 5.2.11 we have (R) $\mathrm{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$ (S) $[1,\ [(l_1, \mu_1)],\ [1],\ 1] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$. Given (S), by Definition 5.2.15 we have (T) $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$.

Given (O), by Definition 5.2.8 we have (U) $\text{public}\ bty \cong_\psi \widehat{bty}$.

Given (E), (T), (U), and (I), by Lemma 5.2.20 we have (V) $((\widehat{l}_2, \widehat{\mu}_2), 1) = \mathrm{GetLocation}((\widehat{l}_1, \widehat{\mu}_1), \tau(\widehat{bty}), \widehat{\sigma})$ such that (W) $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$.

Given (F), (I), (N), (W), and (O), by Lemma 5.2.17 we have (X) $\mathrm{UpdatePtr}(\widehat{\sigma}, (\widehat{l}, 0), [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], 1], \widehat{bty*}) = (\widehat{\sigma}_1, 1)$ such that (Y) $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given (G), (M), (P), (R), (V), and (X), by Vanilla C rule Pre-Increment Pointer we have $\Sigma\triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, ++\ \widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{pin1}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_2, \widehat{\mu}_2)) \parallel \widehat{C})$.

Given (Y), (W), and (J), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_2, \widehat{\mu}_2)) \parallel \widehat{C})$.

By Definition 5.2.23 we have $pin1 \cong \hat{pin1}$, by Definition 5.2.25 we have $(\mathrm{p}, [pin1]) \cong (\mathrm{p}, [\widehat{pin1}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin2])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin6])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin7])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin5])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ [\alpha,\ \bar{l}_1,\ \bar{j},\ i]) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$. We use Lemma 5.2.21 in place of Lemma 5.2.20 to reason about the use of $\mathrm{IncrementList}$ to increment every location, whereas $\mathrm{GetLocation}$ increments the single location. As for the resulting location that is returned, we reason about the true location of the pointer being ψ-congruent to the Vanilla C location that is returned.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin4])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ [n,\ \bar{l}_1,\ \bar{j},\ 1]) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin5])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ [\alpha,\ \bar{l}_1,\ \bar{j},\ i]) \parallel C)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{malloc}(e)) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[mal])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta,\ \mathrm{acc},\ (l, 0)) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{malloc}(e)) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[mal])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta,\ \mathrm{acc},\ (l, 0)) \parallel C_1)$ by SMC^2 rule

Public Malloc, we have (B) $\text{acc} = 0$, (C) $(e) \nvdash \gamma$, (D) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ n) \parallel C_1)$, (E) $l = \phi()$, and (F) $\sigma_2 = \sigma_1\big[l \rightarrow \big(\text{NULL}, \text{void}*, n, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, n)\big)\big]$.

Given (G) $((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \text{malloc}(\widehat{e})) \parallel \widehat{C})$ and ψ such that (H) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{malloc}(e)) \parallel C) \cong_{\psi} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \text{malloc}(\widehat{e})) \parallel \widehat{C})$, by Definition 5.2.22 we have (I) $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, (J) $C \cong_{\psi} \widehat{C}$, and (K) $\text{malloc}(e) \cong_{\psi} \text{malloc}(\widehat{e})$. Given (K), by Definition 5.2.20 we have (L) $e \cong_{\psi} \widehat{e}$.

Given (D), (I), (L), and (J), by Lemma 5.2.50 we have (M) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C) \cong_{\psi} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C})$. Given (M), by the inductive hypothesis we have (N) $((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \hat{n}) \parallel \widehat{C}_1)$ and ψ_1 such that (O) $((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ n) \parallel C_1) \cong_{\psi_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \hat{n}) \parallel \widehat{C}_1)$ and (P) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (O), by Definition 5.2.22 we have (Q) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$, (R) $n \cong_{\psi_1} \hat{n}$, and (S) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (E), by Axiom 5.2.2 we have (T) $\widehat{l} = \phi()$ and (U) $l = \widehat{l}$.

Given (D), (C), and (R), by Lemmas 5.2.52 and 5.2.51 we have (V) $n = \hat{n}$.

Given (F), (Q), (U), and (V), by Lemma 5.2.61 we have (W) $\widehat{\sigma}_2 = \widehat{\sigma}_1\big[\widehat{l} \rightarrow \big(\text{NULL}, \text{void}*, \hat{n}, \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, \hat{n})\big)\big]$ such that (X) $(\gamma, \sigma_2) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given (G), (N), (T), and (W), by Vanilla C rule Malloc we have $\Sigma \triangleright ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \text{malloc}(\widehat{e})) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::(\text{p}, [\widehat{mal}])} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, (\widehat{l}, 0)) \parallel \widehat{C}_1)$.

Given (X), (U), and (S), by Definition 5.2.22 we have $((\text{p}, \gamma,\ \sigma_2,\ \Delta,\ \text{acc},\ (l, 0)) \parallel C_1) \cong_{\psi_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, (\widehat{l}, 0)) \parallel \widehat{C}_1)$.

By Definition 5.2.23 we have $mal \cong \hat{mal}$. Given (P), $\mathcal{D}_1 :: (\text{p}, [mal])$ and $\widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{mal}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\text{p}, [mal]) \cong \widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{mal}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{pmalloc}(e,\ ty)) \parallel C) \Downarrow^{\mathcal{L}_1::(\text{p}, [(l,0)])}_{\mathcal{D}_1::(\text{p}, [malp])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta,\ \text{acc},\ (l, 0)) \parallel C_1)$

Given (A) $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{pmalloc}(e,\ ty)) \parallel C) \Downarrow^{\mathcal{L}_1::(\text{p}, [(l,0)])}_{\mathcal{D}_1::(\text{p}, [malp])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta,\ \text{acc},\ (l, 0)) \parallel C_1)$ by SMC^2 rule Private Malloc, we have (B) $(e) \nvdash \gamma$, (C) $(ty = \text{private}\ bty*) \vee (ty = \text{private}\ bty)$, (D) $\text{acc} = 0$, (E)

$((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e)\ \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ n) \parallel C_1)$, (F) $l = \phi()$, and (G) $\sigma_2 = \sigma_1\big[l \rightarrow \big(\mathrm{NULL},\ \mathrm{void}*,\ n \cdot \tau(ty), \mathrm{PermL}(\mathrm{Freeable}, \mathrm{void}*, \mathrm{private}, n \cdot \tau(ty))\big)\big]$.

Given (H) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \mathrm{malloc}(\widehat{e} \cdot \mathrm{sizeof}(\widehat{ty}))) \parallel \widehat{C})$ and ψ such that (I) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{pmalloc}(e,\ ty)) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \mathrm{malloc}(\widehat{e} \cdot \mathrm{sizeof}(\widehat{ty}))) \parallel \widehat{C})$, by Definition 5.2.22 we have (J) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (K) $C \cong_\psi \widehat{C}$, and (L) $\mathrm{pmalloc}(e,\ ty) \cong_\psi \mathrm{malloc}(\widehat{e} \cdot \mathrm{sizeof}(\widehat{ty}))$. Given (L), by Definition 5.2.20 we have (M) $e \cong_\psi \widehat{e}$ and (N) $ty \cong_\psi \widehat{ty}$.

Given (H), we have (O) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e} \cdot \mathrm{sizeof}(\widehat{ty})) \parallel \widehat{C})$.

Given (J), (K), and (M), by Lemma 5.2.50 we have (P) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e)\ \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}) \parallel \widehat{C})$. Given (E) and (P), by the inductive hypothesis we have (Q) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}) \parallel \widehat{C}_1)$ and ψ_1 such that (N) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}) \parallel \widehat{C}_1)$ and (O) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (N), by Definition 5.2.22 we have (P) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$, (Q) $n \cong_{\psi_1} \widehat{n}$ and (R) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (O) and (Q), we have (S) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \mathrm{sizeof}(\widehat{ty})) \parallel \widehat{C}_1)$.

Given $\widehat{ty}$, by Algorithm τ we have (T) $\widehat{n}_1 = \tau(\widehat{ty})$.

Given (S), (T), by Vanilla C rule Size of Type we have (U) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \mathrm{sizeof}(\widehat{ty})) \parallel \widehat{C}_1) \Downarrow'_{(\mathrm{p}, [\widehat{ty}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}_1)\ \parallel \widehat{C}_1)$.

Given (Q) and (U), we have (V) $\widehat{n} * \widehat{n}_1 = \widehat{n}_2$.

Given (O), (Q), (U), and (V), by Vanilla C rule Multiplication we have (W) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e} \cdot \mathrm{sizeof}(\widehat{ty})) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{ty}]) :: (\mathrm{p}, [\widehat{bm}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \widehat{n}_2) \parallel \widehat{C}_1)$.

Given (F), by Axiom 5.2.2 we have (X) $\widehat{l} = \phi()$ and (Y) $l = \widehat{l}$.

Given (B) and (Q), by (Z) $n = \widehat{n}$.

Given (G), (Y), (P), (V), (T), and (Z) by Lemma 5.2.69 we have (A1) $\widehat{\sigma}_2 = \widehat{\sigma}_1[\widehat{l} \rightarrow (\mathrm{NULL},\ \mathrm{void}*,\ \widehat{n}_2, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{void}*, \mathrm{public}, \widehat{n}_2))]$ such that (B1) $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$.

Given (H), (W), (X), and (A1), by Vanilla C rule Malloc we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \mathrm{malloc}(\widehat{e} \cdot \mathrm{sizeof}(\widehat{ty}))) \parallel \widehat{C})$ $\Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{ty}, \widehat{bm}]) :: (\mathrm{p}, [\widehat{mal}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, (\widehat{l}, 0)) \parallel \widehat{C}_1)$.

Given (B1), (Y), and (R), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta,\ \mathrm{acc},\ (l, 0)) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, (\widehat{l}, 0))$ $\parallel \widehat{C}_1)$.

By Definition 5.2.23 we have $malp \cong [\widehat{ty}, \widehat{bm}, \widehat{mal}]$.

Given (O), $\mathcal{D}_1 :: (\mathrm{p}, [malp])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{ty}, \widehat{bm}]) :: (\mathrm{p}, [\widehat{mal}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [malp]) \cong$ $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{ty}, \widehat{bm}, \widehat{mal}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{free}(x)) \parallel C) \Downarrow^{(\mathrm{p}, [(l,0),(l_1,0)])}_{(\mathrm{p}, [fre])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{free}(x)) \parallel C) \Downarrow^{(\mathrm{p}, [(l,0),(l_1,0)])}_{(\mathrm{p}, [fre])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ by SMC^2 rule Public Free, we have (B) $\gamma(x) = (l,\ \mathrm{public}\ bty*)$, (C) $\sigma(l) = (\omega, \mathrm{public}\ bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{public}\ bty*, \mathrm{public}, 1))$, (D) $\mathrm{acc} = 0$, (E) $\mathrm{DecodePtr}(\mathrm{public}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$, (F) $\mathrm{CheckFreeable}(\gamma, [(l_1, 0)], [1], \sigma) = 1$, and (G) $\mathrm{Free}(\sigma,\ l_1) = (\sigma_1, (l_1, 0))$.

Given (H) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \mathrm{free}(\widehat{x})) \parallel \widehat{C})$ and ψ such that (I) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{free}(x)) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box,$ $\mathrm{free}(\widehat{x}))\ \parallel \widehat{C})$, by Definition 5.2.22 we have (J) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (K) $C \cong_\psi \widehat{C}$, and (L) $\mathrm{free}(x) \cong_\psi \mathrm{free}(\widehat{x})$. Given (L), by Definition 5.2.20 we have (M) $x = \widehat{x}$.

Given (B), (J), and (M), by Lemma 5.2.62 we have (N) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that (O) $l = \widehat{l}$ and (P) $\mathrm{public}\ bty* \cong_\psi$ $\widehat{bty}*$.

Given (C), (J), and (O), by Lemma 5.2.63 we have (Q) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}*, \mathrm{public}, 1))$ such that (R) $\omega \cong_\psi \widehat{\omega}$.

Given (E), (P), and (R), by Lemma 5.2.11 we have (S) $\mathrm{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ such that (T) $[1, [(l_1, 0)], [1], 1] \cong_\psi [1, [(\widehat{l}_1, 0)], [1], 1]$. Given (T), by Definition 5.2.15 we have (U) $l_1 \cong_\psi \widehat{l}_1$.

Given (F), (J), and (U), by Axiom 5.2.3 we have (V) $\mathrm{CheckFreeable}(\widehat{\gamma}, [(\widehat{l}_1, 0)], [1], \widehat{\sigma}) = 1$.

Given (G), (J), and (U), by Lemma 5.2.22 we have (W) $\text{Free}(\widehat{\sigma}, \widehat{l_1}) = \widehat{\sigma}_1$ such that (X) $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$.

Given (H), (N), (Q), (S), (V), and (W), by Vanilla C rule Free we have $\Sigma \triangleright ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \text{free}(\widehat{x})) \parallel \widehat{C}) \Downarrow'_{(\text{p},[\widehat{fre}])}$ $((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \text{skip}) \parallel \widehat{C})$.

Given (X) and (K), by Definition 5.2.22 we have $((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ \text{skip}) \parallel C) \cong_\psi ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \text{skip}) \parallel \widehat{C})$.
By Definition 5.2.23 we have $fre \cong \hat{fre}$, and by Definition 5.2.25 we have $(\text{p}, [fre]) \cong (\text{p}, [\widehat{fre}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{pfree}(x)) \parallel C) \Downarrow^{(\text{p},[(l,0),(l_1,0)])}_{(\text{p},[pfre])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ \text{skip}) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{free}(x)) \parallel C) \Downarrow^{(\text{p},[(l,0),(l_1,0)])}_{(\text{p},[fre])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ \text{skip}) \parallel C)$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)])}_{\mathcal{D}_1::(\text{p},[cl1])} ((\text{p}, \gamma,\ \sigma_3,\ \Delta_1,\ \text{acc},\ (l,0)) \parallel C_1)$

Given (A) $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)])}_{\mathcal{D}_1::(\text{p},[cl1])} ((\text{p}, \gamma,\ \sigma_3,\ \Delta_1,\ \text{acc},\ (l,0)) \parallel C_1)$ by SMC^2 rule Cast Private Location, we have (B) $(ty = \text{private}\ bty*)$, (C) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ (l,0))$- $\parallel C_1)$, (D) $\sigma_1 = \sigma_2\big[l \to \big(\omega,\ \text{void}*,\ n, \text{PermL}(\text{Freeable}, \text{void}*, \text{private}, n)\big)\big]$, and (E) $\sigma_3 = \sigma_2\big[l \to \big(\omega,\ ty,\ \frac{n}{\tau(ty)}, \text{PermL}(\text{Freeable}, ty, \text{private}, \frac{n}{\tau(ty)})\big)\big]$.

Given (F) $((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{ty})\ \widehat{e}) \parallel \widehat{C})$ and ψ such that (G) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ (ty)\ e) \parallel C) \cong_\psi ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{ty})\ \widehat{e})$ $\parallel \widehat{C})$, by Definition 5.2.22 we have (H) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (I) $C \cong_\psi \widehat{C}$, and (J) $(ty)\ e \cong_\psi (\widehat{ty})\ \widehat{e}$. Given (J), by Definition 5.2.20 we have (K) $ty \cong_\psi \widehat{ty}$ and (L) $e \cong_\psi \widehat{e}$.

Given (B) and (K), by Definition 5.2.8 we have (M) $(\widehat{ty} = \widehat{bty}*)$.

Given (H), (L), and (I), by Lemma 5.2.50 we have (N) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C) \cong_\psi ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C})$ Given (C) and (N), by the inductive hypothesis we have (O) $((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}, 0)) \parallel \widehat{C}_1)$ and ψ_1 such that (P) $((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ (l,0)) \parallel C_1) \cong_{\psi_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}, 0)) \parallel \widehat{C}_1)$ (Q) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (P), by Definition 5.2.22 we have (R) $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, (S) $C_1 \cong_\psi \widehat{C}_1$, and (T) $(l,0) \cong_{\psi_1} (\widehat{l}, 0)$.

Given (D), (R), and (T), by Lemma 5.2.69 we have (U) $\widehat{\sigma}_1 = \widehat{\sigma}_2[\widehat{l} \rightarrow (\widehat{\omega}, \text{void}, \widehat{n}, \text{PermL}(\text{Freeable}, \text{void}, \text{public}, \widehat{n}))]$ (V) $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$, and (W) $\exists ty_1, \widehat{ty}_1$ such that $ty_1 \cong \widehat{ty}_1$ and $\frac{n}{\tau(ty_1)} = \frac{\widehat{n}}{\tau(\widehat{ty}_1)}$.

Given Axiom 5.2.1, (W), and (K), we have (X) $\frac{n}{\tau(ty)} = \frac{\widehat{n}}{\tau(\widehat{ty})}$.

Given (E), (V), (T), (K), and (X), by Lemma 5.2.61 we have (Y) $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{ty}, \frac{\widehat{n}}{\tau(\widehat{ty})}, \text{PermL}(\text{Freeable}, \widehat{ty}, \text{public}, \frac{\widehat{n}}{\tau(\widehat{ty})}))]$ such that (Z) $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$.

Given (F), (M), (O), (U), and (Y), by Vanilla C rule Cast Location we have $\Sigma \triangleright ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, (\widehat{ty})\ \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{cl}])} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_3, \Box, \Box, (\widehat{l}, 0)) \parallel \widehat{C}_1)$.

Given (Z), (T), and (S), by Definition 5.2.22 we have $((\text{p}, \gamma,\ \sigma_3,\ \Delta_1,\ \text{acc},\ (l, 0)) \parallel C_1) \cong_{\psi_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_3, \Box, \Box, (\widehat{l}, 0)) \parallel \widehat{C}_1)$.

By Definition 5.2.23 we have $cl1 \cong \hat{cl}$. Given (Q), $\mathcal{D}_1 :: (\text{p}, [cl1])$ and $\widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{cl}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\text{p}, [cl1]) \cong \widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{cl}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0)])}_{\mathcal{D}_1 :: (\text{p}, [cl])} ((\text{p}, \gamma,\ \sigma_3,\ \Delta_1,\ \text{acc},\ (l, 0)) \parallel C_1)$

Given (A) $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0)])}_{\mathcal{D}_1 :: (\text{p}, [cl])} ((\text{p}, \gamma,\ \sigma_3,\ \Delta_1,\ \text{acc},\ (l, 0)) \parallel C_1)$ by SMC^2 rule Cast Public Location, we have $\text{acc} = 0$, (B) $(ty = \text{public}\ bty*)$, (C) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ (l,\ 0)) \parallel C_1)$, (D) $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega,\ \text{void}*,\ n,\ \text{PermL}(\text{Freeable}, \text{void}*, \text{public}, n)\big)\big]$, and (E) $\sigma_3 = \sigma_2\big[l \rightarrow \big(\omega,\ ty,\ \frac{n}{\tau(ty)}, \text{PermL}(\text{Freeable}, ty, \text{public}, \frac{n}{\tau(ty)})\big)\big]$.

Given (F) $((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, (\widehat{ty})\ \widehat{e}) \parallel \widehat{C})$ and ψ such that (G) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ (ty)\ e) \parallel C) \cong_\psi ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, (\widehat{ty})\ \widehat{e}) \parallel \widehat{C})$, by Definition 5.2.22 we have (H) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (I) $C \cong_\psi \widehat{C}$, and (J) $(ty)\ e \cong_\psi (\widehat{ty})\ \widehat{e}$. Given (J), by Definition 5.2.20 we have (K) $ty \cong_\psi \widehat{ty}$ and (L) $e \cong_\psi \widehat{e}$.

Given (B) and (K), by Definition 5.2.8 we have (M) $(\widehat{ty} = \widehat{bty}*)$.

Given (H), (L), and (I), by Lemma 5.2.50 we have (N) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e) \parallel C) \cong_\psi ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \widehat{e}) \parallel \widehat{C})$ Given

(C) and (N), by the inductive hypothesis we have (O) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}, 0)) \parallel \widehat{C}_1)$ and ψ_1 such that (P) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ (l, 0)) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}, 0)) \parallel \widehat{C}_1)$ (Q) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (P), by Definition 5.2.22 we have (R) $(\gamma,\ \sigma_1) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_1)$, (S) $C_1 \cong_\psi \widehat{C}_1$, and (T) $(l, 0) \cong_{\psi_1} (\widehat{l}, 0)$.

Given (T), by Lemma 5.2.65 we have (U) $l = \widehat{l}$.

Given (D), (R), and (U), by Lemma 5.2.68 we have (V) $\widehat{\sigma}_1 = \widehat{\sigma}_2[\widehat{l} \rightarrow (\widehat{\omega}, \mathrm{void}*, \widehat{n}, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{void}*, \mathrm{public}, \widehat{n}))]$ such that (W) $(\gamma,\ \sigma_2) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_2)$ and (X) $n = \widehat{n}$.

Given (B) and (K), by Lemma 5.2.70 we have (Y) $\tau(ty) = \tau(\widehat{ty})$.

Given (E), (X), (Y), (U), and (W) by Lemma 5.2.61 we have (Z) $\widehat{\sigma}_3 = \widehat{\sigma}_2[\widehat{l} \rightarrow (\widehat{\omega}, \widehat{ty}, \frac{\widehat{n}}{\tau(\widehat{ty})}, \mathrm{PermL}(\mathrm{Freeable}, \widehat{ty}, \mathrm{public}, \frac{\widehat{n}}{\tau(\widehat{ty})}))]$ such that (A1) $(\gamma,\ \sigma_3) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma}_3)$.

Given (F), (M), (O), (U), and (Z), by Vanilla C rule Cast Location we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{ty})\ \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{cl}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, (\widehat{l}, 0)) \parallel \widehat{C}_1)$.

Given (A1), (T), and (S), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_1,\ \mathrm{acc},\ (l, 0)) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, (\widehat{l}, 0)) \parallel \widehat{C}_1)$.

By Definition 5.2.23 we have $cl \cong \hat{cl}$. Given (Q), $\mathcal{D}_1 :: (\mathrm{p}, [cl])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{cl}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [cl]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{cl}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1 :: (\mathrm{p}, [cv])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1 :: (\mathrm{p}, [cv])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1)$ by SMC^2 rule Cast Public Value, we have (B) $(e) \nvdash \gamma$, (C) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$, (D) $(ty = \mathrm{public}\ bty)$, and (E) $n_1 = \mathrm{Cast}(\mathrm{public},\ ty,\ n)$.

Given (F) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma},\ \square, \square, (\widehat{ty})\ \widehat{e}) \parallel \widehat{C})$ and ψ such that (G) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma},\ \square, \square, (\widehat{ty})\ \widehat{e}) \parallel \widehat{C})$, by Definition 5.2.22 we have (H) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (I) $C \cong_\psi \widehat{C}$, and (J) $(ty)\ e \cong_\psi (\widehat{ty})\ \widehat{e}$. Given (J), by Definition 5.2.20 we have (K) $ty \cong_\psi \widehat{ty}$ (L) $e \cong_\psi \widehat{e}$.

Given (H), (L), and (I), by Lemma 5.2.50 we have (M) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C})$. Given (M), by the inductive hypothesis we have (N) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$ and ψ_1 such that (O) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$ and (P) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (O), by Definition 5.2.22 we have (Q) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$, (R) $n \cong_{\psi_1} \widehat{n}$ and (S) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (B), (C), and (R), by Lemmas 5.2.52 and 5.2.51 we have (T) $n = \widehat{n}$.

Given (E), (K), and (T), by Lemma 5.2.23 we have (U) $\widehat{n}_1 = \mathrm{Cast}(\mathrm{public}, \widehat{ty}, \widehat{n})$ such that (V) $n_1 \cong_{\psi_1} \widehat{n}_1$.

Given (F), (M), and (U), by Vanilla C rule Cast Value we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma},\ \square, \square, (\widehat{ty})\ \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{cv}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}_1) \parallel \widehat{C}_1)$.

Given (Q), (V), and (S), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}_1) \parallel \widehat{C}_1)$. By Definition 5.2.23 we have $cv \cong \hat{cv}$. Given (P), $\mathcal{D}_1 :: (\mathrm{p}, [cv])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{cv}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [cv]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{cv}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1 :: (\mathrm{p}, [cv1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1 :: (\mathrm{p}, [cv])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1)$. The main difference is using Lemma 5.2.24 in place of Lemma 5.2.23, as we are reasoning about private values that are congruent, whereas the previous case has public values that are equivalent.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \&x) \parallel C) \Downarrow^{\epsilon}_{(\mathrm{p}, [loc])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l, 0)) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \&x) \parallel C) \Downarrow^{\epsilon}_{(\mathrm{p}, [loc])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l, 0)) \parallel C)$ by SMC^2 rule Address Of, we have (B) $\gamma(x) = (l,\ ty)$.

Given (C) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \&\widehat{x}) \parallel \widehat{C})$ and ψ such that (D) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \&x) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \&\widehat{x}) \parallel \widehat{C})$, by Definition 5.2.22 we have (E) $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, (F) $C \cong_{\psi} \widehat{C}$, and (G) $\&x \cong_{\psi} \&\widehat{x}$. Given (G), by Definition 5.2.20

we have (H) $x = \widehat{x}$.

Given (B), (E), and (H), by Lemma 5.2.62 we have (I) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{ty})$ such that (J) $l = \widehat{l}$ and (K) $ty \cong_{\psi} \widehat{ty}$.

Given (C) and (I), by Vanilla C rule Address Of we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \&\widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p}, [\widehat{loc}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}, 0)) \parallel \widehat{C})$.

Given (E), (J), and (F), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l, 0)) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}, 0)) \parallel \widehat{C})$.
By Definition 5.2.23 we have $loc \cong \hat{loc}$, and by Definition 5.2.25 we have $(\mathrm{p}, [loc]) \cong (\mathrm{p}, [\widehat{loc}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{sizeof}(ty)) \parallel C) \Downarrow^{\epsilon}_{(\mathrm{p}, [ty])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{sizeof}(ty)) \parallel C) \Downarrow^{\epsilon}_{(\mathrm{p}, [ty])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$ by SMC^2 rule Size of Type, we have (B) $(ty) \nvdash \gamma$ and (C) $n = \tau(ty)$.

Given (D) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{sizeof}(\widehat{ty})) \parallel \widehat{C})$ and ψ such that (E) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{sizeof}(ty)) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{sizeof}(\widehat{ty})) \parallel \widehat{C})$, by Definition 5.2.22 we have (F) $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, (G) $C \cong_{\psi} \widehat{C}$, and (H) $\mathrm{sizeof}(ty) \cong_{\psi} \mathrm{sizeof}(\widehat{ty})$.
Given (H), by Definition 5.2.20 we have (I) $ty \cong_{\psi} \widehat{ty}$.

Given (B), (C), and (I), by Lemma 5.2.70 we have (J) $\widehat{n} = \tau(\widehat{ty})$ and (K) $n = \widehat{n}$.

Given (D) and (J), by Vanilla C rule Size of Type we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{sizeof}(\widehat{ty})) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p}, [\widehat{ty}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}) \parallel \widehat{C})$.

Given (F), (K), and (G), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}) \parallel \widehat{C})$.
By Definition 5.2.23 we have $ty \cong \hat{ty}$, and by Definition 5.2.25 we have $(\mathrm{p}, [ty]) \cong (\mathrm{p}, [\widehat{ty}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [inp])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

Given (A) $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[inp])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$ by SMC^2 rule SMC Input Public Value, we have (B) $(e) \nvdash \gamma$, (C) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1)$, (D) $\gamma(x) = (l, \mathrm{public}\ bty)$, (E) $\mathrm{acc} = 0$, (F) $\mathrm{InputValue}(x, n) = n_1$, and (G) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, x = n_1) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, \mathrm{skip}) \parallel C_2)$.

Given (H) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{mcinput}(\widehat{x}, \widehat{e})) \parallel \widehat{C})$ and ψ such that (I) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e)) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{mcinput}(\widehat{x}, \widehat{e})) \parallel \widehat{C})$, by Definition 5.2.22 we have (J) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, (K) $\mathrm{smcinput}(x,\ e) \cong_\psi \mathrm{mcinput}(\widehat{x}, \widehat{e})$, and (L) $C \cong_\psi \widehat{C}$. Given (K), by Definition 5.2.20 we have (M) $e \cong_\psi \widehat{e}$ and $x \cong_\psi \widehat{x}$ such that (N) $x = \widehat{x}$.

Given (J), (L), and (M), by Lemma 5.2.50 we have (O) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc}, e) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma},\ \square, \square, \widehat{e}) \parallel \widehat{C})$. Given (C) and (O), by the inductive hypothesis we have (P) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma},\ \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$ and ψ_1 such that (Q) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \quad \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$. (R) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (Q), by Definition 5.2.22 we have (S) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$ (T) $n \cong_{\psi_1} \widehat{n}$, and (U) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (T) and (B), by Lemmas 5.2.52 and 5.2.51 we have (V) $n = \widehat{n}$.

Given (D), (J), and (N), by Lemma 5.2.62 we have (W) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that (X) $l = \widehat{l}$ and (Y) $\mathrm{public}\ bty \cong_{\psi_1} \widehat{bty}$.

Given (F), (N), and (V), by Lemma 5.2.71 we have (Z) $\mathrm{InputValue}(\widehat{x}, \widehat{n}) = \widehat{n}_1$ such that (A1) $n_1 \cong_{\psi_1} \widehat{n}_1$.

Given (A1) and (N), by Definition 5.2.20 we have (B1) $x = n_1 \cong_{\psi_1} \widehat{x} = \widehat{n}_1$.

Given (S), (B1), and (U), by Lemma 5.2.50 we have (C1) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, x = n_1) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{x} = \widehat{n}_1) \parallel \widehat{C}_1)$. Given (G) and (C1), by the inductive hypothesis we have (D1) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{x} = \widehat{n}_1) \parallel \widehat{C}_1) \Downarrow'_{\widehat{\mathcal{D}}_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$ and ψ_2 such that (E1) $((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, \mathrm{skip}) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$. Given (E1), by Definition 5.2.22 we have (F1) $(\gamma, \sigma_2) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2)$ (G1) $C_2 \cong_{\psi_2} \widehat{C}_2$, and (H1) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$.

Given (H), (P), (W), (Z), and (D1), by Vanilla C rule Input Value we have $\Sigma\triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{mcinput}(\widehat{x}, \widehat{e})) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1::\widehat{\mathcal{D}}_2::(\mathrm{p},[\widehat{inp}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_2)$.

Given (F1) and (G1), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2) \cong_{\psi_2} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip})$- $\parallel \widehat{C}_2)$.

By Definition 5.2.23 we have $inp \cong \hat{inp}$. Given (R), (H1), $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [inp])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{inp}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [inp]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{p}, [\widehat{inp}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_2} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[inp2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[inp])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[inp1])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_3)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[inp])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$. The difference is an additional use of the inductive hypothesis to evaluate e_2, which contains the length of the array to be read in, and Lemma 5.2.72 to reason about the use of $\mathrm{InputArray}$ in place of Lemma 5.2.71 to reason about the use of $\mathrm{InputValue}$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[inp3])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_3)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[inp1])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_3)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[out2])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[out2])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule SMC Output Private Value, we have (B) $(e) \nvdash \gamma$, (C) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc}, n) \parallel C_1)$, (D) $\gamma(x) = (l,\ \mathrm{private}\ bty)$, (E) $\sigma_1(l) = (\omega,\ \mathrm{private}\ bty,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty, \mathrm{private}, 1))$, (F) $\mathrm{DecodeVal}(\mathrm{private}\ bty,\ \omega) = n_1$, and (G) $\mathrm{OutputValue}(x,\ n, n_1)$.

Given (H) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{mcoutput}(\widehat{x}, \widehat{e})) \parallel \widehat{C})$ and ψ such that (I) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{mcoutput}(\widehat{x}, \widehat{e})) \parallel \widehat{C})$, by Definition 5.2.22 we have (J) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})$, (K) $\mathrm{smcoutput}(x,\ e) \cong_\psi \mathrm{mcoutput}(\widehat{x}, \widehat{e})$, and (L) $C \cong_\psi \widehat{C}$. Given (K), by Definition 5.2.20 we have (M) $e \cong_\psi \widehat{e}$ and $x \cong_\psi \widehat{x}$ such that (N) $x = \widehat{x}$.

Given (J), (M), and (L), by Lemma 5.2.50 we have (O) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C})$. Given (C) and (O), by the inductive hypothesis we have (P) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$ and ψ_1 such that (Q) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$ and (R) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (Q), by Definition 5.2.22 we have (S) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$ (T) $n \cong_{\psi_1} \widehat{n}$, and (U) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (D), (S), and (N), by Lemma 5.2.62 we have (V) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty})$ such that (W) $l = \widehat{l}$ and (X) $\mathrm{private}\ bty \cong_{\psi_1} \widehat{bty}$.

Given (E), (S), and (W), by Lemma 5.2.63 we have (Y) $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, 1))$ such that (Z) $\omega \cong_{\psi_1} \widehat{\omega}$.

Given (F), (X), and (Z), by Lemma 5.2.8 we have (A1) $\mathrm{DecodeVal}(\widehat{bty}, \widehat{\omega}) = \widehat{n}_1$ such that (B1) $n_1 \cong_{\psi_1} \widehat{n}_1$.

Given (B), (C), and (T), by Lemmas 5.2.52 and 5.2.51 we have (C1) $n = \widehat{n}$.

Given (G), (N), (C1), and (B1), by Lemma 5.2.73 we have (D1) $\mathrm{OutputValue}(\widehat{x}, \widehat{n}, \widehat{n}_1)$ such that we have congruent output files.

Given (H), (P), (V), (Y), (A1), and (D1), by Vanilla C rule Output Value we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \mathrm{mcoutput}(\widehat{x}, \widehat{e})) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{out}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \parallel \widehat{C}_1)$.

Given (S) and (U), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \mathrm{skip}) \parallel \widehat{C}_1)$. By Definition 5.2.23 we have $out2 \cong \hat{out}$. Given (R), $\mathcal{D}_1 :: (\mathrm{p}, [out2])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{out}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [out2]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{out}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [out])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [out2])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, \mathrm{smcoutput}(x, e_1, e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,0),\ldots,(l_1,\alpha-1)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[out1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[out2])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1)$. The difference is an additional use of the inductive hypothesis to evaluate e_2, which contains the length of the array to be output, additional handling of the constant pointer to the array data and reading the entire array, and Lemma 5.2.74 to reason about the use of $\mathrm{OutputArray}$ in place of Lemma 5.2.73 to reason about the use of $\mathrm{OutputValue}$. The handling of reading the array is similar to what is shown in Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::(\mathrm{p},[ra])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_i) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, \mathrm{smcoutput}(x, e_1, e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,0),\ldots,(l_1,\alpha-1)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[out3])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, \mathrm{smcoutput}(x, e_1, e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,0),\ldots,(l_1,\alpha-1)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[out1])} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[rp])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_1, \mu_1)) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[rp])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_1, \mu_1)) \parallel C)$ by SMC^2 rule Pointer Read Single Location, we have (B) $\gamma(x) = (l,\ a\ bty*)$, (C) $\sigma(l) = (\omega,\ a\ bty*,\ 1, \mathrm{PermL}(\mathrm{Freeable}, a\ bty*, a, 1))$, and (D) $\mathrm{DecodePtr}(a\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i]$.

Given (E) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C})$ and ψ such that (F) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C})$, by Definition 5.2.22 we have (G) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (H) $C \cong_\psi \widehat{C}$, and (I) $x \cong_\psi \widehat{x}$. Given (I), by Definition 5.2.20 we have (J) $x = \widehat{x}$.

Given (B), (G), and (J), by Lemma 5.2.62 we have (K) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that (L) $l = \widehat{l}$ and (M) $a\ bty* \cong_\psi \widehat{bty}*$.

Given (C), (G), and (L), by Lemma 5.2.64 we have (N) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}*, \mathrm{public}, 1))$ such that (O) $\omega \cong_\psi \widehat{\omega}$.

Given (D), (M), and (O), by Lemma 5.2.11 we have (P) $\mathrm{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$ such that (Q) $[1,\ [(l_1, \mu_1)],\ [1],\ i] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$. Given (Q), by Definition 5.2.15 we have (R) $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$.

Given (E), (K), (N), and (P), by Vanilla C rule Pointer Read Location we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p}, [\widehat{rp}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}_1, \widehat{\mu}_1)) \parallel \widehat{C})$.

Given (G), (R), and (H), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_1, \mu_1)) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}_1, \widehat{\mu}_1)) \parallel \widehat{C})$.

By Definition 5.2.23 we have $rp \cong \hat{rp}$, and by Definition 5.2.25 we have $(\mathrm{p}, [rp]) \cong (\mathrm{p}, [\widehat{rp}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p}, [(l,0)])}_{(\mathrm{p}, [rp1])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [\alpha,\ \bar{l},\ \bar{j},\ i]) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p}, [(l,0)])}_{(\mathrm{p}, [rp1])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [\alpha,\ \bar{l},\ \bar{j},\ i]) \parallel C)$ by SMC^2 rule Private Pointer Read Multiple Locations, we have (B) $\gamma(x) = (l,\ \mathrm{private}\ bty*)$, (C) $\sigma(l) = (\omega,\ \mathrm{private}\ bty*,\ \alpha,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty*, \mathrm{private}, \alpha))$, (D) $(bty = \mathrm{int}) \vee (bty = \mathrm{float})$, and (E) $\mathrm{DecodePtr}(\mathrm{private}\ bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]$.

Given (F) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C})$ and ψ such that (G) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C})$, by Definition 5.2.22 we have (H) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (I) $C \cong_\psi \widehat{C}$, and (J) $x \cong_\psi \widehat{x}$. Given (J), by Definition 5.2.20 we have (K) $x = \widehat{x}$.

Given (B), (H), and (K), by Lemma 5.2.62 we have (L) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that (M) $l = \widehat{l}$ and (N) $\mathrm{private}\ bty* \cong_\psi \widehat{bty*}$.

Given (C), (H), and (M), by Lemma 5.2.64 we have (O) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty*}, \mathrm{public}, 1))$ such that (P) $\omega \cong_\psi \widehat{\omega}$.

Given (D), (N), and (P), by Lemma 5.2.11 we have (Q) $\mathrm{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$ such that (R) $[\alpha,\ \bar{l},\ \bar{j},\ i] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$. Given (R), by Lemma 5.2.75 we have (S) $[\alpha,\ \bar{l},\ \bar{j},\ i] \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$.

Given (F), (L), (O), and (Q), by Vanilla C rule Pointer Read Location we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p}, [\widehat{rp}])}$

$((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l_1}, \widehat{\mu}_1)) \parallel \widehat{C})$.

Given (H), (S), and (I), by Lemma 5.2.75 we have $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [\alpha,\ \bar{l},\ \bar{j},\ i]) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l_1}, \widehat{\mu}_1))$ $\parallel \widehat{C})$.

By Definition 5.2.23 we have $rp1 \cong \hat{rp}$, and by Definition 5.2.25 we have $(\mathrm{p}, [rp1]) \cong (\mathrm{p}, [\widehat{rp}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$ by SMC^2 rule Pointer Dereference Single Location, we have (B) $\gamma(x) = (l,\ a\ bty*)$, (C) $\sigma(l) = (\omega,\ a\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, a\ bty*, a, 1))$, (D) $\mathrm{DecodePtr}(a\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1]$, and (E) $\mathrm{DerefPtr}(\sigma, a\ bty, (l_1, \mu_1)) = (n, 1)$.

Given (F) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel \widehat{C})$ and ψ such that (G) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel \widehat{C})$, by Definition 5.2.22 we have (H) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (I) $C \cong_\psi \widehat{C}$, and (J) $*x \cong_\psi *\widehat{x}$. Given (J), by Definition 5.2.20 we have (K) $x = \widehat{x}$.

Given (B), (H), and (K), by Lemma 5.2.62 we have (L) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that (M) $l = \widehat{l}$ and (N) $a\ bty* \cong_\psi \widehat{bty}*$.

Given (C), (H), and (M), by Lemma 5.2.63 we have (O) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}*, \mathrm{public}, 1))$ such that (P) $\omega \cong_\psi \widehat{\omega}$.

Given (D), (N), and (P), by Lemma 5.2.11 we have (Q) $\mathrm{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l_1}, \widehat{\mu}_1)], [1], 1]$ (R) $[1,\ [(l_1, \mu_1)],$ $[1],\ 1] \cong_\psi [1, [(\widehat{l_1}, \widehat{\mu}_1)], [1], 1]$. Given (R), by Definition 5.2.15 we have (S) $(l_1, \mu_1) \cong_\psi (\widehat{l_1}, \widehat{\mu}_1)$.

Given (E), (H), (N), and (S), by Lemma 5.2.25 we have (T) $\mathrm{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l_1}, \widehat{\mu}_1)) = (\widehat{n}, 1)$ such that (U) $n \cong_\psi \widehat{n}$.

Given (F), (L), (O), (Q), and (T), by Vanilla C rule Pointer Dereference we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{rdp}])}$ $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}) \parallel \widehat{C})$.

Given (H), (U), and (I), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}) \parallel \widehat{C})$.

By Definition 5.2.23 we have $rdp \cong \hat{rdp}$, and by Definition 5.2.25 we have $(\mathrm{p}, [rdp]) \cong (\mathrm{p}, [\widehat{rdp}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp1])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp1])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$ by SMC^2 rule Pointer Dereference Single Location Higher Level Indirection, we have (B) $\gamma(x) = (l,\ a\ bty*)$, (C) $\sigma(l) = (\omega_1,\ a\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, a\ bty*, a, 1))$, (D) $\mathrm{DecodePtr}(a\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i]$, (E) $\mathrm{DerefPtrHLI}(\sigma, a\ bty*, (l_1, \mu_1)) = ([1,\ [(l_2, \mu_2)],\ [1],\ i-1], 1)$, and (F) $i > 1$.

Given (G) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel \widehat{C})$ and ψ such that (H) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel \widehat{C})$, by Definition 5.2.22 we have (I) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (J) $C \cong_\psi \widehat{C}$, and (K) $*x \cong_\psi *\widehat{x}$. Given (K), by Definition 5.2.20 we have (L) $x = \widehat{x}$.

Given (B), (I), and (L), by Lemma 5.2.62 we have (M) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that (N) $l = \widehat{l}$ and (O) $a\ bty* \cong_\psi \widehat{bty}*$.

Given (C), (I), and (N), by Lemma 5.2.63 we have (P) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}*, \mathrm{public}, 1))$ such that (Q) $\omega \cong_\psi \widehat{\omega}$.

Given (D), (O), and (Q), by Lemma 5.2.11 we have (R) $\mathrm{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$ such that (S) $[1,\ [(l_1, \mu_1)],\ [1],\ i] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$. Given (S), by Definition 5.2.15 we have (T) $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$ and (U) $i = \widehat{i}$.

Given (F) and (U), we have (V) $\widehat{i} > 1$.

Given (E), (I), (O), and (T), by Lemma 5.2.26 we have (W) $\mathrm{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty}*, (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i} - 1], 1)$ such that (X) $[1,\ [(l_2, \mu_2)],\ [1],\ i-1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i} - 1]$. Given (X), by Definition 5.2.15 we have (Y) $(l_2, \mu_2) \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$.

Given (G), (M), (P), (R), (V), and (W), by Vanilla C rule Pointer Dereference Higher Level Indirection we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p},[\widehat{rdp1}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}_2, \widehat{\mu}_2)) \parallel \widehat{C})$.

Given (I), (Y), and (J), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}_2, \widehat{\mu}_2))$-

$\| \widehat{C})$.

By Definition 5.2.23 we have $rdp1 \cong \hat{rdp1}$, and by Definition 5.2.25 we have $(\mathrm{p}, [rdp1]) \cong (\mathrm{p}, [\widehat{rdp1}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp2])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [\alpha,\ \bar{l},\ \bar{j},\ i-1]) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp2])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [\alpha,\ \bar{l},\ \bar{j},\ i-1]) \parallel C)$ by SMC^2 rule Pointer Dereference Single Location Higher Level Indirection, we have (B) $\gamma(x) = (l,\ \text{private}\ bty*)$, (C) $\sigma(l) = (\omega_1,\ \text{private}\ bty*,\ 1,\ \mathrm{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, 1))$, (D) $\mathrm{DecodePtr}(\text{private}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i]$, (E) $\mathrm{DerefPtrHLI}(\sigma, \text{private}\ bty*, (l_1, \mu_1)) = ([\alpha,\ \bar{l},\ \bar{j},\ i-1], 1)$, and (F) $i > 1$.

Given (G) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, *\widehat{x}) \parallel \widehat{C})$ and ψ such that (H) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, *\widehat{x}) \parallel \widehat{C})$, by Definition 5.2.22 we have (I) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (J) $C \cong_\psi \widehat{C}$, and (K) $*x \cong_\psi *\widehat{x}$. Given (K), by Definition 5.2.20 we have (L) $x = \widehat{x}$.

Given (B), (I), and (L), by Lemma 5.2.62 we have (M) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that (N) $l = \widehat{l}$ and (O) $\text{private}\ bty* \cong_\psi \widehat{bty}*$.

Given (C), (I), and (N), by Lemma 5.2.63 we have (P) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \mathrm{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that (Q) $\omega \cong_\psi \widehat{\omega}$.

Given (D), (O), and (Q), by Lemma 5.2.11 we have (R) $\mathrm{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$ such that (S) $[1,\ [(l_1, \mu_1)],\ [1],\ i] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$. Given (S), by Definition 5.2.15 we have (T) $(l_1, \mu_1) \cong_\psi (\widehat{l}_1, \widehat{\mu}_1)$ and (U) $i = \widehat{i}$.

Given (F) and (U), we have (V) $\widehat{i} > 1$.

Given (E), (I), (O), and (T), by Lemma 5.2.26 we have (W) $\mathrm{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty}*, (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i} - 1], 1)$ such that (X) $[\alpha,\ \bar{l},\ \bar{j},\ i-1] \cong_\psi [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i} - 1]$. Given (X), by Lemma 5.2.75 we have (Y) $[\alpha,\ \bar{l},\ \bar{j},\ i - 1] \cong_\psi (\widehat{l}_2, \widehat{\mu}_2)$.

Given (G), (M), (P), (R), (V), and (W), by Vanilla C rule Pointer Dereference Higher Level Indirection we have $\Sigma \triangleright$

$((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel \widehat{C}) \Downarrow'_{(\mathrm{p}, [\widehat{rdp1}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}_2, \widehat{\mu}_2)) \parallel \widehat{C})$.

Given (I), (Y), and (J), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [\alpha,\ \bar{l},\ \bar{j},\ i-1]) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}_2, \widehat{\mu}_2)) \parallel \widehat{C})$.

By Definition 5.2.23 we have $rdp2 \cong \hat{rdp1}$, and by Definition 5.2.25 we have $(\mathrm{p}, [rdp2]) \cong (\mathrm{p}, [\widehat{rdp1}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[wp1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[wp1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Private Pointer Write, we have (B) $(e) \nvdash \gamma$, (C) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ (l_e, \mu_e)) \parallel C_1)$, (D) $\gamma(x) = (l,\ \text{private } bty*)$, (E) $\sigma_1(l) = (\omega,\ \text{private } bty*,\ \alpha, \mathrm{PermL}(\mathrm{Freeable}, \text{private } bty*, \text{private}, \alpha))$, (F) $\mathrm{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]$, and (G) $\mathrm{UpdatePtr}(\sigma_1,\ (l, 0),\ [1,\ [(l_e, \mu_e)],\ [1],\ i],\ \text{private } bty*) = (\sigma_2,\ 1)$.

Given (H) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x} = \widehat{e}) \parallel \widehat{C})$ and ψ such that (I) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x} = \widehat{e}) \parallel \widehat{C})$, by Definition 5.2.22 we have (J) $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, (K) $C \cong_{\psi} \widehat{C}$, and (L) $x = e \cong_{\psi} \widehat{x} = \widehat{e}$. Given (M), by Definition 5.2.20 we have (M) $e \cong_{\psi} \widehat{e}$ and (N) $x = \widehat{x}$.

Given (J), (M), and (K), by Lemma 5.2.50 we have (O) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C})$. Given (C) and (O), by the inductive hypothesis we have (P) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_e, \widehat{\mu}_e)) \parallel \widehat{C}_1)$ and ψ_1 such that (Q) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ (l_e, \mu_e)) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_e, \widehat{\mu}_e)) \parallel \widehat{C}_1)$ and (R) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (Q), by Definition 5.2.22 we have (S) $(\gamma,\ \sigma_1) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)$, (T) $(l_e, \mu_e) \cong_{\psi_1} (\widehat{l}_e, \widehat{\mu}_e)$, and (U) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (D), (S), and (N), by Lemma 5.2.62 we have (V) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that (W) $l = \widehat{l}$ and (X) $\text{private } bty* \cong_{\psi_1} \widehat{bty}*$.

Given (E), (S), and (W), by Lemma 5.2.63 we have (Y) $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}*, \mathrm{public}, 1))$ such that (Z) $\omega \cong_{\psi_1} \widehat{\omega}$.

Given (F), (X), and (Z), by Lemma 5.2.11 we have (A1) $\mathrm{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$ such that (B1) $[\alpha,\ \bar{l},\ \bar{j},\ i] \cong_{\psi_1} [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$. Given (B1), by Definition 5.2.15 we have (C1) $i = \widehat{i}$. Given (T) and (C1), by

Definition 5.2.15 we have (D1) $[1,\ [(l_e, \mu_e)],\ [1],\ i] \cong_{\psi_1} 1, [(\widehat{l_e}, \widehat{\mu_e})], [1], \widehat{i}]$.

Given (G), (S), (W), (D1), and (X), by Lemma 5.2.17 we have (E1) $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}, 0), [1, [(\widehat{l_e}, \widehat{\mu_e})], [1], \widehat{i}], \widehat{bty*}) = (\widehat{\sigma}_2, 1)$ such that (F1) $(\gamma, \sigma_2) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given (H), (P), (V), (Y), (A1), and (E1), by Vanilla C rule Pointer Write Location we have $\Sigma \triangleright ((\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x} = \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{wp}])} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \text{skip}) \parallel \widehat{C}_1)$.

Given (F1) and (U), by Definition 5.2.22 we have $((\text{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1) \cong_{\psi_1} ((\text{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \text{skip}) \parallel \widehat{C}_1)$. By Definition 5.2.23 we have $wp1 \cong \widehat{wp}$. Given (R), $\mathcal{D}_1 :: (\text{p}, [wp1])$ and $\widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{wp}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\text{p}, [wp1]) \cong \widehat{\mathcal{D}}_1 :: (\text{p}, [\widehat{wp}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0)])}_{\mathcal{D}_1 :: (\text{p}, [wp])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0)])}_{\mathcal{D}_1 :: (\text{p}, [wp1])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0)])}_{\mathcal{D}_1 :: (\text{p}, [wp2])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0)])}_{\mathcal{D}_1 :: (\text{p}, [wp1])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0)]) :: \bar{l}_1 :: [(l_1, \mu_1)])}_{\mathcal{D}_1 :: (\text{p}, [wdp3])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0)]) :: \bar{l}_1 :: [(l_1, \mu_1)])}_{\mathcal{D}_1 :: (\text{p}, [wdp3])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{skip}) \parallel C_1)$ by SMC^2 rule Private Pointer Dereference Write Single Location Private Value, we have (B) $(e) \vdash \gamma$, (C) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e)$-$\parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \text{acc},\ n) \parallel C_1)$, (D) $\gamma(x) = (l,\ \text{private}\ bty*)$, (E) $\sigma_1(l) = (\omega,\ \text{private}\ bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, 1))$, (F) $(bty = \text{int}) \vee (bty = \text{float})$, (G) $\text{DecodePtr}(\text{private}\ bty*, 1, \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1]$, (H) $\text{DynamicUpdate}(\Delta_1, \sigma_1, [(l_1, \mu_1)], \text{acc}, \text{private}\ bty) = (\Delta_2, \bar{l}_1)$, and (I) $\text{UpdateOffset}(\sigma_1,\ (l_1, \mu_1),\ n, \text{priva}$ $(\sigma_2, 1)$.

Given (J) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e}) \parallel \widehat{C})$ and ψ such that (K) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e}) \parallel \widehat{C})$ by Definition 5.2.22 we have (L) $(\gamma,\ \sigma) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})$, (M) $C \cong_{\psi} \widehat{C}$, and (N) $*x = e \cong_{\psi} *\widehat{x} = \widehat{e}$. Given (N), by Definition 5.2.20 we have (O) $e \cong_{\psi} \widehat{e}$ and (P) $x = \widehat{x}$.

Given (L), (O), and (M), by Lemma 5.2.50 we have (Q) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \cong_{\psi} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C})$. Given (C) and (Q), by the inductive hypothesis we have (R) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$ and ψ_1 such that (S) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}) \parallel \widehat{C}_1)$ and (T) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (S), by Definition 5.2.22 we have (U) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$, (V) $n \cong_{\psi_1} \widehat{n}$ and (W) $C_1 \cong_{\psi_1} \widehat{C}_1$.

Given (D), (U), and (P), by Lemma 5.2.62 we have (X) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that (Y) $l = \widehat{l}$ and (Z) private $bty* \cong_{\psi_1} \widehat{bty}*$.

Given (E), (U), and (Y), by Lemma 5.2.63 we have (A1) $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}*, \mathrm{public}, 1))$ such that (B1) $\omega \cong_{\psi_1} \widehat{\omega}$.

Given (G), (Z), and (B1), by Lemma 5.2.11 we have (C1) $\mathrm{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$ such that (D1) $[1,\ [(l_1, \mu_1)],\ [1],\ 1] \cong_{\psi_1} [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$. Given (D1), by Definition 5.2.15 we have (E1) $(l_1, \mu_1) \cong_{\psi_1} (\widehat{l}_1, \widehat{\mu}_1)$.

Given (Z), by Definition 5.2.8 we have (F1) private $bty \cong_{\psi_1} \widehat{bty}$.

Given (I), (U), (E1), (V), and (F1), by Lemma 5.2.27 we have (G1) $\mathrm{UpdateOffset}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), \widehat{n}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$ such that (H1) $(\gamma, \sigma_2) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given (J), (R), (X), (A1), (C1), and (G1), by Vanilla C rule Pointer Dereference Write Value we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{wdp}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_1)$.

Given (H1) and (W), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}) \parallel \widehat{C}_1)$. By Definition 5.2.23 we have $wdp3 \cong \hat{wdp}$. Given (T), $\mathcal{D}_1 :: (\mathrm{p}, [wdp3])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{wdp}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [wdp3]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{wdp}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0), (l_1, \mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp3])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip})$ $\parallel C_1)$

Case $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp4])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1)$.

This case is similar to Case $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp3])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip})$ $\parallel C_1)$. Given $n = \widehat{n}$, we use Definition 5.2.19 to prove that $\mathrm{encrypt}(n) \cong \widehat{n}$.

Case $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Private Pointer Dereference Write Multiple Locations to Single Location Higher Level Indirection, we have (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ [\alpha, \bar{l}_e, \bar{j}_e, i-1]) \parallel C_1)$, (C) $\gamma(x) = (l,\ \mathrm{private}\ bty*)$, (D) $\sigma_1(l) = (\omega,\ \mathrm{private}\ bty*,\ 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty*, \mathrm{private}, 1))$, (E) $\mathrm{DecodePtr}(\mathrm{private}\ bty*,\ 1,\ \omega)$ $= [1,\ [(l_1, \mu_1)],\ [1],\ i]$, (F) $i > 1$, (G) $\mathrm{DynamicUpdate}(\Delta_1, \sigma_1, [(l_1, \mu_1)],\ \mathrm{acc}, \mathrm{private}\ bty*) = (\Delta_2, \bar{l}_1)$, and (H) $\mathrm{UpdatePtr}(\sigma_1,\ (l_1, \mu_1),\ [\alpha, \bar{l}_e, \bar{j}_e, i-1], \mathrm{private}\ bty*) = (\sigma_2, 1)$.

Given (I) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e}) \parallel \widehat{C})$ and ψ such that (J) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square,$ $*\widehat{x} = \widehat{e}) \parallel \widehat{C})$ by Definition 5.2.22 we have (K) $(\gamma,\ \sigma) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})$, (L) $C \cong_\psi \widehat{C}$, and (M) $*x = e \cong_\psi *\widehat{x} = \widehat{e}$. Given (M), by Definition 5.2.20 we have (N) $e \cong_\psi \widehat{e}$ and (O) $x = \widehat{x}$.

Given (K), (N), and (L), by Lemma 5.2.50 we have (P) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \cong_\psi ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C})$ Given (B) and (P), by the inductive hypothesis we have (Q) $((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_e, \widehat{\mu}_e)) \parallel \widehat{C}_1)$ and ψ_1 such that (R) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ [\alpha, \bar{l}_e, \bar{j}_e, i-1]) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_e, \widehat{\mu}_e)) \parallel \widehat{C}_1)$ and (S) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (R), by Definition 5.2.22 we have (T) $(\gamma, \sigma_1) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)$, (U) $[\alpha, \bar{l}_e, \bar{j}_e, i-1] \cong_{\psi_1} (\widehat{l}_e, \widehat{\mu}_e)$ and (V) $C_1 \cong_{\psi_1} \widehat{C}_1$. Given (U), by Definition 5.2.15 we have (W) $[\alpha, \bar{l}_e, \bar{j}_e, i-1] \cong_{\psi_1} [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1]$

Given (C), (T), and (O), by Lemma 5.2.62 we have (X) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that (Y) $l = \widehat{l}$ and (Z) $\mathrm{private}\ bty* \cong_{\psi_1} \widehat{bty*}$.

Given (D), (T), and (Y), by Lemma 5.2.63 we have (A1) $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}*, \mathrm{public}, 1))$ such that (B1) $\omega \cong_{\psi_1} \widehat{\omega}$.

Given (E), (Z), and (B1), by Lemma 5.2.11 we have (C1) $\mathrm{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$ such that (D1) $[1,\ [(l_1, \mu_1)],\ [1],\ i] \cong_{\psi_1} [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$. Given (D1), by Definition 5.2.15 we have (E1) $(l_1, \mu_1) \cong_{\psi_1} (\widehat{l}_1, \widehat{\mu}_1)$ and (F1) $i = \widehat{i}$.

Given (F) and (F1), by (G1) $\widehat{i} > 1$.

Given (H), (T), (E1), (W), and (Z), by Lemma 5.2.17 we have (H1) $\mathrm{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i} - 1], \widehat{bty}*) = (\widehat{\sigma}_2, 1)$ such that (I1) $(\gamma, \sigma_2) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_2)$.

Given (I), (Q), (X), (A1), (C1), (G1), and (H1), by Vanilla C rule Pointer Dereference Write Higher Level Indirection we have $\Sigma \triangleright ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, *\widehat{x} = \widehat{e}) \parallel \widehat{C}) \Downarrow'_{\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{wdp1}])} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \mathrm{skip}) \parallel \widehat{C}_1)$.

Given (I1) and (V), by Definition 5.2.22 we have $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1) \cong_{\psi_1} ((\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \Box, \Box, \mathrm{skip}) \parallel \widehat{C}_1)$. By Definition 5.2.23 we have $wdp2 \cong \hat{wdp1}$. Given (S), $\mathcal{D}_1 :: (\mathrm{p}, [wdp2])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{wdp1}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\mathrm{p}, [wdp2]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{p}, [\widehat{wdp1}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)] :: \bar{l}_1 :: [(l_1, \mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp5])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)] :: \bar{l}_1 :: [(l_1, \mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0), (l_1, \mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)] :: \bar{l}_1 :: [(l_1, \mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1)$.

Case $\Pi\triangleright ((1,\gamma^1,\sigma^1,\Delta^1,\text{acc},x[e]) \ \| \ ... \ \| \ (\text{q},\gamma^{\text{q}},\sigma^{\text{q}},\Delta^{\text{q}},\text{acc},x[e])) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_I::(\text{ALL},[mpra])} ((1,\gamma^1,\sigma^1_1,\Delta^1_1,\text{acc},n^1) \ \| \ ... \ \| \ (\text{q},\gamma^{\text{q}},\sigma^{\text{q}}_1,\Delta^{\text{q}}_1,\text{acc},n^{\text{q}}))$

Given (A) $\Pi\triangleright ((1,\gamma^1,\sigma^1,\Delta^1,\text{acc},x[e]) \ \| \ ... \ \| \ (\text{q},\gamma^{\text{q}},\sigma^{\text{q}},\Delta^{\text{q}},\text{acc},x[e])) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_I::(\text{ALL},[mpra])} ((1,\gamma^1,\sigma^1_1,\Delta^1_1,\text{acc},n^1) \ \| \ ... \ \| \ (\text{q},\gamma^{\text{q}},\sigma^{\text{q}}_1,\Delta^{\text{q}}_1,\text{acc},n^{\text{q}}))$ by SMC^2 rule Multiparty Array Read Private Index, we have (B) $\{(e) \vdash \gamma^{\text{p}}\}^{\text{q}}_{\text{p}=1}$, (C) $\{(n^{\text{p}}) \vdash \gamma^{\text{p}}\}^{\text{q}}_{\text{p}=1}$, (D) $((1,\gamma^1,\sigma^1,\Delta^1,\text{acc},e) \ \| \ ... \ \| \ (\text{q},\gamma^{\text{q}},\sigma^{\text{q}},\Delta^{\text{q}},\text{acc},e)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_I} ((1,\gamma^1,\sigma^1_1,\Delta^1_1,\text{acc},i^1) \ \| \ ... \ \| \ (\text{q},\gamma^{\text{q}},\sigma^{\text{q}}_1,\Delta^{\text{q}}_1,\text{acc},i^{\text{q}}))$, (E) $\{\gamma^{\text{p}}(x) = (l^{\text{p}},\text{const } a \ bty*)\}^{\text{q}}_{\text{p}=1}$, (F) $\{\sigma^{\text{p}}_1(l^{\text{p}}) = (\omega^{\text{p}}, \ a \text{ const } bty*, 1, \text{PermL}(\text{Freeable}, a \text{ const } bty*, a, 1))\}^{\text{q}}_{\text{p}=1}$, (G) $\{\text{DecodePtr}(a \text{ const } bty*, \ 1, \ \omega^{\text{p}}) = [1, \ [(l^{\text{p}}_1, 0)], \ [1], \ 1]\}^{\text{q}}_{\text{p}=1}$, (H) $\{\sigma^{\text{p}}_1(l^{\text{p}}_1) = (\omega^{\text{p}}_1, \ a \ bty, \ \alpha, \ \text{PermL}(\text{Freeable}, a \ bty, a, \alpha))\}^{\text{q}}_{\text{p}=1}$, (I) $\{\forall i \in \{0...\alpha-1\}\text{DecodeArr}(a \ bty, i, \omega^{\text{p}}_1) = n^{\text{p}}_i\}^{\text{q}}_{\text{p}=1}$, (J) $\text{MPC}_{ar}((i^1, [n^1_0, ..., n^1_{\alpha-1}]), ..., (i^{\text{q}}, [n^{\text{q}}_0, ..., n^{\text{q}}_{\alpha-1}])) = (n^1, ..., n^{\text{q}})$, and $\mathcal{L}_2 = (1, [(l^1, 0), (l^1_1, 0), ..., (l^1_1, \alpha-1)]) \ \| \ ... \ \| \ (\text{q}, [(l^{\text{q}}, 0), (l^{\text{q}}_1, 0), ..., (l^{\text{q}}_1, \alpha-1)])$.

Given (A), $((1,\widehat{\gamma}^1, \ \widehat{\sigma}^1, \ \square,\square, \ \widehat{x}[\widehat{e}]) \ \| \ ... \ \| \ (\text{q},\widehat{\gamma}^{\text{q}}, \ \widehat{\sigma}^{\text{q}}, \ \square,\square, \ \widehat{x}[\widehat{e}]))$ and ψ such that $\{(\text{p},\gamma^{\text{p}},\sigma^{\text{p}},\Delta^{\text{p}},\text{acc}, \ x[e]) \cong_\psi (\text{p},\widehat{\gamma}^{\text{p}}, \ \widehat{\sigma}^{\text{p}}, \ \square,\square, \ \widehat{x}[\widehat{e}])\}^{\text{q}}_{\text{p}=1}$, by Definition 5.2.22 we have $\{(\gamma^{\text{p}}, \ \sigma^{\text{p}}) \cong_\psi (\widehat{\gamma}^{\text{p}}, \ \widehat{\sigma}^{\text{p}})\}^{\text{q}}_{\text{p}=1}$ and (K) $x[e] \cong_\psi \widehat{x}[\widehat{e}]$. By Definition 5.2.20 we have $x \cong_\psi \widehat{x}$ such that (L) $x = \widehat{x}$ and (M) $e \cong_\psi \widehat{e}$.

Given Axiom 5.2.1, by Theorem 5.2.2 we have $\{(1,\gamma^1,\sigma^1,\Delta^1,\text{acc},x[e]) \sim (\text{p}, \ \gamma^{\text{p}},\sigma^{\text{p}},\Delta^{\text{p}}, \ \text{acc},x[e])\}^{\text{q}}_{\text{p}=1}$. By Lemma 5.2.86, we have $\{(\text{p},\gamma^{\text{p}},\sigma^{\text{p}},\Delta^{\text{p}},\text{acc},x[e]) \cong_\psi (\text{p},\widehat{\gamma}, \ \widehat{\sigma}, \ \square,\square, \ \widehat{x}[\widehat{e}])\}^{\text{q}}_{\text{p}=1}$. and therefore (N) $((1,\widehat{\gamma}, \ \widehat{\sigma}, \ \square,\square, \ \widehat{x}[\widehat{e}]) \ \| \ ... \ \| \ (\text{q},\widehat{\gamma}, \ \widehat{\sigma}, \ \square,\square, \ \widehat{x}[\widehat{e}]))$. By Definition 5.2.22 we have (O) $\{(\gamma^{\text{p}}, \ \sigma^{\text{p}}) \cong_\psi (\widehat{\gamma}, \ \widehat{\sigma})\}^{\text{q}}_{\text{p}=1}$.

Given (D), (M), (O), and ψ, by Lemma 5.2.76 we have (P) $((1,\widehat{\gamma}, \ \widehat{\sigma}, \ \square,\square, \ \widehat{e}) \ \| \ ... \ \| \ (\text{q},\widehat{\gamma}, \ \widehat{\sigma}, \ \square,\square, \ \widehat{e}))$ such that (Q) $\{(\text{p},\gamma^{\text{p}},\sigma^{\text{p}},\Delta^{\text{p}},\text{acc},e) \cong_\psi (\text{p},\widehat{\gamma}, \ \widehat{\sigma}, \ \square,\square, \ \widehat{e})\}^{\text{q}}_{\text{p}=1}$. Given (P) and (Q), by the inductive hypothesis, we have (R) $((1,\widehat{\gamma}, \ \widehat{\sigma}, \ \square,\square, \ \widehat{e}) \ \| \ ... \ \| \ (\text{q},\widehat{\gamma}, \ \widehat{\sigma}, \ \square,\square, \ \widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}_1} ((1,\widehat{\gamma}, \ \widehat{\sigma}_1, \ \square,\square, \ \widehat{i}) \ \| \ ... \ \| \ (\text{q},\widehat{\gamma}, \ \widehat{\sigma}_1, \ \square,\square, \ \widehat{i}))$ and ψ_1 such that (S) $\{(\text{p},\gamma^{\text{p}},\sigma^{\text{p}}_1,\Delta^{\text{p}}_1,\text{acc},i^{\text{p}}) \cong_{\psi_1} (\text{p},\widehat{\gamma}, \ \widehat{\sigma}_1, \ \square,\square, \ \widehat{i})\}^{\text{q}}_{\text{p}=1}$ and (T) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (S), by Definition 5.2.22 we have (U) $\{(\gamma^{\text{p}}, \ \sigma^{\text{p}}_1) \cong_{\psi_1} (\widehat{\gamma}, \ \widehat{\sigma}_1)\}^{\text{q}}_{\text{p}=1}$ and (V) $\{i^{\text{p}} \cong_{\psi_1} \widehat{i}\}^{\text{q}}_{\text{p}=1}$.

Given (E), (U), and (L), by Lemma 5.2.77 we have (W) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \text{const } \widehat{bty*})$ such that (X) $\{l^{\text{p}} = \widehat{l}\}^{\text{q}}_{\text{p}=1}$ and (Y) a const $bty* \cong_\psi$ const $\widehat{bty*}$. By Definition 5.2.8 we have $bty = \widehat{bty}$ and therefore (Z) $a bty \cong_\psi \widehat{bty}$.

Given (F), (U), and (X), by Lemma 5.2.78 we have (A1) $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \text{const } \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \text{const } \widehat{bty*}, \text{public}, 1))$ such that (B1) $\{\omega^{\text{p}} \cong_\psi 1\widehat{\omega}\}^{\text{q}}_{\text{p}=1}$.

Given (G), (Y), and (B1), by Lemma 5.2.12 we have (C1) $\text{DecodePtr}(\text{const } \widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ such that

(D1) $\{l_1^{\mathrm{p}} = \widehat{l}_1\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (H), (U), and (D1), by Lemma 5.2.78 we have (E1) $\widehat{\sigma}_1(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, \widehat{\alpha}))$ such that (F1) $\{\omega_1^{\mathrm{p}} \cong_{\psi} 1\widehat{\omega}_1\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (G1) $\alpha = \widehat{\alpha}$.

Given (V), and (G1), by Axiom 5.2.1 we have (H1) $0 \leq \widehat{i} \leq \widehat{\alpha} - 1$.

Given (I), (Z), (V), and (F1), by Lemma 5.2.9 we have (I1) $\mathrm{DecodeArr}(\widehat{bty},\ \widehat{i},\ \widehat{\omega}_1) = \widehat{n}_{\widehat{i}}$ such that (J1) $\{n_i^{\mathrm{p}} \cong_{\psi_1} \widehat{n}_{\widehat{i}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (J), (J1), (H1), (G1), and (V), by Axiom 5.2.8 we have (K1) $\{n^{\mathrm{p}} \cong_{\psi_1} \widehat{n}_{\widehat{i}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (N), (R), (W), (A1), (C1), (E1), (H1), and (I1), by Vanilla C rule Multiparty Array Read we have $\Sigma \triangleright ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}]) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}])) \Downarrow'_{\widehat{\mathcal{D}}_1::(\mathrm{ALL},[\widehat{mpra}])} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}_{\widehat{i}}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}_{\widehat{i}}))$.

Given (U) and (K1), by Definition 5.2.22 we have $((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}})) \cong_{\psi_1} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}_{\widehat{i}}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}_{\widehat{i}}))$.
By Definition 5.2.23 we have $mpra \cong \widehat{mpra}$.
Given (T), $\mathcal{D}_1 :: (\mathrm{ALL}, [mpra])$ and $\widehat{\mathcal{D}}_1 :: (\mathrm{ALL}, [\widehat{mpra}])$, by Lemma 5.2.58 we have
$\mathcal{D}_1 :: (\mathrm{ALL}, [mpra]) \cong \widehat{\mathcal{D}}_1 :: (\mathrm{ALL}, [\widehat{mpra}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, x[e_1] = e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, x[e_1] = e_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpwa])} ((1, \gamma^1, \sigma_{3+\alpha-1}^1, \Delta_2^1, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_{3+\alpha-1}^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, x[e_1] = e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, x[e_1] = e_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpwa])} ((1, \gamma^1, \sigma_{3+\alpha-1}^1, \Delta_2^1, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_{3+\alpha-1}^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip}))$ by SMC^2 rule Multiparty Array Write Private Index, we have (B) $\{(e_1) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (C) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, i^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, i^{\mathrm{q}}))$, (D) $((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}} \mathrm{acc}, e_2)) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}}))$, (E) $\{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \mathrm{private\ const}\ bty*)\}_{\mathrm{p}=1}^{\mathrm{q}}$, (F) $\{\sigma_2^{\mathrm{p}}(l^{\mathrm{p}}) = (\omega^{\mathrm{p}}, \mathrm{private\ const}\ bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private\ const}\ bty*, \mathrm{private}, 1))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (G) $\{\mathrm{DecodePtr}(\mathrm{private\ const}\ bty*, 1,\ \omega^{\mathrm{p}}) = [1,\ [(l_1^{\mathrm{p}},\ 0)],\ [1],\ 1]\}_{\mathrm{p}=1}^{\mathrm{q}}$, (H) $\{\sigma_2^{\mathrm{p}}(l_1^{\mathrm{p}}) = (\omega_1^{\mathrm{p}}, \mathrm{private}\ bty, \alpha,\ \mathrm{PermL}(\mathrm{Freeable},\ \mathrm{private}\ bty,\ \mathrm{private},$

$\alpha))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (I) $\{\forall j \in \{0...\alpha-1\}$ DecodeArr(private $bty, j, \omega_1^{\mathrm{p}}) = n_j^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (J) $\mathrm{MPC}_{aw}((i^1, n^1, [n_0^1, ..., n_{\alpha-1}^1]), ..., (i^{\mathrm{q}}, n^{\mathrm{q}}, [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}])) = ([n_0'^1, ..., n_{\alpha-1}'^1], ..., [n_0'^{\mathrm{q}}, ..., n_{\alpha-1}'^{\mathrm{q}}])$, (K) $\{\forall j \in \{0...\alpha-1\}$ UpdateArr$(\sigma_{2+j}^{\mathrm{p}}, (l_1^{\mathrm{p}}, j), n_j'^{\mathrm{p}}$, private $bty) = \sigma_{3+j}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and $\mathcal{L}_3 = (1, [(l^{\mathrm{p}}, 0), (l_1^{\mathrm{p}}, 0), ..., (l_1^{\mathrm{p}}, \alpha-1)]) \parallel ... \parallel (\mathrm{q}, [(l^{\mathrm{p}}, 0), (l_1^{\mathrm{p}}, 0), ..., (l_1^{\mathrm{p}}, \alpha-1)])$.

Given (A), $((1, \widehat{\gamma}^1, \widehat{\sigma}^1, \square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}^{\mathrm{q}}, \widehat{\sigma}^{\mathrm{q}}, \square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2))$ and ψ such that $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, x[e_1] = e_2) \cong_\psi (\mathrm{p}, \widehat{\gamma}^{\mathrm{p}}, \widehat{\sigma}^{\mathrm{p}}, \square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$, by Definition 5.2.22 we have $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \cong_\psi (\widehat{\gamma}^{\mathrm{p}}, \widehat{\sigma}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (L) $x[e_1] = e_2 \cong_\psi \widehat{x}[\widehat{e}_1] = \widehat{e}_2$. By Definition 5.2.20 we have $x \cong_\psi \widehat{x}$ such that (M) $x = \widehat{x}$, (N) $e_1 \cong_\psi \widehat{e}_1$, and (O) $e_2 \cong_\psi \widehat{e}_2$.

Given Axiom 5.2.1, by Theorem 5.2.2 we have $\{(1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, x[e_1] = e_2) \sim (\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, x[e_1] = e_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$. By Lemma 5.2.86, we have $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, x[e_1] = e_2) \cong_\psi (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$. and therefore (P) $((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2))$. By Definition 5.2.22 we have (Q) $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (C), (Q), (N), and ψ, by Lemma 5.2.76 we have (R) $((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}_1) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}_1))$ such that (S) $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, e_1) \cong_\psi (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (R) and (S), by the inductive hypothesis, we have (T) $((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}_1) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}_1)) \Downarrow'_{\widehat{\mathcal{D}}_1} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{i}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{i}))$ and ψ_1 such that (U) $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, \Delta_1^{\mathrm{p}}, \mathrm{acc}, i^{\mathrm{p}}) \cong_{\psi_1} (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{i})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (V) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (U), by Definition 5.2.22 we have (W) $\{(\gamma^{\mathrm{p}}, \sigma_1^{\mathrm{p}}) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (X) $\{i^{\mathrm{p}} \cong_{\psi_1} \widehat{i}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given Axiom 5.2.1, we have $(l, \mu) \notin e_2$. Given (O), by Lemma 5.2.55 we have (Y) $e_2 \cong_{\psi_1} \widehat{e}_2$.

Given (D), (W), (Y), and ψ_1, by Lemma 5.2.76 we have (Z) $((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}_2))$ such that (A1) $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, e_2) \cong_{\psi_1} (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (Z) and (A1), by the inductive hypothesis, we have (B1) $((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{e}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_2} ((1, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \widehat{n}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \widehat{n}))$ and ψ_2 such that (C1) $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma_2^{\mathrm{p}}, \Delta_2^{\mathrm{p}}, \mathrm{acc}, n^{\mathrm{p}}) \cong_{\psi_2} (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \widehat{n})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (D1) $\mathcal{D}_2 \cong \widehat{\mathcal{D}}_2$. Given (C1), by Definition 5.2.22 we have (E1) $\{(\gamma^{\mathrm{p}}, \sigma_2^{\mathrm{p}}) \cong_{\psi_2} (\widehat{\gamma}, \widehat{\sigma}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (F1) $\{n^{\mathrm{p}} \cong_{\psi_2} \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (E), (E1), and (M), by Lemma 5.2.77 we have (G1) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \mathrm{const}\ \widehat{bty*})$ such that (H1) $\{l^{\mathrm{p}} = \widehat{l}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (I1) private const $bty* \cong_{\psi_2}$ const $\widehat{bty*}$. By Definition 5.2.8 we have (J1) private $bty \cong_{\psi_2} \widehat{bty}$.

Given (F), (E1), and (H1), by Lemma 5.2.78 we have (K1) $\widehat{\sigma}_2(\widehat{l}) = (\widehat{\omega}, \mathrm{const}\ \widehat{bty*}, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{const}\ \widehat{bty*},$

public, 1)) such that (L1) $\{\omega^{\mathrm{p}} \cong_{\psi_2} \widehat{\omega}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (G), (I1), and (L1), by Lemma 5.2.12 we have (M1) $\mathrm{DecodePtr}(\mathrm{const}\ \widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], 1]$ such that (N1) $\{l_1^{\mathrm{p}} = \widehat{l}_1\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (H), (E1), and (N1), by Lemma 5.2.78 we have (O1) $\widehat{\sigma}_2(\widehat{l}_1) = (\widehat{\omega}_1, \widehat{bty}, \widehat{\alpha}, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty}, \mathrm{public}, \widehat{\alpha}))$ such that (P1) $\{\omega_1^{\mathrm{p}} \cong_{\psi_2} \widehat{\omega}_1\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (Q1) $\alpha = \widehat{\alpha}$.

Given (C) and (H), by Axiom 5.2.1, we have (R1) $\{0 \leq i^{\mathrm{p}} \leq \alpha - 1\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (R1), (X), and (Q1), we have (S1) $0 \leq \hat{i} \leq \widehat{\alpha} - 1$.

Given (J), (S1), (Q1), (X), and (F1), by Axiom 5.2.9 we have (T1) $\{n_{\widehat{i}}'^{\mathrm{p}} \cong_{\psi_2} \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (U1) $\{\forall j \neq \widehat{i} \in \{0...\alpha-1\} n_j^{\mathrm{p}} = n_j'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (K), (E1), (N1), (Q1), (P1), (I), (T1), (U1), and (J1), by Lemma 5.2.80 we have (V1) $\mathrm{UpdateArr}(\widehat{\sigma}_2,\ (\widehat{l}_1, \widehat{i}),\ \widehat{n},\ \widehat{bty}) = \widehat{\sigma}_3$ such that (W1) $\{\sigma_{3+\alpha-1}^{\mathrm{p}} \cong_{\psi_2} \widehat{\sigma}_3\}_{\mathrm{p}=1}^{\mathrm{q}}$

Given (P), (T), (B1), (G1), (K1), (M1), (O1), (S1), and (V1), by Vanilla C rule Multiparty Array Write we have $\Sigma \triangleright ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{x}[\widehat{e}_1] = \widehat{e}_2)) \Downarrow'_{\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{ALL}, [\widehat{mpwa}])} ((1, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}))$.

Given (W1), by Definition 5.2.22 we have $((1, \gamma^1, \sigma_{3+\alpha-1}^1, \Delta_2^1, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_{3+\alpha-1}^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip})) \cong_{\psi_2} ((1, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_3, \square, \square, \mathrm{skip}))$.
By Definition 5.2.23 we have $mpwa \cong \widehat{mpwa}$.
Given (V), (D1), $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{ALL}, [mpwa])$ and $\widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{ALL}, [\widehat{mpwa}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{ALL}, [mpwa]) \cong \widehat{\mathcal{D}}_1 :: \widehat{\mathcal{D}}_2 :: (\mathrm{ALL}, [\widehat{mpwa}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_2} \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, ++\ x) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, ++\ x)) \Downarrow_{(\mathrm{ALL}, [mppin])}^{(1, [(l^1, 0)]) \parallel ... \parallel (\mathrm{q}, [(l^{\mathrm{q}}, 0)])} ((1, \gamma^1, \sigma_1^1, \Delta^1, \mathrm{acc}, n_2^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, n_2^{\mathrm{q}}))$

Given (A) $\Pi\triangleright ((1,\gamma^1,\sigma^1,\Delta^1,\text{acc},{+}{+}\ x)\ \|\ ...\ \|\ (\text{q},\gamma^{\text{q}},\sigma^{\text{q}},\Delta^{\text{q}},\text{acc},{+}{+}\ x)) \Downarrow^{(1,[(l^1,0)])\ \|\ ...\ \|\ (\text{q},[(l^{\text{q}},0)])}_{(\text{ALL},[mppin])} ((1,\gamma^1,\sigma^1_1,\Delta^1,\text{acc},n^1_2)\ \|\ ...\ \|\ (\text{q},\gamma^{\text{q}},\sigma^{\text{q}}_1,\Delta^{\text{q}},\text{acc},n^{\text{q}}_2))$ by SMC^2 rule Multiparty Pre-Increment Private Float Variable, we have (B) $\{\gamma^{\text{p}}(x) = (l^{\text{p}},\text{private float})\}^{\text{q}}_{\text{p}=1}$, (C) $\{\sigma^{\text{p}}(l^{\text{p}}) = (\omega^{\text{p}},\text{private float},1,\ \text{PermL}(\text{Freeable},\text{private float},\text{private},1))\}^{\text{q}}_{\text{p}=1}$, (D) $\{(x)\vdash\gamma^{\text{p}}\}^{\text{q}}_{\text{p}=1}$, (E) $\{\text{DecodeVal}(\text{private float},\omega^{\text{p}}) = n^{\text{p}}_1\}^{\text{q}}_{\text{p}=1}$, (F) $\text{MPC}_u({+}{+},n^1_1,...,n^{\text{q}}_1) = (n^1_2,...,n^{\text{q}}_2)$, and (G) $\{\text{UpdateVal}(\sigma^{\text{p}},l^{\text{p}},n^{\text{p}}_2,\text{private float}) = \sigma^{\text{p}}_1\}^{\text{q}}_{\text{p}=1}$.

Given (A), $((1,\widehat{\gamma}^1,\ \widehat{\sigma}^1,\ \Box,\Box,\ {+}{+}\ \widehat{x})\ \|\ ...\ \|\ (\text{q},\widehat{\gamma}^{\text{q}},\ \widehat{\sigma}^{\text{q}},\ \Box,\Box,\ {+}{+}\ \widehat{x}))$ and ψ such that $\{(\text{p},\gamma^{\text{p}},\sigma^{\text{p}},\Delta^{\text{p}},\text{acc},{+}{+}\ x)\cong_\psi(\text{p},\widehat{\gamma}^{\text{p}},\ \widehat{\sigma}^{\text{p}},\ \Box,\Box,\ {+}{+}\ \widehat{x})\}^{\text{q}}_{\text{p}=1}$, by Definition 5.2.22 we have $\{(\gamma^{\text{p}},\ \sigma^{\text{p}})\cong_\psi(\widehat{\gamma}^{\text{p}},\ \widehat{\sigma}^{\text{p}})\}^{\text{q}}_{\text{p}=1}$ and (H) ${+}{+}\ x\cong_\psi{+}{+}\ \widehat{x}$. By Definition 5.2.20 we have $x\cong_\psi\widehat{x}$ such that (I) $x=\widehat{x}$.

Given Axiom 5.2.1, by Theorem 5.2.2 we have $\{(1,\gamma^1,\sigma^1,\Delta^1,\text{acc},{+}{+}\ x)\sim(\text{p},\gamma^{\text{p}},\sigma^{\text{p}},\Delta^{\text{p}},\text{acc},{+}{+}\ x)\}^{\text{q}}_{\text{p}=1}$. By Lemma 5.2.86, we have $\{(\text{p},\gamma^{\text{p}},\sigma^{\text{p}},\Delta^{\text{p}},\text{acc},{+}{+}\ x)\cong_\psi(\text{p},\widehat{\gamma},\ \widehat{\sigma},\ \Box,\Box,\ {+}{+}\ \widehat{x})\}^{\text{q}}_{\text{p}=1}$. and therefore (J) $((1,\widehat{\gamma},\ \widehat{\sigma},\ \Box,\Box,{+}{+}\ \widehat{x})\ \|\ ...\ \|\ (\text{q},\widehat{\gamma},\ \widehat{\sigma},\ \Box,\Box,\ {+}{+}\ \widehat{x}))$. By Definition 5.2.22 we have (K) $\{(\gamma^{\text{p}},\ \sigma^{\text{p}})\cong_\psi(\widehat{\gamma},\ \widehat{\sigma})\}^{\text{q}}_{\text{p}=1}$.

Given (B), (K), and (I), by Lemma 5.2.77 we have (L) $\widehat{\gamma}(\widehat{x}) = (\widehat{l},\widehat{bty})$ such that (M) $\{l^{\text{p}}\cong_\psi\widehat{l}\}^{\text{q}}_{\text{p}=1}$ and (N) $\text{private float}\cong_\psi\widehat{bty}$.

Given (C), (K), and (M), by Lemma 5.2.78 we have (O) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega},\widehat{bty},1,\text{PermL}(\text{Freeable},\widehat{bty},\text{public},1))$ such that (P) $\{\omega^{\text{p}}\cong_\psi\widehat{\omega}\}^{\text{q}}_{\text{p}=1}$.

Given (E), (N), and (P), by Lemma 5.2.8 we have (Q) $\text{DecodeVal}(\widehat{bty},\widehat{\omega}) = \widehat{n}_1$ such that (R) $\{n^{\text{p}}_1\cong_\psi\widehat{n}_1\}^{\text{q}}_{\text{p}=1}$.

Given (F) and (R), by Axiom 5.2.10 we have (S) $\widehat{n}_2 = \widehat{n}_1+1$ such that (T) $\{n^{\text{p}}_2\cong\widehat{n}_2\}^{\text{q}}_{\text{p}=1}$.

Given (G), (K), (M), (T), and (N), by Lemma 5.2.14 we have (U) $\text{UpdateVal}(\widehat{\sigma},\widehat{l},\widehat{n}_2,\widehat{bty}) = \widehat{\sigma}_1$ such that (V) $\{(\gamma^{\text{p}},\ \sigma^{\text{p}}_1)\cong_\psi(\widehat{\gamma},\ \widehat{\sigma}_1)\}^{\text{q}}_{\text{p}=1}$.

Given (J), (L), (O), (Q), (S), and (U), by Vanilla C rule Multiparty Pre-Increment Variable we have $\Sigma\triangleright((1,\widehat{\gamma},\widehat{\sigma},\ \Box,\Box,{+}{+}\ \widehat{x})\ \|\ ...\ \|\ (\text{q},\widehat{\gamma},\widehat{\sigma},\ \Box,\Box,{+}{+}\ \widehat{x}))\Downarrow'_{(\text{ALL},[\widehat{mppin}])}((1,\widehat{\gamma},\widehat{\sigma}_1,\Box,\Box,\widehat{n}_2)\ \|\ ...\ \|\ (\text{q},\widehat{\gamma},\widehat{\sigma}_1,\Box,\Box,\widehat{n}_2))$.

Given (V) and (T), by Definition 5.2.22 we have $((1,\gamma^1,\sigma^1_1,\Delta^1,\text{acc},n^1_2)\ \|\ ...\ \|\ (\text{q},\gamma^{\text{q}},\sigma^{\text{q}}_1,\Delta^{\text{q}},\text{acc},n^{\text{q}}_2))\cong_\psi((1,\widehat{\gamma},\widehat{\sigma}_1,\Box,\Box,\widehat{n}_2)\ \|\ ...\ \|\ (\text{q},\widehat{\gamma},\widehat{\sigma}_1,\Box,\Box,\widehat{n}_2))$.
By Definition 5.2.23 we have $mppin\cong\widehat{mppin}$. by Definition 5.2.25 we have $(\text{ALL},[mppin])\cong(\text{ALL},[\widehat{mppin}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_\psi \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, *x) \parallel ... \parallel (\text{q}, \gamma^\text{q}, \sigma^\text{q}, \Delta^\text{q}, \text{acc}, *x)) \Downarrow^{(1,(l^1,0)::\bar{l}^1) \parallel ... \parallel (\text{q},(l^\text{q},0)::\bar{l}^\text{q})}_{(\text{ALL},[mprdp])} ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, n^1) \parallel ... \parallel (\text{q}, \gamma^\text{q}, \sigma^\text{q}, \Delta^\text{q}, \text{acc}, n^\text{q}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, *x) \parallel ... \parallel (\text{q}, \gamma^\text{q}, \sigma^\text{q}, \Delta^\text{q}, \text{acc}, *x)) \Downarrow^{(1,(l^1,0)::\bar{l}^1) \parallel ... \parallel (\text{q},(l^\text{q},0)::\bar{l}^\text{q})}_{(\text{ALL},[mprdp])} ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, n^1) \parallel ... \parallel (\text{q}, \gamma^\text{q}, \sigma^\text{q}, \Delta^\text{q}, \text{acc}, n^\text{q}))$ by SMC^2 rule Multiparty Private Pointer Dereference Single Level Indirection, we have (B) $\{(x) \vdash \gamma^\text{p}\}^\text{q}_{\text{p}=1}$, (C) $\{\gamma^\text{p}(x) = (l^\text{p}, \text{private } bty*)\}^\text{q}_{\text{p}=1}$, (D) $\{\sigma^\text{p}(l^\text{p}) = (\omega^\text{p}, \text{ private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))\}^\text{q}_{\text{p}=1}$, (E) $\alpha > 1$, (F) $\{\text{DecodePtr}(\text{private } bty*, \alpha, \omega^\text{p}) = [\alpha, \bar{l}^\text{p}, \bar{j}^\text{p}, 1]\}^\text{q}_{\text{p}=1}$, (G) $\{\text{Retrieve_vals}(\alpha, \bar{l}^\text{p}, \text{private } bty, \sigma^\text{p}) = ([n^\text{p}_0, ...n^\text{p}_{\alpha-1}], 1)\}^\text{q}_{\text{p}=1}$, and (H) $\text{MPC}_{dv}([[n^1_0, ..., n^1_{\alpha-1}], ..., [n^\text{q}_0, ..., n^\text{q}_{\alpha-1}]], [\bar{j}^1, ..., \bar{j}^\text{q}]) = (n^1, ..., n^\text{q})$.

Given (A), $((1, \widehat{\gamma}^1, \widehat{\sigma}^1, \square, \square, *\widehat{x}) \parallel ... \parallel (\text{q}, \widehat{\gamma}^\text{q}, \widehat{\sigma}^\text{q}, \square, \square, *\widehat{x}))$ and ψ such that $\{(\text{p}, \gamma^\text{p}, \sigma^\text{p}, \Delta^\text{p}, \text{acc}, *x) \cong_\psi (\text{p}, \widehat{\gamma}^\text{p}, \widehat{\sigma}^\text{p}, \square, \square, *\widehat{x})\}^\text{q}_{\text{p}=1}$, by Definition 5.2.22 we have $\{(\gamma^\text{p}, \sigma^\text{p}) \cong_\psi (\widehat{\gamma}^\text{p}, \widehat{\sigma}^\text{p})\}^\text{q}_{\text{p}=1}$ and (I) $*x \cong_\psi *\widehat{x}$. By Definition 5.2.20 we have $x \cong_\psi \widehat{x}$ such that (J) $x = \widehat{x}$.

Given Axiom 5.2.1, by Theorem 5.2.2 we have $\{(1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, *x) \sim (\text{p}, \gamma^\text{p}, \sigma^\text{p}, \Delta^\text{p}, \text{acc}, *x)\}^\text{q}_{\text{p}=1}$. By Lemma 5.2.86, we have $\{(\text{p}, \gamma^\text{p}, \sigma^\text{p}, \Delta^\text{p}, \text{acc}, *x) \cong_\psi (\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x})\}^\text{q}_{\text{p}=1}$. and therefore (K) $((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel ... \parallel (\text{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}))$. By Definition 5.2.22 we have (L) $\{(\gamma^\text{p}, \sigma^\text{p}) \cong_\psi (\widehat{\gamma}, \widehat{\sigma})\}^\text{q}_{\text{p}=1}$.

Given (C), (L), and (J), by Lemma 5.2.77 we have (M) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty}*)$ such that (N) $\{l^\text{p} = \widehat{l}\}^\text{q}_{\text{p}=1}$ and (O) $\text{private } bty* \cong_\psi \widehat{bty}*$.

Given (D), (L), and (N), by Lemma 5.2.78 we have (P) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that (Q) $\{\omega^\text{p} \cong_\psi \widehat{\omega}\}^\text{q}_{\text{p}=1}$.

Given (F), (O), and (Q), by Lemma 5.2.11 we have (R) $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$ such that (S) $\{[\alpha, \bar{l}^\text{p}, \bar{j}^\text{p}, 1] \cong_\psi [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]\}^\text{q}_{\text{p}=1}$.

Given (O), by Definition 5.2.8 we have (T) $\text{private } bty \cong_\psi \widehat{bty}$.

Given (G), (H), (S), (T), and (L), by Lemma 5.2.81 we have (U) $\text{DerefPtr}(\widehat{\sigma}, \widehat{bty}, (\widehat{l}_1, \widehat{\mu}_1)) = (\widehat{n}, 1)$ such that (V)

$\{n^{\mathrm{p}} \cong \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (K), (M), (P), (R), and (U), by Vanilla C rule Multiparty Pointer Dereference we have $\Sigma \triangleright ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x})) \Downarrow'_{(\mathrm{ALL}, [\widehat{mprdp}])} ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}))$.

Given (L) and (V), by Definition 5.2.22 we have $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}})) \cong_{\psi} ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{n}))$.
By Definition 5.2.23 we have $mprdp \cong \widehat{mprdp}$. by Definition 5.2.25 we have $(\mathrm{ALL}, [mprdp]) \cong (\mathrm{ALL}, [\widehat{mprdp}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, *x)) \Downarrow^{(1,(l^1,0)::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^{\mathrm{q}},0)::\bar{l}^{\mathrm{q}})}_{(\mathrm{ALL},[mprdp1])} ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, [\alpha_\alpha, \bar{l}^1_\alpha, \bar{j}^1_\alpha, i-1]) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, [\alpha_\alpha, \bar{l}^{\mathrm{q}}_\alpha, \bar{j}^{\mathrm{q}}_\alpha, i-1]))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, *x)) \Downarrow^{(1,(l^1,0)::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^{\mathrm{q}},0)::\bar{l}^{\mathrm{q}})}_{(\mathrm{ALL},[mprdp1])} ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, [\alpha_\alpha, \bar{l}^1_\alpha, \bar{j}^1_\alpha, i-1]) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, [\alpha_\alpha, \bar{l}^{\mathrm{q}}_\alpha, \bar{j}^{\mathrm{q}}_\alpha, i-1]))$ by SMC^2 rule Multiparty Private Pointer Dereference Higher Level Indirection, we have (B) $\{(x) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (C) $\{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \text{private } bty*)\}_{\mathrm{p}=1}^{\mathrm{q}}$, (D) $\{\sigma^{\mathrm{p}}(l^{\mathrm{p}}) = (\omega^{\mathrm{p}}, \text{ private } bty*, \alpha, \mathrm{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (E) $\alpha > 1$,
(F) $\{\mathrm{DecodePtr}(\text{private } bty*, \alpha, \omega^{\mathrm{p}}) = [\alpha, \bar{l}^{\mathrm{p}}, \bar{j}^{\mathrm{p}}, i]\}_{\mathrm{p}=1}^{\mathrm{q}}$, (G) $i > 1$, (H) $\{\mathrm{Retrieve_vals}(\alpha, \bar{l}^{\mathrm{p}}, \text{ private } bty*, \sigma^{\mathrm{p}}) = ([[\alpha_0, \bar{l}^{\mathrm{p}}_0, \bar{j}^{\mathrm{p}}_0, i-1], ..., [\alpha_{\alpha-1}, \bar{l}^{\mathrm{p}}_{\alpha-1}, \bar{j}^{\mathrm{p}}_{\alpha-1}, i-1]], 1)\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (I) $\mathrm{MPC}_{dp}([[[\alpha_0, \bar{l}^1_0, \bar{j}^1_0], ..., [\alpha_{\alpha-1}, \bar{l}^1_{\alpha-1}, \bar{j}^1_{\alpha-1}]], ..., [[\alpha_0, \bar{l}^{\mathrm{q}}_0, \bar{j}^{\mathrm{q}}_0], ..., [\alpha_{\alpha-1}, \bar{l}^{\mathrm{q}}_{\alpha-1}, \bar{j}^{\mathrm{q}}_{\alpha-1}]]], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) = ([[\alpha_\alpha, \bar{l}^1_\alpha, \bar{j}^1_\alpha], ..., [\alpha_\alpha, \bar{l}^{\mathrm{q}}_\alpha, \bar{j}^{\mathrm{q}}_\alpha]])$.

Given (A), $((1, \widehat{\gamma}^1, \widehat{\sigma}^1, \square, \square, *\widehat{x}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}^{\mathrm{q}}, \widehat{\sigma}^{\mathrm{q}}, \square, \square, *\widehat{x}))$ and ψ such that $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, *x) \cong_{\psi} (\mathrm{p}, \widehat{\gamma}^{\mathrm{p}}, \widehat{\sigma}^{\mathrm{p}}, \square, \square, *\widehat{x})\}_{\mathrm{p}=1}^{\mathrm{q}}$, by Definition 5.2.22 we have $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}^{\mathrm{p}}, \widehat{\sigma}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (J) $*x \cong_{\psi} *\widehat{x}$. By Definition 5.2.20 we have $x \cong_{\psi} \widehat{x}$ such that (K) $x = \widehat{x}$.

Given Axiom 5.2.1, by Theorem 5.2.2 we have $\{(1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x) \sim (\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, *x)\}_{\mathrm{p}=1}^{\mathrm{q}}$. By Lemma 5.2.86, we have $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, *x) \cong_{\psi} (\mathrm{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x})\}_{\mathrm{p}=1}^{\mathrm{q}}$, and therefore (L) $((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel ... \parallel (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}))$. By Definition 5.2.22 we have (M) $\{(\gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (C), (M), and (K), by Lemma 5.2.77 we have (N) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that (O) $\{l^{\mathrm{p}} = \widehat{l}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (P) private $bty* \cong_{\psi} \widehat{bty*}$.

Given (D), (M), and (O), by Lemma 5.2.78 we have (Q) $\widehat{\sigma}(\widehat{l}) = (\widehat{\omega}, \widehat{bty}*, 1, \text{PermL}(\text{Freeable}, \widehat{bty}*, \text{public}, 1))$ such that (R) $\{\omega^{\text{p}} \cong_{\psi} \widehat{\omega}\}^{\text{q}}_{\text{p}=1}$.

Given (F), (P), and (R), by Lemma 5.2.11 we have (S) $\text{DecodePtr}(\widehat{bty}*, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$ such that (T) $\{[\alpha,\ \bar{l}^{\text{p}},\ \bar{j}^{\text{p}}, i] \cong_{\psi} [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]\}^{\text{q}}_{\text{p}=1}$. Given (T), by Definition 5.2.15 we have (U) $i = \widehat{i}$.

Given (G) and (U), we have (V) $\widehat{i} > 1$.

Given (H), (I), (T), (P), and (M), by Lemma 5.2.82 we have (W) $\text{DerefPtrHLI}(\widehat{\sigma}, \widehat{bty}*, (\widehat{l}_1, \widehat{\mu}_1)) = ([1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i} - 1], 1)$ such that (X) $\{[\alpha_{\alpha},\ \bar{l}^{\text{q}}_{\alpha},\ \bar{j}^{\text{q}}_{\alpha}, \widehat{i} - 1] \cong_{\psi} [1, [(\widehat{l}_2, \widehat{\mu}_2)], [1], \widehat{i} - 1]\}^{\text{q}}_{\text{p}=1}$.

Given (X), by Lemma 5.2.75 we have (Y) $\{[\alpha_{\alpha},\ \bar{l}^{\text{q}}_{\alpha},\ \bar{j}^{\text{q}}_{\alpha}, \widehat{i} - 1] \cong_{\psi} (\widehat{l}_2, \widehat{\mu}_2)$.

Given (L), (N), (Q), (S), (V), and (W), by Vanilla C rule Multiparty Pointer Dereference Higher Level Indirection we have $\Sigma \triangleright ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x}) \parallel ... \parallel (\text{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x})) \Downarrow'_{(\text{ALL}, [\widehat{mprdp1}])} ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}_2, \widehat{\mu}_2)) \parallel ... \parallel (\text{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}_2, \widehat{\mu}_2)))$.

Given (M) and (Y), by Definition 5.2.22 we have $((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, [\alpha_{\alpha}, \bar{l}^1_{\alpha}, \bar{j}^1_{\alpha}, i - 1]) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, [\alpha_{\alpha}, \bar{l}^{\text{q}}_{\alpha}, \bar{j}^{\text{q}}_{\alpha}, i - 1])) \cong_{\psi} ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}_2, \widehat{\mu}_2)) \parallel ... \parallel (\text{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, (\widehat{l}_2, \widehat{\mu}_2)))$.
By Definition 5.2.23 we have $mprdp1 \cong \widehat{mprdp1}$. by Definition 5.2.25 we have $(\text{ALL}, [mprdp1]) \cong (\text{ALL}, [\widehat{mprdp1}])$. Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi} \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc},\ *x = e) \parallel ... \parallel (\text{q}, \gamma^{\text{q}},\ \sigma^{\text{q}},\ \Delta^{\text{q}},\ \text{acc},\ *x = e))$
$\Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}^1_1::\bar{l}^1) \parallel ... \parallel (\text{q},(l^{\text{q}},0)::\bar{l}^{\text{q}}_1::\bar{l}^{\text{q}})}_{\mathcal{D}_1::(\text{ALL},[mpwdp3])} ((1, \gamma^1,\ \sigma^1_2,\ \Delta^1_2,\ \text{acc},\ \text{skip}) \parallel ... \parallel (\text{q}, \gamma^{\text{q}},\ \sigma^{\text{q}}_2,\ \Delta^{\text{q}}_2,\ \text{acc},\ \text{skip}))$

Given (A) $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc},\ *x = e) \parallel ... \parallel (\text{q}, \gamma^{\text{q}},\ \sigma^{\text{q}},\ \Delta^{\text{q}},\ \text{acc},\ *x = e))$
$\Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}^1_1::\bar{l}^1) \parallel ... \parallel (\text{q},(l^{\text{q}},0)::\bar{l}^{\text{q}}_1::\bar{l}^{\text{q}})}_{\mathcal{D}_1::(\text{ALL},[mpwdp3])} ((1, \gamma^1,\ \sigma^1_2,\ \Delta^1_2,\ \text{acc},\ \text{skip}) \parallel ... \parallel (\text{q}, \gamma^{\text{q}},\ \sigma^{\text{q}}_2,\ \Delta^{\text{q}}_2,\ \text{acc},\ \text{skip}))$ by SMC^2 rule Multiparty Private Pointer Dereference Write Private Value, we have (B) $\{(e) \vdash \gamma^{\text{p}}\}^{\text{q}}_{\text{p}=1}$, (C) $((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc},\ e) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, e)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((1, \gamma^1, \sigma^1_1, \Delta^1_1, \text{acc}, n^1) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}_1, \Delta^{\text{q}}_1, \text{acc}, n^{\text{q}}))$, (D) $\{\gamma^{\text{p}}(x) = (l^{\text{p}}, \text{private } bty*)\}^{\text{q}}_{\text{p}=1}$, (E) $\{\sigma^{\text{p}}_1(l^{\text{p}}) = (\omega^{\text{p}},\ \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))\}^{\text{q}}_{\text{p}=1}$, (F) $\alpha > 1$, (G) $\{\text{DecodePtr}(\text{private } bty*, \alpha, \omega^{\text{p}}) = [\alpha,\ \bar{l}^{\text{p}},\ \bar{j}^{\text{p}}, 1]\}^{\text{q}}_{\text{p}=1}$, (H) $\{\text{DynamicUpdate}(\Delta^{\text{p}}_1, \sigma^{\text{p}}_1, \bar{l}^{\text{p}},\ \text{acc}, \text{private } bty) = (\Delta^{\text{p}}_2, \bar{l}^{\text{p}}_1)\}^{\text{q}}_{\text{p}=1}$, (I) $\{\text{Retrieve_vals}(\alpha,\ \bar{l}^{\text{p}}, \text{private } bty, \sigma^{\text{p}}_1) = ([n^{\text{p}}_0, ...n^{\text{p}}_{\alpha-1}], 1)\}^{\text{q}}_{\text{p}=1}$, (J) $\text{MPC}_{wdv}([[n^1_0,\ ...,\ n^1_{\alpha-1}],$

$..., [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}]], [n^1, ..., n^{\mathrm{q}}], [\bar{j}^1, ..., \bar{j}^{\mathrm{q}}]) = ([n_0'^1, ..., n_{\alpha-1}'^1], ..., [n_0'^{\mathrm{q}}, ..., n_{\alpha-1}'^{\mathrm{q}}])$, and
(K) $\{\mathrm{UpdateDerefVals}(\alpha, \bar{l}^{\mathrm{p}}, [n_0'^{\mathrm{p}}, ..., n_{\alpha-1}'^{\mathrm{p}}], \mathrm{private}\ bty, \sigma_1^{\mathrm{p}}) = \sigma_2^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (A), $((1, \widehat{\gamma}^1,\ \widehat{\sigma}^1,\ \square, \square,\ *\widehat{x} = \widehat{e})\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma}^{\mathrm{q}},\ \widehat{\sigma}^{\mathrm{q}},\ \square, \square,\ *\widehat{x} = \widehat{e}))$ and ψ such that $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, *x = e) \cong_\psi (\mathrm{p}, \widehat{\gamma}^{\mathrm{p}},\ \widehat{\sigma}^{\mathrm{p}},\ \square, \square,\ *\widehat{x} = \widehat{e})\}_{\mathrm{p}=1}^{\mathrm{q}}$, by Definition 5.2.22 we have $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}) \cong_\psi (\widehat{\gamma}^{\mathrm{p}},\ \widehat{\sigma}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (L) $*x = e \cong_\psi *\widehat{x} = \widehat{e}$. By Definition 5.2.20 we have $x \cong_\psi \widehat{x}$ such that (M) $x = \widehat{x}$ and (N) $e \cong_\psi \widehat{e}$.

Given Axiom 5.2.1, by Theorem 5.2.2 we have $\{(1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x = e) \sim (\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, *x = e)\}_{\mathrm{p}=1}^{\mathrm{q}}$. By Lemma 5.2.86, we have $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, *x = e) \cong_\psi (\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma},\ \square, \square,\ *\widehat{x} = \widehat{e})\}_{\mathrm{p}=1}^{\mathrm{q}}$. and therefore (O) $((1, \widehat{\gamma},\ \widehat{\sigma},\ \square, \square,\ *\widehat{x} = \widehat{e})\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma},\ \widehat{\sigma},\ \square, \square,\ *\widehat{x} = \widehat{e}))$. By Definition 5.2.22 we have (P) $\{(\gamma^{\mathrm{p}},\ \sigma^{\mathrm{p}}) \cong_\psi (\widehat{\gamma},\ \widehat{\sigma})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (C), (P), (N), and ψ, by Lemma 5.2.76 we have (Q) $((1, \widehat{\gamma},\ \widehat{\sigma},\ \square, \square,\ \widehat{e})\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma},\ \widehat{\sigma},\ \square, \square,\ \widehat{e}))$ such that (R) $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}, \Delta^{\mathrm{p}}, \mathrm{acc}, e) \cong_\psi (\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma},\ \square, \square,\ \widehat{e})\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (Q) and (R), by the inductive hypothesis, we have (S) $((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e})\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n})\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, \widehat{n}))$ and ψ_1 such that (T) $\{(\mathrm{p}, \gamma^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, \Delta_1^{\mathrm{p}}, \mathrm{acc}, n^{\mathrm{p}}) \cong_{\psi_1} (\mathrm{p}, \widehat{\gamma},\ \widehat{\sigma}_1,\ \square, \square,\ \widehat{n})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (U) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (T), by Definition 5.2.22 we have (V) $\{(\gamma^{\mathrm{p}},\ \sigma_1^{\mathrm{p}}) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (W) $\{n^{\mathrm{p}} \cong_{\psi_1} \widehat{n}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (D), (V), and (M), by Lemma 5.2.77 we have (X) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that (Y) $\{l^{\mathrm{p}} = \widehat{l}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (Z) $\mathrm{private}\ bty* \cong_{\psi_1} \widehat{bty*}$. By Definition 5.2.8 we have (A1) $\mathrm{private}\ bty \cong_{\psi_1} \widehat{bty}$.

Given (E), (V), and (Y), by Lemma 5.2.78 we have (B1) $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \mathrm{PermL}(\mathrm{Freeable}, \widehat{bty*}, \mathrm{public}, 1))$ such that (C1) $\{\omega^{\mathrm{p}} \cong_{\psi_1} \widehat{\omega}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (G), (Z), and (C1), by Lemma 5.2.11 we have (D1) $\mathrm{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]$ such that (E1) $\{[\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}}, 1] \cong_{\psi_1} [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], 1]\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (I), (J), (K), (E1), (W), (A1), and (V), by Lemma 5.2.83 we have (F1) $\mathrm{UpdateOffset}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), \widehat{n}, \widehat{bty}) = (\widehat{\sigma}_2, 1)$ such that (G1) $\{(\gamma^{\mathrm{p}},\ \sigma_2^{\mathrm{p}}) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_2)\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (O), (S), (X), (B1), (D1), and (F1), by Vanilla C rule Multiparty Pointer Dereference Write Value we have $\Sigma \triangleright ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e})\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}::(\mathrm{ALL}, [\widehat{mpwdp}])} ((1, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip})\ \|\ ...\ \|\ (\mathrm{q}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \mathrm{skip}))$.

Given (G1), by Definition 5.2.22 we have $((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \text{acc},\ \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}},\ \sigma_2^{\text{q}},\ \Delta_2^{\text{q}},\ \text{acc},\ \text{skip})) \cong_{\psi_1}$ $((1, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \text{skip})\ \|\ ...\ \|\ (\text{q}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \text{skip}))$.

By Definition 5.2.23 we have $mpwdp3 \cong \widehat{mpwdp}$. Given (U), $\mathcal{D}_1 :: (\text{ALL}, [mpwdp3])$ and $\widehat{\mathcal{D}}_1 :: (\text{ALL}, [\widehat{mpwdp}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\text{ALL}, [mpwdp3]) \cong \widehat{\mathcal{D}}_1 :: (\text{ALL}, [\widehat{mpwdp}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc}, *x = e)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, *x = e))$
$\Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}_1^1::\bar{l}^1)\ \|\ ...\ \|\ (\text{q},(l^{\text{q}},0)::\bar{l}_1^{\text{q}}::\bar{l}^{\text{q}})}_{\mathcal{D}_1::(\text{ALL},[mpwdp])} ((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \text{acc},\ \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_2^{\text{q}}, \Delta_2^{\text{q}}, \text{acc}, \text{skip}))$

This case is similar to Case $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc}, *x = e)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, *x = e))$ $\Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}_1^1::\bar{l}^1)\ \|\ ...\ \|\ (\text{q},(l^{\text{q}},0)::\bar{l}_1^{\text{q}}::\bar{l}^{\text{q}})}_{\mathcal{D}_1::(\text{ALL},[mpwdp3])} ((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \text{acc},\ \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_2^{\text{q}}, \Delta_2^{\text{q}}, \text{acc}, \text{skip}))$. Given $\{n^{\text{p}} = \widehat{n}\}_{\text{p}=1}^{\text{q}}$, we use Definition 5.2.19 to prove that $\{\text{encrypt}(n^{\text{p}}) \cong \widehat{n}\}_{\text{p}=1}^{\text{q}}$.

Case $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc},\ *x = e)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, *x = e))$
$\Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}_1^1::\bar{l}^1)\ \|\ ...\ \|\ (\text{q},(l^{\text{q}},0)::\bar{l}_1^{\text{q}}::\bar{l}^{\text{q}})}_{\mathcal{D}_1::(\text{ALL},[mpwdp2])} ((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \text{acc},\ \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_2^{\text{q}}, \Delta_2^{\text{q}}, \text{acc}, \text{skip}))$

Given (A) $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc},\ *x = e)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, *x = e))$ $\Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}_1^1::\bar{l}^1)\ \|\ ...\ \|\ (\text{q},(l^{\text{q}},0)::\bar{l}_1^{\text{q}}::\bar{l}^{\text{q}})}_{\mathcal{D}_1::(\text{ALL},[mpwdp2])} ((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \text{acc},\ \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_2^{\text{q}}, \Delta_2^{\text{q}}, \text{acc}, \text{skip}))$ by SMC^2 rule Multiparty Private Pointer Dereference Write Value Higher Level Indirection, we have (B) $((1, \gamma^1,\ \sigma^1,\ \Delta^1, \text{acc},\ e)$-$\|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, e)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((1, \gamma^1,\ \sigma_1^1,\ \Delta_1^1,\ \text{acc},\ (l_e^1, \mu_e^1))\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_1^{\text{q}}, \Delta_1^{\text{q}}, \text{acc}, (l_e^{\text{q}}, \mu_e^{\text{q}})))$, (C) $\{\gamma^{\text{p}}(x) = (l^{\text{p}}, \text{private } bty*)\}_{\text{p}=1}^{\text{q}}$, (D) $\{\sigma_1^{\text{p}}(l^{\text{p}}) = (\omega^{\text{p}},\ \text{private } bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))\}_{\text{p}=1}^{\text{q}}$, (E) $\alpha > 1$, (F) $\{\text{DecodePtr}(\text{private } bty*,\ \alpha,\ \omega^{\text{p}}) = [\alpha,\ \bar{l}^{\text{p}},\ \bar{j}^{\text{p}}, i]\}_{\text{p}=1}^{\text{q}}$, (G) $i > 1$, (H) $\{\text{DynamicUpdate}(\Delta_1^{\text{p}}, \sigma_1^{\text{p}}, \bar{l}^{\text{p}},\ \text{acc}, \text{private } bty*) = (\Delta_2^{\text{p}}, \bar{l}_1^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (I) $\{\text{Retrieve_vals}(\alpha,\ \bar{l}^{\text{p}},\ \text{private } bty*, s\sigma_1^{\text{p}}) = ([[\alpha_0,\ \bar{l}_0^{\text{p}},\ \bar{j}_0^{\text{p}}, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^{\text{p}},\ \bar{j}_{\alpha-1}^{\text{p}}, i-1]], 1)\}_{\text{p}=1}^{\text{q}}$, (J) $\text{MPC}_{wdp}([[[1, [(l_e^1, \mu_e^1)], [1], i-1],\ [\alpha_0,\ \bar{l}_0^1,\ \bar{j}_0^1, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^1,\ \bar{j}_{\alpha-1}^1, i-1]], ..., [[1, [(l_e^{\text{q}}, \mu_e^{\text{q}})], [1], i-1],\ [\alpha_0,\ \bar{l}_0^{\text{q}},\ \bar{j}_0^{\text{q}}, i-1], ..., [\alpha_{\alpha-1},\ \bar{l}_{\alpha-1}^{\text{q}},\ \bar{j}_{\alpha-1}^{\text{q}}, i-1]]], [\bar{j}^1, ..., \bar{j}^{\text{q}}]) = [[[\alpha'_0,\ \bar{l}'^1_0,\ \bar{j}'^1_0, i-1], ..., [\alpha'_{\alpha-1},\ \bar{l}'^1_{\alpha-1},\ \bar{j}'^1_{\alpha-1}, i-1]], ..., [[\alpha'_0,\ \bar{l}'^{\text{q}}_0,\ \bar{j}'^{\text{q}}_0, i-1], ..., [\alpha'_{\alpha-1},\ \bar{l}'^{\text{q}}_{\alpha-1}, \bar{j}'^{\text{q}}_{\alpha-1}, i-1]]]$, (K) $\{\text{UpdateDerefVals}(\alpha, \bar{l}^{\text{p}}, [[\alpha'_0, \bar{l}'^{\text{p}}_0, \bar{j}'^{\text{p}}_0, i-1], ..., [\alpha'_{\alpha-1}, \bar{l}'^{\text{p}}_{\alpha-1}, \bar{j}'^{\text{p}}_{\alpha-1}, i-1]], \text{private } bty*, \sigma_1^{\text{p}}) = \sigma_2^{\text{p}}\}_{\text{p}=1}^{\text{q}}$.

Given (A), $((1, \widehat{\gamma}^1,\ \widehat{\sigma}^1,\ \square, \square,\ *\widehat{x} = \widehat{e})\ \|\ ...\ \|\ (\text{q}, \widehat{\gamma}^{\text{q}},\ \widehat{\sigma}^{\text{q}},\ \square, \square,\ *\widehat{x} = \widehat{e}))$ and ψ such that $\{(\text{p}, \gamma^{\text{p}}, \sigma^{\text{p}}, \Delta^{\text{p}}, \text{acc}, *x = e) \cong_{\psi} (\text{p}, \widehat{\gamma}^{\text{p}},\ \widehat{\sigma}^{\text{p}},\ \square, \square,\ *\widehat{x} = \widehat{e})\}_{\text{p}=1}^{\text{q}}$, by Definition 5.2.22 we have $\{(\gamma^{\text{p}},\ \sigma^{\text{p}}) \cong_{\psi} (\widehat{\gamma}^{\text{p}},\ \widehat{\sigma}^{\text{p}})\}_{\text{p}=1}^{\text{q}}$ and (L)

$*x = e \cong_{\psi} *\widehat{x} = \widehat{e}$. By Definition 5.2.20 we have $x \cong_{\psi} \widehat{x}$ such that (M) $x = \widehat{x}$ and (N) $e \cong_{\psi} \widehat{e}$.

Given Axiom 5.2.1, by Theorem 5.2.2 we have $\{(1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, *x = e) \sim (\text{p}, \gamma^{\text{p}}, \sigma^{\text{p}}, \Delta^{\text{p}}, \text{acc}, *x = e)\}_{\text{p}=1}^{\text{q}}$. By Lemma 5.2.86, we have $\{(\text{p}, \gamma^{\text{p}}, \sigma^{\text{p}}, \Delta^{\text{p}}, \text{acc}, *x = e) \cong_{\psi} (\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e})\}_{\text{p}=1}^{\text{q}}$. and therefore (O) $((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e}) \parallel ... \parallel (\text{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e}))$. By Definition 5.2.22 we have (P) $\{(\gamma^{\text{p}}, \sigma^{\text{p}}) \cong_{\psi} (\widehat{\gamma}, \widehat{\sigma})\}_{\text{p}=1}^{\text{q}}$.

Given (B), (P), (N), and ψ, by Lemma 5.2.76 we have (Q) $((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel ... \parallel (\text{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}))$ such that (R) $\{(\text{p}, \gamma^{\text{p}}, \sigma^{\text{p}}, \Delta^{\text{p}}, \text{acc}, e) \cong_{\psi} (\text{p}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e})\}_{\text{p}=1}^{\text{q}}$. Given (Q) and (R), by the inductive hypothesis, we have (S) $((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e}) \parallel ... \parallel (\text{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, \widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_e, \widehat{\mu}_e)) \parallel ... \parallel (\text{q}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_e, \widehat{\mu}_e)))$ and ψ_1 such that (T) $\{(\text{p}, \gamma^{\text{p}}, \sigma_1^{\text{p}}, \Delta_1^{\text{p}}, \text{acc}, (l_e^{\text{p}}, \mu_e^{\text{p}})) \cong_{\psi_1} (\text{p}, \widehat{\gamma}, \widehat{\sigma}_1, \square, \square, (\widehat{l}_e, \widehat{\mu}_e))\}_{\text{p}=1}^{\text{q}}$ and (U) $\mathcal{D}_1 \cong \widehat{\mathcal{D}}_1$. Given (T), by Definition 5.2.22 we have (V) $\{(\gamma^{\text{p}}, \sigma_1^{\text{p}}) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_1)\}_{\text{p}=1}^{\text{q}}$ and (W) $\{(l_e^{\text{p}}, \mu_e^{\text{p}}) \cong_{\psi_1} (\widehat{l}_e, \widehat{\mu}_e)\}_{\text{p}=1}^{\text{q}}$.

Given (C), (V), and (M), by Lemma 5.2.77 we have (X) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that (Y) $\{l^{\text{p}} = \widehat{l}\}_{\text{p}=1}^{\text{q}}$ and (Z) private $bty* \cong_{\psi_1} \widehat{bty*}$.

Given (D), (V), and (Y), by Lemma 5.2.78 we have (A1) $\widehat{\sigma}_1(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$ such that (B1) $\{\omega^{\text{p}} \cong_{\psi_1} \widehat{\omega}\}_{\text{p}=1}^{\text{q}}$.

Given (F), (Z), and (B1), by Lemma 5.2.11 we have (C1) $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$ such that (D1) $[\alpha, \overline{l}^{\text{p}}, \overline{j}^{\text{p}}, i] \cong_{\psi_1} [1, [(\widehat{l}_1, \widehat{\mu}_1)], [1], \widehat{i}]$. Given (D1) by Definition 5.2.15 we have (E1) $i = \widehat{i}$.

Given (G) and (E1), we have (F1) $\widehat{i} > 1$.

Given (I), (J), (K), (D1), (W), (Z), and (V), by Lemma 5.2.84 we have (G1) $\text{UpdatePtr}(\widehat{\sigma}_1, (\widehat{l}_1, \widehat{\mu}_1), [1, [(\widehat{l}_e, \widehat{\mu}_e)], [1], \widehat{i}-1], \widehat{bty*}) = (\widehat{\sigma}_2, 1)$ such that (H1) $\{(\gamma^{\text{p}}, \sigma_2^{\text{p}}) \cong_{\psi_1} (\widehat{\gamma}, \widehat{\sigma}_2)\}_{\text{p}=1}^{\text{q}}$.

Given (O), (S), (X), (A1), (C1), (F1), and (G1), by Vanilla C rule Multiparty Pointer Dereference Write Value Higher Level Indirection we have $\Sigma \triangleright ((1, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e}) \parallel ... \parallel (\text{q}, \widehat{\gamma}, \widehat{\sigma}, \square, \square, *\widehat{x} = \widehat{e})) \Downarrow'_{\widehat{\mathcal{D}}::(\text{ALL}, [\widehat{mpwdp1}])} ((1, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \text{skip}) \parallel ... \parallel (\text{q}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \text{skip}))$.

Given (H1), by Definition 5.2.22 we have $((1, \gamma^1, \sigma_2^1, \Delta_2^1, \text{acc}, \text{skip}) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma_2^{\text{q}}, \Delta_2^{\text{q}}, \text{acc}, \text{skip})) \cong_{\psi_1} ((1, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \text{skip}) \parallel ... \parallel (\text{q}, \widehat{\gamma}, \widehat{\sigma}_2, \square, \square, \text{skip}))$.

By Definition 5.2.23 we have $mpwdp2 \cong \widehat{mpwdp1}$. Given (U), $\mathcal{D}_1 :: (\text{ALL}, [mpwdp2])$ and $\widehat{\mathcal{D}}_1 :: (\text{ALL}, [\widehat{mpwdp1}])$, by Lemma 5.2.58 we have $\mathcal{D}_1 :: (\text{ALL}, [mpwdp2]) \cong \widehat{\mathcal{D}}_1 :: (\text{ALL}, [\widehat{mpwdp1}])$.

Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc},\ *x = e)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, *x = e)) \Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}^1_1::\bar{l}^1)\ \|\ ...\ \|\ (\text{q},(l^{\text{q}},0)::\bar{l}^{\text{q}}_1::\bar{l}^{\text{q}})}_{\mathcal{D}_1::(\text{ALL},[mpwdp1])}$ $((1, \gamma^1,\ \sigma^1_2,\ \Delta^1_2,\ \text{acc},\ \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}_2, \Delta^{\text{q}}_2, \text{acc}, \text{skip}))$

This case is similar to Case $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc},\ *x = e)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, *x = e))$ $\Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}^1_1::\bar{l}^1)\ \|\ ...\ \|\ (\text{q},(l^{\text{q}},0)::\bar{l}^{\text{q}}_1::\bar{l}^{\text{q}})}_{\mathcal{D}_1::(\text{ALL},[mpwdp2])}$ $((1, \gamma^1,\ \sigma^1_2,\ \Delta^1_2,\ \text{acc},\ \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}_2, \Delta^{\text{q}}_2, \text{acc}, \text{skip}))$. The main difference between the two is that $mpwdp1$ uses reasoning about evaluating an expression to multiple locations, similar to that in Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ *x = e)\ \|\ C) \Downarrow^{\mathcal{L}_1::(\text{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\text{p},[wdp2])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{skip})\ \|\ C_1)$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, \text{pfree}(x))\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{pfree}(x)))$ $\Downarrow^{(1,[(l^1,0)]::\bar{l}^1::\bar{l}^1_1)\ \|\ ...\ \|\ (\text{q},[(l^{\text{q}},0)]::\bar{l}^{\text{q}}::\bar{l}^{\text{q}}_1)}_{(\text{ALL},[mpfre])}$ $((1, \gamma^1, \sigma^1_2, \Delta^1, \text{acc}, \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}_2, \Delta^{\text{q}}, \text{acc}, \text{skip}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, \text{pfree}(x))\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{pfree}(x)))$ $\Downarrow^{(1,[(l^1,0)]::\bar{l}^1::\bar{l}^1_1)\ \|\ ...\ \|\ (\text{q},[(l^{\text{q}},0)]::\bar{l}^{\text{q}}::\bar{l}^{\text{q}}_1)}_{(\text{ALL},[mpfre])}$ $((1, \gamma^1,\ \sigma^1_2,\ \Delta^1, \text{acc}, \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}_2, \Delta^{\text{q}}, \text{acc}, \text{skip}))$ by SMC^2 rule Private Free Multiple Locations, we have (B) $\{\gamma^{\text{p}}(x) = (l^{\text{p}},\ \text{private}\ bty*)\}^{\text{q}}_{\text{p}=1}$, (C) $\text{acc} = 0$, (D) $(bty = \text{int}) \vee (bty = \text{float})$, (E) $\{\sigma^{\text{p}}(l^{\text{p}}) = (\omega^{\text{p}}, \text{private}\ bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, \alpha))\}^{\text{q}}_{\text{p}=1}$, (F) $\{\alpha > 1\}^{\text{q}}_{\text{p}=1}$, (G) $\{[\alpha,\ \bar{l}^{\text{p}},\ \bar{j}^{\text{p}},\ i] = \text{DecodePtr}(\text{private}\ bty*, \alpha,\ \omega^{\text{p}})\}^{\text{q}}_{\text{p}=1}$, (H) $\text{if}(i > 1)\{ty = \text{private}\ bty*\}$ else $\{ty = \text{private}\ bty\}$, (I) $\{\text{CheckFreeable}(\gamma^{\text{p}}, \bar{l}^{\text{p}}, \bar{j}^{\text{p}}, \sigma^{\text{p}}) = 1\}^{\text{q}}_{\text{p}=1}$, (J) $\{\forall (l^{\text{p}}_m, 0) \in \bar{l}^{\text{p}}.\ \sigma^{\text{p}}(l^{\text{p}}_m) = (\omega^{\text{p}}_m, ty, \alpha_m,$ $\text{PermL}(\text{Freeable},\ ty, \text{private}, \alpha_m))\}^{\text{q}}_{\text{p}=1}$, (K) $\text{MPC}_{free}([[\omega^1_0, ..., \omega^1_{\alpha-1}],\ ...,\ [\omega^{\text{q}}_0, ..., \omega^{\text{q}}_{\alpha-1}]], [\bar{j}^1, ...\bar{j}^{\text{q}}]) = ([[\omega'^1_0,\ ...,$ $\omega'^1_{\alpha-1}],\ ...,\ [\omega'^{\text{q}}_0, ..., \omega'^{\text{q}}_{\alpha-1}]], [\bar{j}'^1, ..., \bar{j}'^{\text{q}}])$, (L) $\{\text{UpdateBytesFree}(\sigma^{\text{p}}, \bar{l}^{\text{p}}, [\omega'^{\text{p}}_0, ..., \omega'^{\text{p}}_{\alpha-1}]) = \sigma^{\text{p}}_1\}^{\text{q}}_{\text{p}=1}$, and (M) $\{\sigma^{\text{p}}_2 = \text{UpdatePointerLocations}(\sigma^{\text{p}}_1, \bar{l}^{\text{p}}[1 : \alpha - 1], \bar{j}^{\text{p}}[1 : \alpha - 1], \bar{l}^{\text{p}}[0], \bar{j}^{\text{p}}[0])\}^{\text{q}}_{\text{p}=1}$.

Given (A), $((1, \widehat{\gamma}^1,\ \widehat{\sigma}^1,\ \square, \square,\ \text{free}(\widehat{x}))\ \|\ ...\ \|\ (\text{q}, \widehat{\gamma}^{\text{q}},\ \widehat{\sigma}^{\text{q}},\ \square, \square,\ \text{free}(\widehat{x})))$ and ψ such that $\{(\text{p}, \gamma^{\text{p}}, \sigma^{\text{p}}, \Delta^{\text{p}}, \text{acc}, \text{pfree}(x)) \cong_{\psi} (\text{p}, \widehat{\gamma}^{\text{p}},\ \widehat{\sigma}^{\text{p}},\ \square, \square,\ \text{free}(\widehat{x}))\}^{\text{q}}_{\text{p}=1}$, by Definition 5.2.22 we have $\{(\gamma^{\text{p}},\ \sigma^{\text{p}}) \cong_{\psi} (\widehat{\gamma}^{\text{p}},\ \widehat{\sigma}^{\text{p}})\}^{\text{q}}_{\text{p}=1}$ and (N) $\text{pfree}(x) \cong_{\psi} \text{free}(\widehat{x})$. By Definition 5.2.20 we have $x \cong_{\psi} \widehat{x}$ such that (O) $x = \widehat{x}$.

Given Axiom 5.2.1, by Theorem 5.2.2 we have $\{(1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, \text{pfree}(x)) \sim (\text{p},\ \gamma^{\text{p}}, \sigma^{\text{p}}, \Delta^{\text{p}},\ \text{acc}, \text{pfree}(x))\}^{\text{q}}_{\text{p}=1}$. By Lemma 5.2.86, we have $\{(\text{p}, \gamma^{\text{p}}, \sigma^{\text{p}}, \Delta^{\text{p}}, \text{acc}, \text{pfree}(x)) \cong_{\psi} (\text{p}, \widehat{\gamma},\ \widehat{\sigma},\ \square, \square,\ \text{free}(\widehat{x}))\}^{\text{q}}_{\text{p}=1}$. and therefore (P)

$((1, \widehat{\gamma},\ \widehat{\sigma},\ \Box, \Box,\ \text{free}(\widehat{x}))\ \ \|\ ...\ \|\ \ (\text{q}, \widehat{\gamma},\ \widehat{\sigma},\ \Box, \Box,\ \text{free}(\widehat{x})))$. By Definition 5.2.22 we have (Q) $\{(\gamma^{\text{p}},\ \sigma^{\text{p}}) \cong_{\psi} (\widehat{\gamma},\ \widehat{\sigma})\}_{\text{p}=1}^{\text{q}}$.

Given (B), (Q), and (O), by Lemma 5.2.77 we have (R) $\widehat{\gamma}(\widehat{x}) = (\widehat{l}, \widehat{bty*})$ such that (S) $\{l^{\text{p}} = \widehat{l}\}_{\text{p}=1}^{\text{q}}$ and (T) private $bty* \cong_{\psi} \widehat{bty*}$.

Given (E), (Q), and (S), by Lemma 5.2.78 we have (U) $\sigma(\widehat{l}) = (\widehat{\omega}, \widehat{bty*}, 1, \text{PermL}(\text{Freeable}, \widehat{bty*}, \text{public}, 1))$ such that (V) $\{\omega^{\text{p}} \cong_{\psi} \widehat{\omega}\}_{\text{p}=1}^{\text{q}}$.

Given (G), (T), and (V), by Lemma 5.2.11 we have (W) $\text{DecodePtr}(\widehat{bty*}, 1, \widehat{\omega}) = [1, [(\widehat{l}_1, 0)], [1], \widehat{i}]$ such that (X) $\{[\alpha,\ \bar{l}^{\text{p}}\ \bar{j}^{\text{p}},\ i] \cong_{\psi} [1, [(\widehat{l}_1, 0)], [1], \widehat{i}]\}_{\text{p}=1}^{\text{q}}$.

Given (I), (Q), and (X), by Axiom 5.2.3 we have (Y) $\text{CheckFreeable}(\widehat{\gamma}, [(\widehat{l}_1, 0)], [1], \widehat{\sigma}) = 1$.

Given (J), (K), (L), (M), (X), and (Q), by Lemma 5.2.85 we have (Z) $\text{Free}(\widehat{\sigma}, \widehat{l}_1) = \widehat{\sigma}_1$ and ψ_1 such that (A1) $\{(\gamma^{\text{p}},\ \sigma_2^{\text{p}}) \cong_{\psi_1} (\widehat{\gamma},\ \widehat{\sigma}_1)\}_{\text{p}=1}^{\text{q}}$.

Given (P), (R), (U), (W), (Y), and (Z), by Vanilla C rule Multiparty Free we have $\Sigma \triangleright ((1, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \text{free}(\widehat{x}))\ \|\ ...\ \|\ (\text{q}, \widehat{\gamma}, \widehat{\sigma}, \Box, \Box, \text{free}(\widehat{x}))) \Downarrow'_{(\text{ALL}, [\widehat{mpfre}])} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \text{skip})\ \|\ ...\ \|\ (\text{q}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \text{skip}))$.

Given (A1), by Definition 5.2.22 we have $((1, \gamma^1, \sigma_2^1, \Delta^1, \text{acc}, \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_2^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{skip})) \cong_{\psi_1} ((1, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \text{skip})\ \|\ ...\ \|\ (\text{q}, \widehat{\gamma}, \widehat{\sigma}_1, \Box, \Box, \text{skip}))$.
By Definition 5.2.23 we have $mpfre \cong \widehat{mpfre}$. by Definition 5.2.25 we have $(\text{ALL}, [mpfre]) \cong (\text{ALL}, [\widehat{mpfre}])$.
Therefore, by Definition 5.2.26 we have $\Pi \cong_{\psi_1} \Sigma$.

$\Box$

5.3 Noninterference

Multiparty SMC^2 satisfies a strong form of noninterference guaranteeing that two execution traces are indistinguishable up to differences in private values. This stronger version entails data-obliviousness. Instead of using execution traces, we will work directly with evaluation trees in the Multiparty SMC^2 semantics – equivalence of evaluation trees up to private values implies equivalence of execution traces based on the Multiparty SMC^2 semantics. This guarantee is provided at the semantics level, we do not consider here

compiler optimizations.

For noninterference, it is convenient to introduce a notion of equivalence requiring that the two memories agree on publicly observable values. Because we assume that private data in memories are encrypted, and so their encrypted value is publicly observable, it is sufficient to consider syntactic equality of memories. Notice that if $\sigma_1 = \sigma_2$ we can still have $\sigma_1\ell \neq \sigma_2\ell$, i.e., two executions starting from the same configuration can actually differ with respect to private data. We can now state our main noninterference result.

Theorem 5.3.1 (Multiparty Noninterference)**.** *For every environment* $\{\gamma^{\mathrm{p}}, \gamma_1^{\mathrm{p}}, \gamma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*; memory* $\{\sigma^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ $\in \mathrm{Mem}$*; location map*$\{\Delta^{\mathrm{p}}, \Delta_1^{\mathrm{p}}, \Delta_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*; accumulator* $\{\mathrm{acc}^{\mathrm{p}}, \mathrm{acc}_1^{\mathrm{p}}, \mathrm{acc}_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}} \in \mathbb{N}$*; statement* s*, values* $\{v^{\mathrm{p}}, v'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*; step evaluation code lists* $\mathcal{D}, \mathcal{D}'$ *and their corresponding lists of locations accessed* $\mathcal{L}, \mathcal{L}'$*, party* $\mathrm{p} \in \{1...\mathrm{q}\}$*;*

if $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, s) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}^{\mathrm{q}}, s))$

$\Downarrow_{\mathcal{D}}^{\mathcal{L}} ((1, \gamma_1^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}_1^1, v^1) \parallel ... \parallel (\mathrm{q}, \gamma_1^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}_1^{\mathrm{q}}, v^{\mathrm{q}}))$

and $\Sigma \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, s) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}^{\mathrm{q}}, s))$

$\Downarrow_{\mathcal{D}'}^{\mathcal{L}'} ((1, \gamma_1'^1, \sigma_1'^1, \Delta_1'^1, \mathrm{acc}_1'^1, v'^1) \parallel ... \parallel (\mathrm{q}, \gamma_1'^{\mathrm{q}}, \sigma_1'^{\mathrm{q}}, \Delta_1'^{\mathrm{q}}, \mathrm{acc}_1'^{\mathrm{q}}, v'^{\mathrm{q}}))$

then $\{\gamma_1^{\mathrm{p}} = \gamma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\sigma_1^{\mathrm{p}} = \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\Delta_1^{\mathrm{p}} = \Delta_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\mathrm{acc}_1^{\mathrm{p}} = \mathrm{acc}_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{v^{\mathrm{p}} = v'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\mathcal{D} = \mathcal{D}'$, $\mathcal{L} = \mathcal{L}'$, $\Pi \simeq_L \Sigma$.

Proof. Proof Sketch: By induction over all SMC^2 semantic rules. Notice that low-equivalence of evaluation trees already implies the equivalence of the resulting configurations. We repeated them to make the meaning of the theorem clearer. This also proves data-obliviousness over memory accesses: in any two executions of the same program with the same public data, which locations in memory are accessed will be based on public data, and therefore identical between the two executions. As we proceed to prove Theorem 5.3.2, we leverage our Axioms reasoning about noninterference of the multiparty protocols.

We make the assumption that both evaluation traces are over the same program (this is given by having the same s in the starting states) and all public data will remain the same, including data read as input during the evaluation of the program. A portion of the complexity of this proof is within ensuring that memory accesses within our semantics remain data oblivious. Several rules follow fairly simply and leverage similar ideas, which we will discuss first, and then we will provide further intuition behind the more complex cases. The full proof is available in Section 5.3.4, with this theorem identical to Theorem 5.3.2.

For all rules leveraging helper algorithms, we must reason about the helper algorithms, and that they behave deterministically by definition and have data-oblivious memory accesses. Given this and that these helper algorithms do no modify the private data, we maintain the properties of noninterference of this theorem. First we reason that our helper algorithms to translate values into their byte representation will do so deterministically, and therefore maintain

indistinguishability between the value and byte representation. We can then reason that our helper algorithms that take these byte values and store them into memory will also do so deterministically, so that when we later access the data in memory we will obtain the same indistinguishable values we had stored.

It is also important to take note here our functions to help us retrieve data from memory, particularly in cases such as when reading out of bounds of an array. When proving these cases to maintain noninterference, we leverage our definition of how memory blocks are assigned in a monotonically increasing fashion, and how the algorithms for choosing which memory block to read into after the current one are deterministic. This, as well as our original assumptions of having identical public input, allows us to reason that if we access out of bounds (including accessing data at a non-aligned position, such as a chunk of bytes in the middle of a memory block), we will be pulling from the same set of bytes each time, and therefore we will end up with the same interpretation of the data as we continue to evaluate the remainder of the program. It is important to note again here that by definition, our semantics will always interpret bytes of data as the type it is expected to be, not the type it actually is (i.e., reading bytes of data that marked private in memory by overshooting a public array will not decrypt the bytes of data, but instead give you back a garbage public value). To reiterate this point, even when reading out of bounds, we will not reveal anything about private data, as the results of these helper algorithms will be indistinguishable.

To reason about the multiparty protocols, we leverage Axioms, such as Axiom 5.3.7, to reason that the protocols will maintain our definition of noninterference. With each of these Axioms, we ensure that over two different evaluations, if the values of the first run $(v_1^{\mathrm{p}}, v_2^{\mathrm{p}})$ are not distinguishable from those of the second $(v_1'^{\mathrm{p}}, v_2'^{\mathrm{p}})$, then the resulting values are also not distinguishable $(v_3^{\mathrm{p}} = v_3'^{\mathrm{p}})$. These Axioms should be proven by a library developer to ensure the completeness of the formal model.

For private pointers, it is important to note that the obtaining multiple locations is deterministic based upon the program that is being evaluated. A pointer can initially gain multiple locations through the evaluation of a private if else. Once there exists a pointer that has obtained multiple locations in such a way, it can be assigned to another pointer to give that pointer multiple locations. The other case for a pointer to gain multiple location is through the use of `pfree` on a pointer with multiple locations (i.e., the case where a pointer has locations l_1, l_2, l_3 and we free l_1) - when this occurs, if another pointer had referred to only l_1, it will now gain locations in order to mask whether we had to move the true location or not. When reasoning about pointers with multiple locations, we maintain that given the tags for which location is the true location are indistinguishable, then it is not possible to distinguish between them by their usage as defined in the rules or helper algorithms using them. Additionally, to reason about `pfree`, we leverage that the definitions of the helper algorithms are deterministic, and that (wlog), we will be freeing the same location. We will then leverage our Axiom about the multiparty protocol MPC_{free}. After the evaluation of MPC_{free}, it will deterministically update memory and all other pointers as we mentioned in the brief example above.

For both Private If Else rules, the most important element we must leverage is how values are resolved, showing

that given our resolution style, we are not able to distinguish between the ending values. In order to do this, we also must reason about the entirety of the rule, including all of if else helper algorithms. First, we note that the evaluation of the **then** branches follows by induction, as does the evaluation of the **else** branch once we have reasoned through the restoration phase. For variable tracking, it is clear from the definitions of $\mathrm{Extract}$, $\mathrm{InitializeVariables}$, and $\mathrm{RestoreVariables}$ that the behavior of these algorithms is deterministic and given the same program, we will be extracting, initializing, and restoring the same set variables every time we evaluate the program. For location tracking, $\mathrm{Initialize}$ is also immediately clear that it will be initializing the same locations each time. We must then reason about $\mathrm{DynamicUpdate}$, and how given a program, we will deterministically find the pointer dereference writes and array writes at public indices at corresponding positions in memory and add them to our tracking structure Δ. Then we can reason that the behavior of $\mathrm{Restore}$ will deterministically perform the same updates, because Δ will contain the same information in every evaluation. Now, we are able to move on to reasoning about resolution, and show that given all of this and the definitions of the resolution helper algorithms and rule, we are not able to distinguish between the ending values.

One of the main complexities of this proof revolves around ensuring *data-oblivious memory accesses* (i.e. that we always access locations deliberately based on public information), particularly when handling arrays and pointers. Within the proof, we must consider all helper algorithms, and what locations are accessed within the algorithms as well as within the rules. What locations are accessed within the algorithms follows deterministically from the definition of the algorithms, and we return from the algorithms which locations were accessed in order to properly reason about the entire evaluation trace of the program. Our semantics are designed in such a way that we give the multiparty protocols all of the information they need, with all memory accesses being completed within the rule itself or our helper algorithms. This also helps show that memory accesses are purely local, not distributed operations. Within the array rules, the main concern is in reading from and writing at a private index. We currently handle this complexity within our rules by accessing all locations within the array in rules Multiparty Array Read Private Index and Multiparty Array Write Private Index. In Multiparty Array Read Private Index, we clearly read data from every index of the array ($\{\forall i \in \{0...\alpha-1\} \quad \mathrm{DecodeArr}(a\ bty, i, \omega_1^{\mathrm{p}}) = n_i^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$), then that data is passed to the multiparty protocol. Similarly, in Multiparty Array Write Private Index, we read data from every index of the array, pass it to the multiparty protocol, then proceed to update every index of the array with what was returned from the protocol. Within the multiparty protocols used in these two rules, we will ensure the usage of the data is data-oblivious within the main noninterference proof in the following subsection. All other array rules use public indices, and in turn only access that publicly known location. Within the pointer rules, our main concern is that we access all locations that are referred to by a private pointer when we have multiple locations. For this, we will reason about the contents of the rules and the helper algorithms used by the pointer rules, which can be shown to deterministically do so.

□

5.3.1 Supporting Metatheory

Definition 5.3.1 (ϕ)**.** We define the function ϕ to return a single unused memory block identifier in a monotonically increasing fashion.

Definition 5.3.2 ($\Pi \simeq_L \Sigma$)**.** Two SMC^2 evaluation trees Π and Σ are *low-equivalent*, in symbols $\Pi \simeq_L \Sigma$, if and only if Π and Σ have the same structure as trees, and for each node in

Π proving $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, s) \parallel ... \parallel (\mathrm{q}, \gamma^\mathrm{q}, \sigma^\mathrm{q}, \Delta^\mathrm{q}, \mathrm{acc}^\mathrm{q}, s))$

$\Downarrow^{\mathcal{L}}_{\mathcal{D}} ((1, \gamma^1_1, \sigma^1_1, \Delta^1_1, \mathrm{acc}^1_1, v^1) \parallel ... \parallel (\mathrm{q}, \gamma^\mathrm{q}_1, \sigma^\mathrm{q}_1, \Delta^\mathrm{q}_1, \mathrm{acc}^\mathrm{q}_1, v^\mathrm{q}))$, the corresponding node in

Σ proves $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, s) \parallel ... \parallel (\mathrm{q}, \gamma^\mathrm{q}, \sigma^\mathrm{q}, \Delta^\mathrm{q}, \mathrm{acc}^\mathrm{q}, s))$

$\Downarrow^{\mathcal{L}'}_{\mathcal{D}'} ((1, \gamma^1_1, \sigma^1_1, \Delta^1_1, \mathrm{acc}^1_1, v^1) \parallel ... \parallel (\mathrm{q}, \gamma^\mathrm{q}_1, \sigma^\mathrm{q}_1, \Delta^\mathrm{q}_1, \mathrm{acc}^\mathrm{q}_1, v^\mathrm{q}))$ and both $\mathcal{D} = \mathcal{D}'$ and $\mathcal{L} = \mathcal{L}'$.

Definition 5.3.3 ($\gamma = \gamma'$)**.** Two environments are equivalent, in symbols $\gamma = \gamma'$, if and only if $(x \to (l, ty)) \in \gamma \iff (x \to (l, ty)) \in \gamma'$.

Definition 5.3.4 ($\sigma = \sigma'$)**.** Two memories are equivalent, in symbols $\sigma = \sigma'$, if and only if $(l \to (\omega, ty, \alpha, \mathrm{PermL}(perm, ty, a, \alpha))) \in \sigma \iff (l \to (\omega, ty, \alpha, \mathrm{PermL}(perm, ty, a, \alpha))) \in \sigma'$.

Definition 5.3.5 ($\Delta = \Delta'$)**.** Two location maps are equivalent, in symbols $\Delta = \Delta'$, if and only if $\delta \in \Delta \iff \delta' \in \Delta'$ such that $\delta = \delta'$.

Definition 5.3.6 ($\delta = \delta'$)**.** Two nested location maps are equivalent, in symbols $\delta = \delta'$, if and only if $((l, \mu) \to (v_1, v_2, j, ty)) \in \delta \iff ((l, \mu) \to (v_1, v_2, j, ty)) \in \delta'$.

Definition 5.3.7 (Input Equality)**.** Given input files $input1, input2$, $input1 = input2$ if and only if

- for every public variable x, if $\{x = n\} \in input1$ then $\{x = n\} \in input2$,
- for every public array x, if $\{x = n_0, ..., n_m\} \in input1$ then $\{x = n_0, ..., n_m\} \in input2$,
- for every private variable x, if $\{x = n\} \in input1$ then $\{x = n'\} \in input2$ such that $n = n'$ by Axiom 5.3.1, and
- for every private array x, if $\{x = n_0, ..., n_m\} \in input1$ then $\{x = n'_0, ..., n'_m\} \in input2$ such that for every index i in $0...m$, $n_i = n'_i$ by Axiom 5.3.1.

Axiom 5.3.1 (encrypt)**.** *Given the use of an encryption scheme that ensures encrypted numbers are indistinguishable, we assume that given any two numbers n_1, n_2, their respective encrypted values* $\mathrm{encrypt}(n_1)$, $\mathrm{encrypt}(n_2)$ *can be viewed as equivalent.*

Axiom 5.3.2 (InputValue)**.** *Given two input files $input1, input2$ and variable x corresponding to a program of statement s, if and only if $input1 = input2$ by Definition 5.3.7 then* $\mathrm{InputValue}(x, input1) = n$ *and* $\mathrm{InputValue}(x, input2) = n'$ *such that $n = n'$.*

Axiom 5.3.3 (InputArray)**.** *Given two input files $input1, input2$ and array x of length m corresponding to a program of statement s, if and only if $input1 = input2$ by Definition 5.3.7 then* $\mathrm{InputArray}(x, input1, m) = [n_0, ..., n_{m-1}]$ *and* $\mathrm{InputArray}(x, input2, m) = [n'_0, ..., n'_{m-1}]$ *such that for every index i in $0...m$, $n_i = n'_i$.*

Axiom 5.3.4 (ϕ)**.** *Given a program of statement s, during any two executions Π, Σ over s such that $\Pi \simeq_L \Sigma$ by Definition 5.3.2, if ϕ returns memory block identifier l at step d in Π, then by definition 5.3.1 ϕ will also return l at step d in Σ.*

Lemma 5.3.1 (OutputValue)**.** *Given variable x, x', values n, n_1, n', n'_1 such that* $\mathrm{OutputValue}(x, n, n_1)$ *and* $\mathrm{OutputValue}(x', n', n'_1)$*, if $x = x'$, $n = n'$, and $n_1 = n'_1$, then* OutputValue *will give identical output to the same parties.*

Proof. By definition of Algorithm OutputValue, the content of the output and the parties it is given to by OutputValue is deterministic based on the given input. □

Lemma 5.3.2 (OutputArray)**.** *Given variable x, x', values n, n', α, α', $[m_0,\ ...,\ m_{\alpha-1}]$, $[m'_0,\ ...,\ m'_{\alpha'-1}]$) such that* $\mathrm{OutputArray}(x,\ n,\ [m_0,\ ...,\ m_{\alpha-1}])$ *and* $\mathrm{OutputArray}(x',\ n',\ [m'_0,\ ...,\ m'_{\alpha'-1}])$*, if $x = x'$, $n = n'$, $\alpha = \alpha'$, and $[m_0,\ ...,\ m_{\alpha-1}] = [m'_0,\ ...,\ m'_{\alpha'-1}]$), then* OutputArray *will give identical output to the same parties.*

Proof. By definition of Algorithm OutputArray, the content of the output and the parties it is given to by algorithm OutputArray is deterministic based on the given input. □

Lemma 5.3.3 (GetFunTypeList)**.** *Given parameter list $\overline{p}, \overline{p}'$, such that* $\mathrm{GetFunTypeList}(\overline{p}) = \overline{ty}$ *and* $\mathrm{GetFunTypeList}(\overline{p}') = \overline{ty}'$*, if $\overline{p} = \overline{p}'$ then $\overline{ty} = \overline{ty}'$.*

Proof. By definition of Algorithm GetFunTypeList, the type list returned by GetFunTypeList is deterministic based on the given input. □

Lemma 5.3.4 (GetFunParamAssign)**.** *Given parameter list $\overline{p} = \overline{p}'$ and expression list $\overline{e} = \overline{e}'$ such that* $\mathrm{GetFunParamAssign}(\overline{p}, \overline{e}) = s$ *and* $\mathrm{GetFunParamAssign}(\overline{p}', \overline{e}') = s'$ *if $\overline{p} = \overline{p}'$, and $\overline{e} = \overline{e}'$, then $s = s'$.*

Proof. By definition of Algorithm GetFunParamAssign, the statement returned by GetFunParamAssign is deterministic based on the given input. □

Lemma 5.3.5 (CheckPublicEffects)**.** *Given statement s, s', variable x, x', environment γ, γ', and memory σ, σ' such that* $\mathrm{CheckPublicEffects}(s,\ x,\ \gamma, \sigma) = n$ *and* $\mathrm{CheckPublicEffects}(s',\ x',\ \gamma', \sigma') = n'$ *if $s = s'$, $x = x'$, $\gamma = \gamma'$, and $\sigma = \sigma'$, then $n = n'$.*

Proof. By definition of Algorithm CheckPublicEffects, the value returned by CheckPublicEffects is deterministic based on the given input. □

Lemma 5.3.6 (τ)**.** *Given type ty, ty' such that $\tau(ty) = n$ and $\tau(ty') = n'$ if $ty = ty'$ then $n = n'$.*

Proof. By definition of Algorithm τ, the value returned by τ is deterministic based on the given input. □

Lemma 5.3.7 (Cast)**.** *Given type* ty, ty'*, privacy label* a, a'*, and value* n_1, n_1' *such that* $n_2 = \text{Cast}(a,\ ty,\ n_1)$ *and* $n_2' = \text{Cast}(a',\ ty',\ n_1')$ *if* $ty = ty'$*,* $a = a'$*, and* $n_1 = n_1'$*, then* $n_2 = n_2'$*.*

Proof. By definition of Algorithm Cast, the value returned by Cast is deterministic based on the given input. □

Lemma 5.3.8 (Free)**.** *Given memory* σ, σ' *and memory block identifier* l, l' *such that* $\text{Free}(\sigma,\ l) = (\sigma_1, (l, 0))$ *and* $\text{Free}(\sigma',\ l') = (\sigma_1', (l', 0))$ *if* $\sigma = \sigma'$ *and* $l = l'$*, then* $\sigma_1 = \sigma_1'$*.*

Proof. By definition of Algorithm Free, the memory returned by Free is deterministic based on the given input. □

Lemma 5.3.9 (IncrementList)**.** *Given location list* $\bar{l}_1, \bar{l}_1'$*, type* $\text{private } bty*, \text{private } bty'*$*, and memory* σ, σ' *such that* $\text{IncrementList}(\bar{l}_1,\ \tau(\text{private } bty*), \sigma) = (\bar{l}_2, j)$ *and* $\text{IncrementList}(\bar{l}_1',\ \tau(\text{private } bty'*), \sigma) = (\bar{l}_2', j')$ *if* $\bar{l}_1 = \bar{l}_1'$*,* $bty = bty'$*, and* $\sigma = \sigma'$*, then* $\bar{l}_2 = \bar{l}_2'$ *and* $j = j'$*.*

Proof. By definition of Algorithm IncrementList, the location list and tag returned by IncrementList is deterministic based on the given input. □

Lemma 5.3.10 (GetLocation)**.** *Given locations* $(l_1, \mu_1), (l_1', \mu_1')$*, type* $a\ bty*, a\ bty'*$*, and memory* σ, σ' *such that* $((l_2, \mu_2), j) = \text{GetLocation}((l_1, \mu_1),\ \tau(a\ bty'*),\ \sigma)$ *and* $((l_2', \mu_2'), j') = \text{GetLocation}((l_1', \mu_1'),\ \tau(a'\ bty'*),\ \sigma')$ *if* $l_1 = l_1'$*,* $\mu_1 = \mu_1'$*,* $a\ bty = a'\ bty'$*, and* $\sigma = \sigma'$*, then* $l_2 = l_2'$*,* $\mu_2 = \mu_2'$*, and* $j = j'$*.*

Proof. By definition of Algorithm GetLocation, the location and tag returned by GetLocation is deterministic based on the given input. □

Lemma 5.3.11 (ReadOOB)**.** *Given index* i, i'*, number* α, α'*, location* l_1, l_1'*, type* $ty, ty' \in \{a\ bty\}$*, and memory* σ, σ' *such that* $\text{ReadOOB}(i, \alpha, l_1, ty, \sigma) = (n, j, (l_2, \mu))$ *and* $\text{ReadOOB}(i', \alpha', l_1', ty', \sigma') = (n', j', (l_2', \mu'))$*, if* $i = i'$*,* $\alpha = \alpha'$*,* $l_1 = l_1'$*,* $ty = ty'$*, and* $\sigma = \sigma'$*, then* $n = n'$*,* $j = j'$*, and* $(l_2, \mu) = (l_2', \mu')$*.*

Proof. By definition of Algorithm ReadOOB, the value, tag, and location returned by ReadOOB is deterministic based on the given input. □

Lemma 5.3.12 (WriteOOB)**.** *Given index* i, i'*, number* α, α', n, n'*, location* l_1, l_1'*, type* $ty, ty' \in \{a\ bty\}$*, and memory* σ_1, σ_1'*, location map* Δ_1, Δ_1'*, and accumulator* acc, acc' *such that* $\text{WriteOOB}(n,\ i,\ \alpha,\ l_1,\ ty,\ \sigma_1,\ \Delta_1,\ \text{acc}) = (\sigma_2,\ \Delta_2, j, (l_2, \mu))$ *and* $\text{WriteOOB}(n',\ i',\ \alpha',\ l_1',\ ty',\ \sigma_1',\ \Delta_1',\ \text{acc}') = (\sigma_2',\ \Delta_2', j', (l_2', \mu'))$*, if* $i = i'$*,* $n = n'$*,* $\alpha = \alpha'$*,* $l_1 = l_1'$*,* $ty = ty'$*,* $\sigma_1 = \sigma_1'$*,* $\Delta_1 = \Delta_1'$*, and* $\text{acc} = \text{acc}'$*, then* $\sigma_2 = \sigma_2'$*,* $\Delta_2 = \Delta_2'$*,* $j = j'$*, and* $(l_2, \mu) = (l_2', \mu')$*.*

Proof. By definition of Algorithm WriteOOB, the memory, location map, tag, and location returned by WriteOOB is deterministic based on the given input. □

Lemma 5.3.13 (GetIndirection)**.** *Given* $*, *'$ *such that* $\mathrm{GetIndirection}(*) = i$ *and* $\mathrm{GetIndirection}(*') = i'$*, if* $|*| = |*'|$ *then* $i = i'$*.*

Proof. By definition of Algorithm GetIndirection, the level of indirection returned by GetIndirection is deterministic based on the given input, as GetIndirection counts and returns the number of * to allow for any level of indirection for pointers within our semantics. □

Lemma 5.3.14 (DerefPtr)**.** *Given memory* σ, σ'*, type* ty, ty'*, and location* $(l_1, \mu_1), (l'_1, \mu'_1)$ *such that* $\mathrm{DerefPtr}(\sigma, ty, (l_1, \mu_1)) = (n, j)$ *and* $\mathrm{DerefPtr}(\sigma', ty', (l'_1, \mu'_1)) = (n', j')$*, if* $\sigma = \sigma'$*,* $ty = ty'$*, and* $(l_1, \mu_1) = (l'_1, \mu'_1)$*, then* $n = n'$ *and* $j = j'$*.*

Proof. By definition of Algorithm DerefPtr, the value and tag (which indicates whether the access is aligned) that are returned by DerefPtr is deterministic based on the given input, and if all elements of the input are equivalent, then the output will also be equivalent. □

Lemma 5.3.15 (DerefPtrHLI)**.** *Given memory* σ, σ'*, type* ty, ty'*, and location* $(l_1, \mu_1), (l'_1, \mu'_1)$ *such that* $\mathrm{DerefPtrHLI}(\sigma, ty, (l_1, \mu_1)) = ([\alpha,\ \bar{l},\ \bar{j},\ i], j)$ *and* $\mathrm{DerefPtrHLI}(\sigma', ty', (l'_1, \mu'_1)) = ([\alpha',\ \bar{l}',\ \bar{j}',\ i'], j')$*, if* $\sigma = \sigma'$*,* $ty = ty'$*, and* $(l_1, \mu_1) = (l'_1, \mu'_1)$*, then* $[\alpha,\ \bar{l},\ \bar{j},\ i] = [\alpha',\ \bar{l}',\ \bar{j}',\ i']$ *and* $j = j'$*.*

Proof. By definition of Algorithm DerefPtrHLI, the value and tag (which indicates whether the access is aligned) that are returned by DerefPtrHLI is deterministic based on the given input, and if all elements of the input are equivalent, then the output will also be equivalent. □

Lemma 5.3.16 (Extract)**.** *Given statement* s_1, s_2, s'_1, s'_2 *such that* $\mathrm{Extract}(s_1, s_2) = (x_{list}, j)$ *and* $\mathrm{Extract}(s'_1, s'_2) = (x'_{list}, j')$ *if* $s_1 = s'_1$ *and* $s_2 = s'_2$*, then* $x_{list} = x'_{list}$ *and* $j = j'$*.*

Proof. By definition of Algorithm Extract, the variable list and tag returned by Extract is deterministic based on the given input. □

Lemma 5.3.17 (InitializeVariables)**.** *Given variable list* x_{list}, x'_{list}*, environment* γ_1, γ'_1*, memory* σ_1, σ'_1*, value* n, n' *and accumulator* $\mathrm{acc}, \mathrm{acc}'$ *such that* $\mathrm{InitializeVariables}(x_{list}, \gamma_1, \sigma_1, n, \mathrm{acc}) = (\gamma_2, \sigma_2, \bar{l})$ *and* $\mathrm{InitializeVariables}(x'_{list}, \gamma'_1, \sigma'_1, n', \mathrm{acc}') = (\gamma'_2, \sigma'_2, \bar{l}')$ *if* $x_{list} = x'_{list}$*,* $\gamma_1 = \gamma'_1$*,* $\sigma_1 = \sigma'_1$*,* $n = n'$*, and* $\mathrm{acc} = \mathrm{acc}'$*, then* $\gamma_2 = \gamma'_2$*,* $\sigma_2 = \sigma'_2$*, and* $\bar{l} = \bar{l}'$*.*

Proof. By definition of Algorithm InitializeVariables, the environment, memory, and location list returned by InitializeVariables are deterministic based on the given input. □

Lemma 5.3.18 (RestoreVariables)**.** *Given variable list* x_{list}, x'_{list}*, environment* γ, γ'*, memory* σ_1, σ'_1*, and accumulator* $\mathrm{acc}, \mathrm{acc}'$ *such that* $\mathrm{RestoreVariables}(x_{list}, \gamma, \sigma_1, \mathrm{acc}) = (\sigma_2, \bar{l})$ *and* $\mathrm{RestoreVariables}(x'_{list}, \gamma', \sigma'_1, \mathrm{acc}') = (\sigma'_2, \bar{l}')$ *if* $x_{list} = x'_{list}$*,* $\gamma = \gamma'$*,* $\sigma_1 = \sigma'_1$*, and* $\mathrm{acc} = \mathrm{acc}'$*, then* $\sigma_2 = \sigma'_2$ *and* $\bar{l} = \bar{l}'$*.*

Proof. By definition of Algorithm RestoreVariables, the memory and location list returned by RestoreVariables are deterministic based on the given input. □

Lemma 5.3.19 (ResolveVariables_Retrieve)**.** *Given variable list* x_{list}, x'_{list}*, accumulator* $\mathrm{acc}, \mathrm{acc}'$*, environment* γ, γ'*, and memory* σ, σ'*, such that*
$\mathrm{ResolveVariables_Retrieve}(x_{list},\ \mathrm{acc}, \gamma, \sigma) = ([(v_{t1}, v_{e1}), ..., (v_{tm}, v_{em})], n, \bar{l})$ *and*
$\mathrm{ResolveVariables_Retrieve}(x'_{list},\ \mathrm{acc}', \gamma', \sigma') = ([(v'_{t1}, v'_{e1}), ..., (v'_{tm}, v'_{em})], n', \bar{l}')$ *if* $x_{list} = x'_{list}$ *and* $\mathrm{acc} = \mathrm{acc}'$*, then* $[(v_{t1}, v_{e1}), ..., (v_{tm}, v_{em})] = [(v'_{t1}, v'_{e1}), ..., (v'_{tm}, v'_{em})]$ $n = n'$*, and* $\bar{l} = \bar{l}'$.

Proof. By definition of Algorithm ResolveVariables_Retrieve, the value list, number, and location list returned by ResolveVariables_Retrieve are deterministic based on the given input. □

Lemma 5.3.20 (ResolveVariables_Store)**.** *Given variable list* x_{list}, x'_{list}*, accumulator* $\mathrm{acc}, \mathrm{acc}'$*, environment* γ, γ'*, memory* σ_1, σ'_1*, and value list* $[v_1, ..., v_m], [v'_1, ..., v'_m]$*, such that*
$\mathrm{ResolveVariables_Store}(x_{list},\ \mathrm{acc}, \gamma, \sigma_1, [v_1, ..., v_m]) = (\sigma_2, \bar{l})$ *and*
$\mathrm{ResolveVariables_Store}(x'_{list},\ \mathrm{acc}', \gamma', \sigma'_1, [v'_1, ..., v'_m]) = (\sigma'_2, \bar{l}')$ *if* $x_{list} = x'_{list}$*,* $\mathrm{acc} = \mathrm{acc}'$*,* $\gamma = \gamma'$*,* $\sigma_1 = \sigma'_1$*, and* $[v_1, ..., v_m] = [v'_1, ..., v'_m]$ *then* $\sigma_2 = \sigma'_2$ *and* $\bar{l} = \bar{l}'$.

Proof. By definition of Algorithm ResolveVariables_Store, the memory and location list returned by ResolveVariables_Store are deterministic based on the given input. □

Lemma 5.3.21 (Initialize)**.** *Given location map* Δ_1, Δ'_1*, variable list* x_{list}, x'_{list}*, environment* γ_1, γ'_1*, memory* σ_1, σ'_1*, value* n, n'*, and accumulator* $\mathrm{acc}, \mathrm{acc}'$*, such that* $\mathrm{Initialize}(\Delta_1,\ x_{list},\ \gamma_1, \sigma_1, n,\ \mathrm{acc}) = (\gamma_2, \sigma_2, \Delta_2, \bar{l})$ *and* $\mathrm{Initialize}(\Delta'_1,\ x'_{list},\ \gamma'_1, \sigma'_1, n',\ \mathrm{acc}) = (\gamma'_2, \sigma'_2, \Delta'_2, \bar{l}')$ *if* $\Delta_1 = \Delta'_1$*,* $x_{list} = x'_{list}$*,* $\gamma_1 = \gamma'_1$*,* $\sigma_1 = \sigma'_1$*,* $n = n'$ *and* $\mathrm{acc} = \mathrm{acc}'$ *then* $\gamma_2 = \gamma'_2$*,* $\sigma_2 = \sigma'_2$*,* $\Delta_2 = \Delta'_2$ *and* $\bar{l} = \bar{l}'$.

Proof. By definition of Algorithm Initialize, the environment, memory, location map, and location list returned by Initialize is deterministic based on the given input. □

Lemma 5.3.22 (Restore)**.** *Given memory* σ_1, σ'_1*, location map* Δ_1, Δ'_1*, and accumulator* $\mathrm{acc}, \mathrm{acc}'$*, such that* $\mathrm{Restore}(\sigma_1,\ \Delta_1,\ \mathrm{acc}) = (\sigma_2, \Delta_2, \bar{l})$ *and* $\mathrm{Restore}(\sigma'_1, \Delta'_1, \mathrm{acc}') = (\sigma'_2, \Delta'_2, \bar{l}')$ *if* $\sigma_1 = \sigma'_1$*,* $\Delta_1 = \Delta'_1$*, and* $\mathrm{acc} = \mathrm{acc}'$ *then* $\sigma_2 = \sigma'_2$*,* $\Delta_2 = \Delta'_2$*, and* $\bar{l} = \bar{l}'$.

Proof. By definition of Algorithm Restore, the memory, location map, and location list returned by Restore is deterministic based on the given input. □

Lemma 5.3.23 (Resolve_Retrieve)**.** *Given environment* γ, γ'*, memory* σ, σ'*, location map* Δ, Δ'*, and accumulator* $\mathrm{acc}, \mathrm{acc}'$*, such that* $\mathrm{Resolve_Retrieve}(\gamma, \sigma, \Delta, \mathrm{acc}) = ([(v_{t1}, v_{e1}), ..., (v_{tm}, v_{em})], n, \bar{l})$ *and*

$\text{Resolve_Retrieve}(\gamma', \sigma', \Delta', \text{acc}') = ([(v'_{t1}, v'_{e1}), ..., (v'_{tm}, v'_{em})], n', \bar{l}')$ *if* $\gamma = \gamma'$, $\sigma = \sigma'$, $\Delta = \Delta'$, *and* $\text{acc} = \text{acc}'$, *then* $[(v_{t1}, v_{e1}), ..., (v_{tm}, v_{em})] = [(v'_{t1}, v'_{e1}), ..., (v'_{tm}, v'_{em})]$, $n = n'$, *and* $\bar{l} = \bar{l}'$.

Proof. By definition of Algorithm Resolve_Retrieve, the value list, value, and location list returned by Resolve_Retrieve is deterministic based on the given input. □

Lemma 5.3.24 (Resolve_Store)**.** *Given memory* σ_1, σ'_1, *location map* Δ_1, Δ'_1, *accumulator* acc, acc', *and values* $[v_1, ..., v_m]$, $[v'_1, ..., v'_m]$, *such that* $\text{Resolve_Store}(\Delta_1, \sigma_1, \text{acc}, [v_1, ..., v_m]) = (\sigma_2, \Delta_2, \bar{l})$ *and* $\text{Resolve_Store}(\Delta'_1, \sigma'_1, \text{acc}', [v'_1, ..., v'_m]) = (\sigma'_2, \Delta'_2, \bar{l}')$ *if* $\sigma_1 = \sigma'_1$, $\Delta_1 = \Delta'_1$, $\text{acc} = \text{acc}'$, *and* $[v_1, ..., v_m] = [v'_1, ..., v'_m]$ *then* $\sigma_2 = \sigma'_2$, $\Delta_2 = \Delta'_2$, *and* $\bar{l} = \bar{l}'$

Proof. By definition of Algorithm Resolve_Store, the memory, location map, and location list returned by Resolve_Store is deterministic based on the given input. □

Lemma 5.3.25 (DynamicUpdate)**.** *Given memory* σ, σ', *location map* Δ_1, Δ'_1, *location list* $\bar{l}_1, \bar{l}'_1$, *and type* $ty, ty' \in \{\text{private } a\ bty, \text{private } a\ bty*\}$, *such that* $\text{DynamicUpdate}(\Delta_1, \sigma, \bar{l}_1, \text{acc}, ty) = (\Delta_2, \bar{l}_2)$ *and* $\text{DynamicUpdate}(\Delta'_1, \sigma', \bar{l}'_1, \text{acc}, ty') = (\Delta'_2, \bar{l}'_2)$ *if* $\sigma = \sigma'$, $\Delta_1 = \Delta'_1$, $\bar{l}_1 = \bar{l}'_1$, $\text{acc} = \text{acc}'$, *and* $ty = ty'$, *then* $\Delta_2 = \Delta'_2$, *and* $\bar{l}_2 = \bar{l}'_2$.

Proof. By definition of Algorithm DynamicUpdate, the location map and location list returned by DynamicUpdate is deterministic based on the given input. □

Lemma 5.3.26 (DecodePtr)**.** *Given type* ty, ty', *value* α, α', *and bytes* ω, ω' *such that* $\text{DecodePtr}(ty,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]$ *and* $\text{DecodePtr}(ty',\ \alpha',\ \omega') = [\alpha',\ \bar{l}',\ \bar{j}',\ i']$, *if* $ty = ty'$, $\alpha = \alpha'$, *and* $\omega = \omega'$, *then* $\bar{l} = \bar{l}'$, $\bar{j} = \bar{j}'$, *and* $i = i'$.

Proof. By definition of Algorithm DecodePtr, the pointer data structure returned by DecodePtr is deterministic based on the given input. □

Lemma 5.3.27 (DecodeArr)**.** *Given type* $a\ bty$, $a'\ bty'$, *index* i, i', *and bytes* ω, ω' *such that* $\text{DecodeArr}(a\ bty, i, \omega) = n$ *and* $\text{DecodeArr}(a'\ bty', i', \omega') = n'$ *if* $a\ = a'$, $bty = bty'$, $i = i'$, *and* $\omega = \omega'$, *then* $n = n'$.

Proof. By definition of Algorithm DecodeArr, the value returned by DecodeArr is deterministic based on the given input. □

Lemma 5.3.28 (DecodeFun)**.** *Given bytes* ω, ω' *such that* $\text{DecodeFun}(\omega) = (s,\ n,\ \overline{p})$ *and* $\text{DecodeFun}(\omega') = (s',\ n',\ \overline{p}')$ *if* $\omega = \omega'$, *then* $s = s'$, $n = n'$, *and* $\overline{p} = \overline{p}'$.

Proof. By definition of Algorithm DecodeFun, the statement, tag, and parameter list returned by DecodeFun is deterministic based on the given input. □

Lemma 5.3.29 (DecodeVal)**.** *Given type $a\ bty$, $a'\ bty'$ and bytes ω, ω' such that* $\mathrm{DecodeVal}(a\ bty, \omega) = n$ *and* $\mathrm{DecodeVal}(a'\ bty', \omega') = n'$ *if $a\ = a'$, $bty = bty'$, and $\omega = \omega'$, then $n = n'$.*

Proof. By definition of Algorithm DecodeVal, the value returned by DecodeVal is deterministic based on the given input. □

Lemma 5.3.30 (EncodeVal)**.** *Given type $ty, ty' \in \{a\ bty\}$ and value $v, v' \in \{n, \mathrm{NULL}\}$ such that $\omega = \mathrm{EncodeVal}(ty, v)$ and $\omega' = \mathrm{EncodeVal}(ty', v')$ if $ty = ty'$ and $v = v'$ then $\omega = \omega'$.*

Proof. By definition of Algorithm EncodeVal, the byte representation returned by EncodeVal is deterministic based on the given input. □

Lemma 5.3.31 (EncodeArr)**.** *Given type $ty, ty' \in \{a\ bty\}$, index i, i', number α, α', and value $v, v' \in \{n, \mathrm{NULL}\}$ such that $\omega = \mathrm{EncodeArr}(ty, i, \alpha, v)$ and $\omega' = \mathrm{EncodeArr}(ty', i', \alpha', v')$ if $ty = ty'$, $i = i'$, $\alpha = \alpha'$, and $v = v'$, then $\omega = \omega'$.*

Proof. By definition of Algorithm EncodeArr, the byte representation returned by EncodeArr is deterministic based on the given input. □

Lemma 5.3.32 (EncodePtr)**.** *Given type $ty, ty' \in \{a\ bty*, a\ \mathrm{const}\ bty*\}$, number of locations α, α', location list $\bar{l}, \bar{l}'$, tag list $\bar{j}, \bar{j}'$, and level of indirection i, i' such that $\omega = \mathrm{EncodePtr}(ty,\ [\alpha,\ \bar{l},\ \bar{j},\ i])$ and $\omega' = \mathrm{EncodePtr}(ty',\ [\alpha',\ \bar{l}',\ \bar{j}',\ i'])$ if $ty = ty'$, $\alpha = \alpha'$, $\bar{l} = \bar{l}'$, $\bar{j} = \bar{j}'$, and $i = i'$, then $\omega = \omega'$.*

Proof. By definition of Algorithm EncodePtr, the byte representation returned by EncodePtr is deterministic based on the given input. □

Lemma 5.3.33 (EncodeFun)**.** *Given statement s, s', value n, n', and parameter list $\overline{p}, \overline{p}'$ such that* $\mathrm{EncodeFun}(s, n, \overline{p}) = \omega$ *and* $\mathrm{EncodeFun}(s', n', \overline{p}') = \omega'$, *if $s = s'$, $n = n'$, and $\overline{p} = \overline{p}'$, then $\omega = \omega'$.*

Proof. By definition of Algorithm EncodeFun, the byte representation returned by EncodeFun is deterministic based on the given input. □

Lemma 5.3.34 (UpdateVal)**.** *Given memory σ_1, σ_1', memory block identifier l, l', value n, n', and type $a\ bty$, $a'\ bty'$ such that* $\mathrm{UpdateVal}(\sigma_1,\ l,\ n,\ a\ bty) = \sigma_2$ *and* $\mathrm{UpdateVal}(\sigma_1',\ l',\ n',\ a'\ bty') = \sigma_2'$ *if $\sigma_1 = \sigma_1'$, $l = l'$, $n = n'$, $a = a'$, and $bty = bty'$, then $\sigma_2 = \sigma_2'$.*

Proof. By definition of Algorithm UpdateVal, the memory returned by UpdateVal is deterministic based on the given input. □

Lemma 5.3.35 (UpdateArr)**.** *Given memory σ_1, σ_1', memory block identifier l, l', index i, i', value n, n', and type $a\ bty, a'\ bty'$ such that* $\mathrm{UpdateArr}(\sigma_1,\ (l,i),\ n,\ a\ bty) = \sigma_2$ *and* $\mathrm{UpdateArr}(\sigma_1',\ (l',i'),\ n',\ a'\ bty') = \sigma_2'$ *if $\sigma_1 = \sigma_1'$, $l = l'$, $i = i'$, $n = n'$, $a = a'$, and $bty = bty'$, then $\sigma_2 = \sigma_2'$.*

Proof. By definition of Algorithm UpdateArr, the memory returned by UpdateArr is deterministic based on the given input. □

Lemma 5.3.36 (UpdatePtr)**.** *Given memory σ_1, σ_1', location $(l,\mu), (l',\mu')$, pointer data structure $[\alpha,\ \bar{l},\ \bar{j},\ i]$, $[\alpha',\ \bar{l}',\ \bar{j}',\ i']$, and type $a\ bty*, a'\ bty'*$ such that* $\mathrm{UpdatePtr}(\sigma_1,\ (l,\mu),\ [\alpha,\ \bar{l},\ \bar{j},\ i],\ a\ bty*) = (\sigma_2,\ j)$ *and* $\mathrm{UpdatePtr}(\sigma_1',\ (l',\mu'),\ [\alpha',\ \bar{l}',\ \bar{j}',\ i'],\ a'\ bty'*) = (\sigma_2',\ j')$ *if $\sigma_1 = \sigma_1'$, $l = l'$, $\mu = \mu'$ $\alpha = \alpha'$, $a = a'$, $bty = bty'$, $\bar{l} = \bar{l}'$, $\bar{j} = \bar{j}'$, and $i = i'$, then $\sigma_2 = \sigma_2'$ and $j = j'$.*

Proof. By definition of Algorithm UpdatePtr, the memory and tag returned by UpdatePtr is deterministic based on the given input. □

Lemma 5.3.37 (UpdateOffset)**.** *Given memory σ_1, σ_1', location $(l,\mu), (l',\mu')$, number n, n' and type $a\ bty, a'\ bty'$ such that* $\mathrm{UpdateOffset}(\sigma_1,\ (l,\mu),\ n,\ a\ bty) = (\sigma_2, j)$ *and* $\mathrm{UpdateOffset}(\sigma_1',\ (l',\mu'),\ n',\ a\ bty') = (\sigma_2', j')$ *if $\sigma_1 = \sigma_1'$, $l = l'$, $\mu = \mu'$ $n = n'$, $a = a'$, and $bty = bty'$, then $\sigma_2 = \sigma_2'$ and $j = j'$.*

Proof. By definition of Algorithm UpdateOffset, the memory and tag returned by UpdateOffset is deterministic based on the given input. □

Lemma 5.3.38 ($\mathcal{D}_1 :: \mathcal{D}_2 = \mathcal{D}_1' :: \mathcal{D}_2'$)**.** *Given $\mathcal{D}_1 :: \mathcal{D}_2$, $\mathcal{D}_1' :: \mathcal{D}_2'$, if $\mathcal{D}_1 = \mathcal{D}_1'$ and $\mathcal{D}_2' = \mathcal{D}_2'$ then $\mathcal{D}_1 :: \mathcal{D}_2 = \mathcal{D}_1' :: \mathcal{D}_2'$.*

Proof. By definition of Algorithm 140, the result of adding party-wise evaluation code lists is deterministic based on the content and ordering of the party-wise evaluation code lists. □

Lemma 5.3.39. *Given number α, α', location list $\{\bar{l}^{\mathrm{p}}, \bar{l}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, type ty, ty', and memory $\{\sigma^{\mathrm{p}}, \sigma'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ such that* $\{\mathrm{Retrieve_vals}(\alpha,\ \bar{l}^{\mathrm{p}},\ ty,\ \sigma^{\mathrm{p}}) = ([v_0^{\mathrm{p}},\ ...,\ v_{\alpha-1}^{\mathrm{p}}], j^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{\mathrm{Retrieve_vals}(\alpha',\ \bar{l}'^{\mathrm{p}},\ ty',\ \sigma'^{\mathrm{p}}) = ([v_0'^{\mathrm{p}},\ ...,\ v_{\alpha'-1}'^{\mathrm{p}}], j'^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$*, if $\alpha = \alpha'$, $\{\bar{l}^{\mathrm{p}} = \bar{l}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $ty = ty'$, and $\{\sigma^{\mathrm{p}} = \sigma'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, then $\{\forall i \in \{0...\alpha-1\}, v_i^{\mathrm{p}} = v_i'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\{j^{\mathrm{p}} = j'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.*

Proof. By definition of Algorithm Retrieve_vals, the values returned by Retrieve_vals are deterministic based on the given input. □

Lemma 5.3.40. *Given environment $\{\gamma^{\mathrm{p}}, \gamma'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, location list $\{\bar{l}^{\mathrm{p}}, \bar{l}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, tag list $\{\bar{j}^{\mathrm{p}}, \bar{j}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and memory $\{\sigma^{\mathrm{p}}, \sigma'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ such that* $\{\mathrm{CheckFreeable}(\gamma^{\mathrm{p}}, \bar{l}^{\mathrm{p}}, \bar{j}^{\mathrm{p}}, \sigma^{\mathrm{p}}) = j\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{\mathrm{CheckFreeable}(\gamma'^{\mathrm{p}}, \bar{l}'^{\mathrm{p}}, \bar{j}'^{\mathrm{p}}, \sigma'^{\mathrm{p}}) = j'\}_{\mathrm{p}=1}^{\mathrm{q}}$ *if $\{\gamma^{\mathrm{p}} = \gamma'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\bar{l}^{\mathrm{p}} = \bar{l}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\bar{j}^{\mathrm{p}} = \bar{j}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and $\{\sigma^{\mathrm{p}} = \sigma'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ then $j = j'$.*

Proof. By definition of Algorithm CheckFreeable, the tag returned by CheckFreeable is deterministic based on the input. □

Lemma 5.3.41. *Given memory* $\{\sigma_1^{\mathrm{p}}, \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, number* α, α'*, location list* $\{\bar{l}^{\mathrm{p}}, \bar{l}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, and byte representations* $\{[\omega_0^{\mathrm{p}}, ..., \omega_{\alpha-1}^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$*,* $\{[\omega_0'^{\mathrm{p}}, ..., \omega_{\alpha'-1}'^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$ *such that* $\{\mathrm{UpdateBytesFree}(\sigma_1^{\mathrm{p}}, \bar{l}^{\mathrm{p}}, [\omega_0^{\mathrm{p}}, ..., \omega_{\alpha-1}^{\mathrm{p}}]) = \sigma_2^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{\mathrm{UpdateBytesFree}(\sigma_1^{\mathrm{p}}, \bar{l}'^{\mathrm{p}}, [\omega_0'^{\mathrm{p}}, ..., \omega_{\alpha'-1}'^{\mathrm{p}}]) = \sigma_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, if* $\{\sigma_1^{\mathrm{p}} = \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*,* $\{\bar{l}^{\mathrm{p}} = \bar{l}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*,* $\alpha = \alpha'$*, and* $\{[\omega_0^{\mathrm{p}}, ..., \omega_{\alpha-1}^{\mathrm{p}}] = [\omega_0'^{\mathrm{p}}, ..., \omega_{\alpha'-1}'^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$*, then* $\{\sigma_2^{\mathrm{p}} = \sigma_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Proof. By definition of Algorithm UpdateBytesFree, the memory returned by UpdateBytesFree is deterministic based on the input. □

Lemma 5.3.42. *Given memory* $\{\sigma_1^{\mathrm{p}}, \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, location list* $\{\bar{l}^{\mathrm{p}}, \bar{l}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, and tag list* $\{\bar{j}^{\mathrm{p}}, \bar{j}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *such that* $\{\mathrm{UpdatePointerLocations}(\sigma_1^{\mathrm{p}}, \bar{l}_1^{\mathrm{p}}[1:\alpha-1], \bar{j}^{\mathrm{p}}[1:\alpha-1], \bar{l}_1^{\mathrm{p}}[0], \bar{j}^{\mathrm{p}}[0]) = (\sigma_2^{\mathrm{p}}, \bar{l}_2^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{\mathrm{UpdatePointerLocations}(\sigma_1'^{\mathrm{p}}, \bar{l}_1'^{\mathrm{p}}[1:\alpha'-1], \bar{j}'^{\mathrm{p}}[1:\alpha'-1], \bar{l}_1'^{\mathrm{p}}[0], \bar{j}'^{\mathrm{p}}[0]) = (\sigma_2'^{\mathrm{p}}, \bar{l}_2'^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$*, if* $\{\sigma_1^{\mathrm{p}} = \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*,* $\{\bar{l}_1^{\mathrm{p}} = \bar{l}_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, and* $\{\bar{j}^{\mathrm{p}} = \bar{j}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, then* $\{\sigma_2^{\mathrm{p}} = \sigma_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{\bar{l}_2^{\mathrm{p}} = \bar{l}_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Proof. By definition of Algorithm UpdatePointerLocations, the memory and location list returned by UpdatePointerLocations is deterministic based on the input. □

Lemma 5.3.43. *Given number* α, α'*, location list* $\{\bar{l}^{\mathrm{p}}, \bar{l}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, type* ty, ty'*, values* $\{[v_0^{\mathrm{p}}, ..., v_{\alpha-1}^{\mathrm{p}}], [v_0'^{\mathrm{p}}, ..., v_{\alpha'-1}'^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$*, and memory* $\{\sigma_1^{\mathrm{p}}, \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *such that* $\{\mathrm{UpdateDerefVals}(\alpha, \bar{l}^{\mathrm{p}}, [v_0^{\mathrm{p}}, ..., v_{\alpha-1}^{\mathrm{p}}], ty, \sigma_1^{\mathrm{p}}) = \sigma_2^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ *and* $\{\mathrm{UpdateDerefVals}(\alpha', \bar{l}'^{\mathrm{p}}, [v_0'^{\mathrm{p}}, ..., v_{\alpha'-1}'^{\mathrm{p}}], ty', \sigma_1'^{\mathrm{p}}) = \sigma_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, if* $\alpha = \alpha'$*,* $\{\bar{l}^{\mathrm{p}} = \bar{l}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*,* $ty = ty'$*,* $\{[v_0^{\mathrm{p}}, ..., v_{\alpha-1}^{\mathrm{p}}] = [v_0'^{\mathrm{p}}, ..., v_{\alpha'-1}'^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$*, and* $\{\sigma_1^{\mathrm{p}} = \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, then* $\{\sigma_2^{\mathrm{p}} = \sigma_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Proof. By definition of Algorithm UpdateDerefVals, the memory returned by UpdateDerefVals is deterministic based on the input. □

5.3.2 Multiparty Computation Axioms

Axiom 5.3.5 (MPC_{ar})**.** *Given indices* $\{i^{\mathrm{p}}, i'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, arrays* $\{[n_0^{\mathrm{p}}, ..., n_{\alpha-1}^{\mathrm{p}}], [n_0'^{\mathrm{p}}, ..., n_{\alpha'-1}'^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$*,*
if $\mathrm{MPC}_{ar}((i^1, [n_0^1, ..., n_{\alpha-1}^1]), ..., (i^{\mathrm{q}}, [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}])) = (n^1, ..., n^{\mathrm{q}})$*,*
$\mathrm{MPC}_{ar}((i'^1, [n_0'^1, ..., n_{\alpha'-1}'^1]), ..., (i'^{\mathrm{q}}, [n_0'^{\mathrm{q}}, ..., n_{\alpha'-1}'^{\mathrm{q}}])) = (n'^1, ..., n'^{\mathrm{q}})$*,*
$\{i^{\mathrm{p}} = i'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, and* $\{[n_0^{\mathrm{p}}, ..., n_{\alpha-1}^{\mathrm{p}}] = [n_0'^{\mathrm{p}}, ..., n_{\alpha'-1}'^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$
then $\{n^{\mathrm{p}} = n'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Axiom 5.3.6 (MPC_{aw})**.** *Given indices* $\{i^{\mathrm{p}}, i'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*, arrays* $\{[n_0^{\mathrm{p}}, ..., n_{\alpha-1}^{\mathrm{p}}], [n_0''^{\mathrm{p}}, ..., n_{\alpha'-1}''^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$*, and values* $\{n^{\mathrm{p}}, n'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$*,*
if $\mathrm{MPC}_{aw}((i^1, n^1, [n_0^1, ..., n_{\alpha-1}^1]), ..., (i^{\mathrm{q}}, n^{\mathrm{q}}, [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}])) = ([n_0'^1, ..., n_{\alpha-1}'^1], ..., [n_0'^{\mathrm{q}}, ..., n_{\alpha-1}'^{\mathrm{q}}])$*,*

$\mathrm{MPC}_{aw}((i'^1, n'^1, [n''^1_0, ..., n''^1_{\alpha'-1}]), ..., (i'^q, n'^q, [n''^q_0, ..., n''^q_{\alpha'-1}])) = ([n'''^1_0, ..., n'''^1_{\alpha'-1}], ..., [n'''^q_0, ..., n'''^q_{\alpha'-1}])$,

$\{i^p = i'^p\}^q_{p=1}$, $\{n^p = n'^p\}^q_{p=1}$ *and* $\{[n^p_0, ..., n^p_{\alpha-1}] = [n''^p_0, ..., n''^p_{\alpha'-1}]\}^q_{p=1}$

then $\{[n'^p_0, ..., n'^p_{\alpha-1}] = [n'''^p_0, ..., n'''^p_{\alpha'-1}]\}^q_{p=1}$.

Axiom 5.3.7 (MPC_b)**.** *Given values* $\{v^p_1, v^p_2, v^p_3, v'^p_1, v'^p_2, v'^p_3\}^q_{p=1}$ *and binary operation bop* $\in \{\cdot, +, -, \div\}$,

if $\mathrm{MPC}_b(bop, v^1_1, v^1_2, ..., v^q_1, v^q_2) = (v^1_3, ..., v^q_3)$,

$\mathrm{MPC}_b(bop, v'^1_1, v'^1_2, ..., v'^q_1, v'^q_2) = (v'^1_3, ..., v'^q_3)$, $\{v^p_1 = v'^p_1\}^q_{p=1}$, *and* $\{v^p_2 = v'^p_2\}^q_{p=1}$

then $\{v^p_3 = v'^p_3\}^q_{p=1}$.

Axiom 5.3.8 (MPC_{cmp})**.** *Given values* $\{v^p_1, v^p_2, v^p_3, v'^p_1, v'^p_2, v'^p_3\}^q_{p=1}$ *and binary operation bop* $\in \{==, !=, <\}$,

if $\mathrm{MPC}_{cmp}(bop, v^1_1, v^1_2, ..., v^q_1, v^q_2) = (v^1_3, ..., v^q_3)$,

$\mathrm{MPC}_{cmp}(bop, v'^1_1, v'^1_2, ..., v'^q_1, v'^q_2) = (v'^1_3, ..., v'^q_3)$, $\{v^p_1 = v'^p_1\}^q_{p=1}$, *and* $\{v^p_2 = v'^p_2\}^q_{p=1}$

then $\{v^p_3 = v'^p_3\}^q_{p=1}$.

Axiom 5.3.9 (MPC_u)**.** *Given values* $\{n^p_1, n'^p_1\}^q_{p=1}$ *and binary operation uop* $\in \{++\}$,

if $\mathrm{MPC}_u(uop, n^1_1, ..., n^q_1) = (n^1_2, ..., n^q_2)$,

$\mathrm{MPC}_u(uop, n'^1_1, ..., n'^q_1) = (n'^1_2, ..., n'^q_2)$, *and* $\{n^p_1 = n'^p_1\}^q_{p=1}$,

then $\{n^p_2 = n'^p_2\}^q_{p=1}$.

Axiom 5.3.10 ($\mathrm{MPC}_{resolve}$)**.** *Given values* $\{n^1, n'^p, [(v^p_{t1}, v^p_{e1}), ..., (v^p_{tm}, v^p_{em})], [(v'^1_{t1}, v'^1_{e1}), ..., (v'^1_{tm}, v'^1_{em})]\}^q_{p=1}$,

if $\mathrm{MPC}_{resolve}([n^1, ..., n^q], [[(v^1_{t1}, v^1_{e1}), ..., (v^1_{tm}, v^1_{em})], ..., [(v^q_{t1}, v^q_{e1}), ..., (v^q_{tm}, v^q_{em})]])$

$= [[v^1_1, ..., v^1_m], ..., [v^q_1, ..., v^q_m]]$

$\mathrm{MPC}_{resolve}([n'^1, ..., n'^q], [[(v'^1_{t1}, v'^1_{e1}), ..., (v'^1_{tm}, v'^1_{em})], ..., [(v'^q_{t1}, v'^q_{e1}), ..., (v'^q_{tm}, v'^q_{em})]])$

$= [[v'^1_1, ..., v'^1_m], ..., [v'^q_1, ..., v'^q_m]]$,

$\{n^p = n'^p\}^q_{p=1}$ *and* $\{[(v^p_{t1}, v^p_{e1}), ..., (v^p_{tm}, v^p_{em})] = [(v'^p_{t1}, v'^p_{e1}), ..., (v'^p_{tm}, v'^p_{em})]\}^q_{p=1}$,

then $\{[v^p_1, ..., v^p_m] = [v^p_1, ..., v^p_m]\}^q_{p=1}$.

Axiom 5.3.11 (MPC_{dv})**.** *Given values* $\{[n^p_0, ..., n^p_{\alpha-1}]\}^q_{p=1}$, *and tag lists* $\{\bar{j}^p, \bar{j}'^p\}^q_{p=1}$,

if $\mathrm{MPC}_{dv}([[n^1_0, ..., n^1_{\alpha-1}], ..., [n^q_0, ..., n^q_{\alpha-1}]], [\bar{j}^1, ..., \bar{j}^q]) = (n^1, ..., n^q)$,

$\mathrm{MPC}_{dv}([[n'^1_0, ..., n'^1_{\alpha'-1}], ..., [n'^q_0, ..., n'^q_{\alpha'-1}]], [\bar{j}'^1, ..., \bar{j}'^q]) = (n'^1, ..., n'^q)$,

$\{[n^p_0, ..., n^p_{\alpha-1}] = [n'^p_0, ..., n'^p_{\alpha'-1}]\}^q_{p=1}$, *and* $\{\bar{j}^p = \bar{j}'^p\}^q_{p=1}$

then $\{n^p = n'^p\}^q_{p=1}$.

Axiom 5.3.12 (MPC_{dp})**.** *Given values* $\{\forall i \in \{0...\alpha-1\}[\alpha_i, \bar{l}^p_i, \bar{j}^p_i, i]\}^q_{p=1}$, $\{\forall j \in \{0...\alpha'-1\}[\alpha'_j, \bar{l}'^p_j, \bar{j}'^p_j, i']\}^q_{p=1}$

and tag lists $\{\bar{j}^p, \bar{j}'^p\}^q_{p=1}$,

if $\mathrm{MPC}_{dp}([[[\alpha_0, \bar{l}^1_0, \bar{j}^1_0], ..., [\alpha_{\alpha-1}, \bar{l}^1_{\alpha-1}, \bar{j}^1_{\alpha-1}]], ..., [[\alpha_0, \bar{l}^q_0, \bar{j}^q_0], ..., [\alpha_{\alpha-1}, \bar{l}^q_{\alpha-1}, \bar{j}^q_{\alpha-1}]]], [\bar{j}^1, ..., \bar{j}^q]) = ([[\alpha_\alpha, \bar{l}^1_\alpha, \bar{j}^1_\alpha], ...,$
$[\alpha_\alpha, \bar{l}^q_\alpha, \bar{j}^q_\alpha]])$,

$\mathrm{MPC}_{dp}([[[\alpha'_0,\ \bar{l}'^1_0,\ \bar{j}'^1_0],...,[\alpha'_{\alpha'-1},\ \bar{l}'^1_{\alpha'-1},\ \bar{j}'^1_{\alpha'-1}]],...,[[\alpha'_0,\ \bar{l}'^q_0,\ \bar{j}'^q_0],...,\ [\alpha'_{\alpha'-1},\ \bar{l}'^q_{\alpha'-1},\ \bar{j}'^q_{\alpha'-1}]]],[\bar{j}'^1,...,\bar{j}'^q]) =$
$([[\alpha'_{\alpha'},\ \bar{l}'^1_{\alpha'},\ \bar{j}'^1_{\alpha'}],...,\ [\alpha'_{\alpha'},\ \bar{l}'^q_{\alpha'},\ \bar{j}'^q_{\alpha'}]])$,
$\alpha = \alpha'$, $i = i'$, $\{\forall i \in \{0...\alpha-1\}, [\alpha_i,\ \bar{l}^p_i,\ \bar{j}^p_i] = [\alpha'_i,\ \bar{l}'^p_i,\ \bar{j}'^p_i]\}^q_{p=1}$, *and* $\{\bar{j}^p = \bar{j}'^p\}^q_{p=1}$
then $\{[\alpha_\alpha,\ \bar{l}^p_\alpha,\ \bar{j}^p_\alpha] = [\alpha'_{\alpha'},\ \bar{l}'^p_{\alpha'},\ \bar{j}'^p_{\alpha'}]\}^q_{p=1}$.

Axiom 5.3.13 (MPC_{free})**.** *Given number* α, α', *bytes* $\{[\omega^p_0,\ ...,\ \omega^p_{\alpha-1}]\}^q_{p=1}$, $\{[\omega''^p_0,\ ...,\ \omega''^p_{\alpha'-1}]\}^q_{p=1}$, *and tag lists* $\{\bar{j}^p, \bar{j}''^p\}^q_{p=1}$,
if $\mathrm{MPC}_{free}([[\omega^1_0,...,\omega^1_{\alpha-1}],\ ...,\ [\omega^q_0,...,\omega^q_{\alpha-1}]],\ [\bar{j}^1,...\bar{j}^q]) = ([[\omega'^1_0,...,\omega'^1_{\alpha-1}],...,\ [\omega'^q_0,...,\omega'^q_{\alpha-1}]],\ [\bar{j}'^1,...,\bar{j}'^q])$,
$\mathrm{MPC}_{free}([[\omega''^1_0,...,\omega''^1_{\alpha'-1}],\ ...,\ [\omega''^q_0,...,\omega''^q_{\alpha'-1}]],\ [\bar{j}''^1,...\bar{j}''^q]) = ([[\omega'''^1_0,...,\omega'''^1_{\alpha'-1}],...,\ [\omega'''^q_0,...,\omega'''^q_{\alpha'-1}]],\ [\bar{j}'''^1,...,$
$\bar{j}'''^q])$, $\alpha = \alpha'$, $\{[\omega^p_0,...,\omega^p_{\alpha-1}] = [\omega''^p_0,...,\omega''^p_{\alpha'-1}]]\}^q_{p=1}$, *and* $\{\bar{j}^p = \bar{j}''^p\}^q_{p=1}$
then $\{[\omega'^p_0,...,\omega'^p_{\alpha-1}] = [\omega'''^p_0,...,\omega'''^p_{\alpha'-1}]]\}^q_{p=1}$ *and* $\{\bar{j}'^p = \bar{j}'''^p\}^q_{p=1}$.

Axiom 5.3.14 (MPC_{wdv})**.** *Given list of values* $\{[n^p_0,...,n^p_{\alpha-1}]\}^q_{p=1}$, $\{[n''^p_0,...,n''^p_{\alpha'-1}]\}^q_{p=1}$, *number* α, α', $\{n^p, n'^p\}^q_{p=1}$, *and tag list* $\{\bar{j}^p, \bar{j}'^p\}^q_{p=1}$,
if $\mathrm{MPC}_{wdv}([[n^1_0,...,n^1_{\alpha-1}],...,[n^q_0,...,n^q_{\alpha-1}]],\ [n^1,...,n^q],\ [\bar{j}^1,\ ...,\ \bar{j}^q]) = ([n'^1_0,\ ...,\ n'^1_{\alpha-1}],...,[n'^q_0,...,n'^q_{\alpha-1}])$,
$\mathrm{MPC}_{wdv}([[n''^1_0,...,n''^1_{\alpha'-1}],...,[n''^q_0,...,n''^q_{\alpha'-1}]],\ [n'^1,...,n'^q],\ [\bar{j}'^1,\ ...,\ \bar{j}'^q]) = ([n'''^1_0,\ ...,\ n'''^1_{\alpha'-1}],...,[n'''^q_0,...,n'''^q_{\alpha'-1}])$,
$\{[n^p_0,...,n^p_{\alpha-1}] = [n''^p_0,...,n''^p_{\alpha'-1}]\}^q_{p=1}$, $\alpha = \alpha'$, $\{n^p = n'^p\}^q_{p=1}$ *and* $\{\bar{j}^p = \bar{j}'^p\}^q_{p=1}$
then $\{[n'^p_0,...,n'^p_{\alpha-1}] = [n'''^p_0,...,n'''^p_{\alpha'-1}]\}^q_{p=1}$.

Axiom 5.3.15 (MPC_{wdp})**.** *Given location list* $\{[\alpha_e, \bar{l}^p_e, \bar{j}^p_e, i],\ [\alpha'_e, \bar{l}'^p_e, \bar{j}'^p_e, i']\}^q_{p=1}$, $\{\forall m \in \{0...\alpha-1\},\ [\alpha_m,\ \bar{l}^p_m,\ \bar{j}^p_m, i]\}^q_{p=1}$, $\{\forall m' \in \{0...\alpha''-1\},\ [\alpha''_{m'},\ \bar{l}''^p_{m'},\ \bar{j}''^p_{m'}, i']\}^q_{p=1}$, *and tag list* $\{\bar{j}^p, \bar{j}'^p\}^q_{p=1}$,
if $\mathrm{MPC}_{wdp}([[[\alpha_e, \bar{l}^1_e, \bar{j}^1_e, i],\ [\alpha_0,\ \bar{l}^1_0,\ \bar{j}^1_0, i],...,\ [\alpha_{\alpha-1},\ \bar{l}^1_{\alpha-1},\ \bar{j}^1_{\alpha-1}, i]],...,\ [[\alpha_e, \bar{l}^q_e, \bar{j}^q_e, i],\ [\alpha_0,\ \bar{l}^q_0,\ \bar{j}^q_0, i],...,\ [\alpha_{\alpha-1},$
$\bar{l}^q_{\alpha-1},\ \bar{j}^q_{\alpha-1}, i]]],\ [\bar{j}^1,...,\bar{j}^q]) = [[[\alpha'_0,\ \bar{l}'^1_0,\ \bar{j}'^1_0, i],...,\ [\alpha'_{\alpha-1},\ \bar{l}'^1_{\alpha-1},\ \bar{j}'^1_{\alpha-1}, i]],...,\ [[\alpha'_0, \bar{l}'^q_0, \bar{j}'^q_0, i],...,\ [\alpha'_{\alpha-1},\ \bar{l}'^q_{\alpha-1},$
$\bar{j}'^q_{\alpha-1}, i]]]$,
$\mathrm{MPC}_{wdp}([[[\alpha'_e, \bar{l}'^1_e, \bar{j}'^1_e, i'],\ [\alpha''_0,\ \bar{l}''^1_0,\ \bar{j}''^1_0, i'],...,\ [\alpha''_{\alpha''-1},\ \bar{l}''^1_{\alpha''-1},\ \bar{j}''^1_{\alpha''-1}, i']],...,\ [[\alpha'_e, \bar{l}'^q_e, \bar{j}'^q_e, i'],\ [\alpha''_0,\ \bar{l}''^q_0, \bar{j}''^q_0, i'],...,$
$[\alpha''_{\alpha''-1},\ \bar{l}''^q_{\alpha''-1}, \bar{j}''^q_{\alpha''-1}, i']]],\ [\bar{j}'^1,...,\bar{j}'^q]) = [[[\alpha'''_0,\ \bar{l}'''^1_0,\ \bar{j}'''^1_0, i'],...,\ [\alpha'''_{\alpha''-1},\ \bar{l}'''^1_{\alpha''-1}, \bar{j}'''^1_{\alpha''-1}, i']],...,\ [[\alpha'''_0,\ \bar{l}'''^q_0,\ \bar{j}'''^q_0, i'],...,$
$[\alpha'''_{\alpha''-1},\ \bar{l}'''^q_{\alpha''-1},\ \bar{j}'''^q_{\alpha''-1}, i']]]$,
$\alpha = \alpha''$, $\{[\alpha_e, \bar{l}^p_e, \bar{j}^p_e, i] = [\alpha'_e, \bar{l}'^p_e, \bar{j}'^p_e, i']\}^q_{p=1}$, $\{\forall m \in \{0...\alpha-1\},\ [\alpha_m,\ \bar{l}^p_m,\ \bar{j}^p_m, i] = [\alpha''_{m'},\ \bar{l}''^p_{m'},\ \bar{j}''^p_{m'}, i']\}^q_{p=1}$, *and* $\{\bar{j}^p = \bar{j}'^p\}^q_{p=1}$
then $\{\forall m \in \{0...\alpha-1\},\ [\alpha'_m,\ \bar{l}'^p_m,\ \bar{j}'^p_m, i] = [\alpha'''_{m'},\ \bar{l}'''^p_{m'},\ \bar{j}'''^p_{m'}, i']\}^q_{p=1}$.

5.3.3 Location Access Tracking Supporting Metatheory

Definition 5.3.8 (Location Access)**.** A location in memory (l, μ) is defined to have been accessed if we look up memory block identifier l in memory σ and obtain or modify the data that is stored at offset μ from within memory block.

Definition 5.3.9 ($\bar{l} = \bar{l}'$)**.** Two location lists are equivalent, in symbols $\bar{l} = \bar{l}'$, if and only if $(l, \mu) \in \bar{l} \iff (l, \mu) \in \bar{l}'$.

Definition 5.3.10 ($\mathcal{L} = \mathcal{L}'$)**.** Two party-wise location lists are equivalent, in symbols $\mathcal{L} = \mathcal{L}'$, if and only if $(\mathrm{p}, \bar{l}) \in \mathcal{L} \iff (\mathrm{p}, \bar{l}) \in \mathcal{L}'$.

Lemma 5.3.44 ($\mathcal{L}_1, \mathcal{L}_2$)**.** *Given two party-wise location lists $\mathcal{L}_1, \mathcal{L}_2$, if $\mathcal{L}_1$ was accessed first and $\mathcal{L}_2$ accessed second, then we have $\mathcal{L}_1 :: \mathcal{L}_2$.*

Proof. By the definition of Algorithm 139 and analysis of all rule cases. □

Lemma 5.3.45 ($(\mathrm{p}, \bar{l}_1) :: (\mathrm{p}, \bar{l}_1)$)**.** *Given two party-wise location lists $(\mathrm{p}, \bar{l}_1)$, $(\mathrm{p}, \bar{l}_2)$, if $(\mathrm{p}, \bar{l}_1) :: (\mathrm{p}, \bar{l}_2)$, then $(\mathrm{p}, \bar{l}_1 :: \bar{l}_2)$.*

Proof. By the definition of Algorithm 139. □

Lemma 5.3.46 ($\{(\mathrm{p}, \bar{l}_1^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}} :: \{(\mathrm{p}, \bar{l}_2^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$)**.** *Given $\{(\mathrm{p}, \bar{l}_1^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ and $\{(\mathrm{p}, \bar{l}_2^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ if $\{(\mathrm{p}, \bar{l}_1^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}} :: \{(\mathrm{p}, \bar{l}_2^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ then $(1, \bar{l}_1^1 :: \bar{l}_2^1) \parallel ... \parallel (\mathrm{q}, \bar{l}_1^{\mathrm{q}} :: \bar{l}_2^{\mathrm{q}})$.*

Proof. By the definition of Algorithm 139. □

Lemma 5.3.47 ($\mathcal{L}_1 :: \mathcal{L}_2 = \mathcal{L}_1' :: \mathcal{L}_2'$)**.** *Given $\mathcal{L}_1 :: \mathcal{L}_2$, $\mathcal{L}_1' :: \mathcal{L}_2'$, if $\mathcal{L}_1 = \mathcal{L}_1'$ and $\mathcal{L}_2' = \mathcal{L}_2'$ then $\mathcal{L}_1 :: \mathcal{L}_2 = \mathcal{L}_1' :: \mathcal{L}_2'$.*

Proof. By Definition 5.3.10. □

Lemma 5.3.48 (Free Location Access)**.** *Given memory σ and memory block identifier l, if $\mathrm{Free}(\sigma,\ l) = (\sigma_1, (l, 0))$ then $(l, 0)$ has been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm Free. □

Lemma 5.3.49 (ReadOOB Location Access)**.** *Given index i, number of elements α, type ty, memory σ and memory block identifier l_1, if $\mathrm{ReadOOB}(i, \alpha, l_1, ty, \sigma) = (n, 1, (l_2, \mu))$ then (l_2, μ) has been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm ReadOOB. □

Lemma 5.3.50 (WriteOOB Location Access)**.** *Given index i, numbers n, α, type ty, memory σ_1, location map Δ_1, and memory block identifier l_1, if $\mathrm{WriteOOB}(n,\ i,\ \alpha,\ l_1,\ ty,\ \sigma_1,\ \Delta_1, \mathrm{acc}) = (\sigma_2,\ \Delta_2, j, (l_2, \mu))$ then (l_2, μ) has been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm WriteOOB. □

Lemma 5.3.51 (Memory Addition Location Access)**.** *Given memory σ, memory block identifier l, bytes ω, number α, type ty and privacy label a, and permission $perm$, if $\sigma_1 = \sigma[l \rightarrow (\omega,\ ty,\ \alpha,\ \mathrm{PermL}(perm, ty, a, \alpha))]$ then the location $(l, 0)$ has been accessed.*

Proof. By Definition 5.3.8. □

Lemma 5.3.52 (Memory Modification Location Access)**.** *Given memory σ, memory block identifier l, bytes ω, ω', number α, type ty and privacy label a, and permission $perm$, if $\sigma = \sigma_1[l \rightarrow (\omega,\ ty,\ \alpha,\ \mathrm{PermL}(perm, ty, a, \alpha))]$ and $\sigma_2 = \sigma_1[l \rightarrow (\omega',\ ty,\ \alpha,\ \mathrm{PermL}(perm, ty, a, \alpha))]$ then the location $(l, 0)$ has been accessed.*

Proof. By Definition 5.3.8. □

Lemma 5.3.53 (InitializeVariables Location Access)**.** *Given variable list x_{list}, environment γ_1, memory σ_1, value n, and accumulator* acc*, if* $\mathrm{InitializeVariables}(x_{list},\ \gamma_1,\ \sigma_1,\ n,\ \mathrm{acc}) = (\gamma_2, \sigma_2, \bar{l})$ *then the locations $\bar{l}$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm InitializeVariables. □

Lemma 5.3.54 (RestoreVariables Location Access)**.** *Given environment γ, memory σ_1, variable list x_{list}, and accumulator* acc*, if* $\mathrm{RestoreVariables}(x_{list}, \gamma, \sigma_1, \mathrm{acc}) = (\sigma_2, \bar{l})$ *then the locations $\bar{l}$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm RestoreVariables. □

Lemma 5.3.55 (ResolveVariables_Retrieve Location Access)**.** *Given environment γ, memory σ, variable list x_{list}, and accumulator* acc*, if* $\mathrm{ResolveVariables_Retrieve}(x_{list}, \mathrm{acc}, \gamma, \sigma) = ([(v_{t1}, v_{e1}), ..., (v_{tm}, v_{em})], n, \bar{l})$ *then the locations $\bar{l}$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm ResolveVariables_Retrieve. □

Lemma 5.3.56 (ResolveVariables_Store Location Access)**.** *Given environment γ, memory σ_1, variable list x_{list}, values $[v_1, ..., v_m]$, and accumulator* acc*, if* $\mathrm{ResolveVariables_Store}(x_{list}, \mathrm{acc}, \gamma, \sigma_1, [v_1, ..., v_n]) = (\sigma_2, \bar{l})$ *then the locations $\bar{l}$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm ResolveVariables_Store. □

Lemma 5.3.57 (Initialize Location Access)**.** *Given location map Δ_1, variable list x_{list}, environment γ_1, memory σ_1, value n, and accumulator* acc*, if* $\mathrm{Initialize}(\Delta_1, x_{list},\ \gamma_1,\ \sigma_1,\ n, \mathrm{acc}) = (\gamma_2, \sigma_2, \Delta_2, \bar{l})$ *then the locations $\bar{l}$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm Initialize. □

Lemma 5.3.58 (Restore Location Access)**.** *Given memory σ_1, location map Δ_1, and accumulator* acc*, if* $\text{Restore}(\sigma_1, \Delta_1, \text{acc}) = (\sigma_2, \Delta_2, \bar{l})$ *then the locations $\bar{l}$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm Restore. □

Lemma 5.3.59 (Resolve_Retrieve Location Access)**.** *Given environment γ, memory σ, location map Δ, and accumulator* acc*, if* $\text{Resolve_Retrieve}(\gamma, \sigma, \Delta, \text{acc}) = ([(v_{t1}, v_{e1}), ..., (v_{tm}, v_{em})], n, \bar{l})$ *then the locations $\bar{l}$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm Resolve_Retrieve. □

Lemma 5.3.60 (Resolve_Store Location Access)**.** *Given memory σ_1, location map Δ_1, values $[v_1, ..., v_m]$, and accumulator* acc*, if* $\text{Resolve_Store}(\Delta_1, \sigma_1, \text{acc}, [v_1, ..., v_m]) = (\sigma_2, \Delta_2, \bar{l})$ *then the locations $\bar{l}$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm Resolve_Store. □

Lemma 5.3.61 (DynamicUpdate Location Access)**.** *Given memory σ, location map Δ_1, location list $\bar{l}_1$, and type $ty \in \{\text{private } a\ bty, \text{private } a\ bty*\}$, if* $\text{DynamicUpdate}(\Delta_1, \sigma, \bar{l}_1, \text{acc}, ty) = (\Delta_2, \bar{l}_2)$ *then the locations $\bar{l}_2$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm DynamicUpdate. □

Lemma 5.3.62 (Pointer Data Location Access)**.** *Given memory σ, memory block identifier l, and $ty \in \{a\ bty*, \text{const } a\ bty*\}$, if $\sigma(l) = (\omega,\ ty,\ \alpha, \text{PermL}(\text{Freeable } ty, a, \alpha))$ and* $\text{DecodePtr}(ty,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]$*, then the location $(l, 0)$ has been accessed.*

Proof. By Definition 5.3.8. □

Lemma 5.3.63 (Array Data Location Access)**.** *Given memory σ and memory block identifier l, if $\sigma(l) = (\omega, a\ bty, \alpha, \text{PermL}(\text{Freeable } a\ bty, a, \alpha))$ and* $\text{DecodeArr}(a\ bty,\ i,\ \omega) = n$*, then the location (l, i) has been accessed.*

Proof. By Definition 5.3.8. □

Lemma 5.3.64 (Data Location Access)**.** *Given memory σ and memory block identifier l, if $\sigma(l) = (\omega,\ a\ bty,\ 1, \text{PermL}(\text{Freeable } a\ bty, a, 1))$ and* $\text{DecodeVal}(a\ bty,\ \omega) = n$*, then the location $(l, 0)$ has been accessed.*

Proof. By Definition 5.3.8. □

Lemma 5.3.65 (Function Data Location Access)**.** *Given memory σ and memory block identifier l, if $\sigma(l) = (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \text{PermL_Fun}(\text{public}))$ and* $\text{DecodeFun}(\omega) = (s,\ n,\ \overline{p})$*, then the location $(l, 0)$ has been accessed.*

Proof. By Definition 5.3.8. □

Lemma 5.3.66 (UpdateVal Location Access)**.** *Given memory σ_1, memory block identifier l, value n, and type $a\ bty$, if* $\text{UpdateVal}(\sigma_1,\ l,\ n,\ a\ bty) = \sigma_2$*, then the location* $(l, 0)$ *has been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm UpdateVal. □

Lemma 5.3.67 (UpdateArr Location Access)**.** *Given memory σ_1, memory block identifier l, index i, value n, and type $a\ bty$, if* $\text{UpdateArr}(\sigma_1,\ (l, i),\ n,\ a\ bty) = \sigma_2$ *then the location* (l, i) *has been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm UpdateArr. □

Lemma 5.3.68 (UpdatePtr Location Access)**.** *Given memory σ_1, location (l, μ), pointer data structure $[\alpha,\ \bar{l},\ \bar{j},\ i]$, and type $a\ bty*$,*
if $\text{UpdatePtr}(\sigma_1,\ (l, \mu),\ [\alpha,\ \bar{l},\ \bar{j},\ i], a\ bty*) = (\sigma_2,\ j)$ *then the location* (l, μ) *has been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm UpdatePtr. □

Lemma 5.3.69 (UpdateOffset Location Access)**.** *Given memory σ_1, location (l, μ), number n and type $a\ bty$, if* $\text{UpdateOffset}(\sigma_1,\ (l, \mu),\ n, a\ bty) = (\sigma_2, j)$ *then the location* (l, μ) *has been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm UpdateOffset. □

Lemma 5.3.70 (DerefPtr Location Access)**.** *Given memory σ, location (l, μ), and type ty,*
if $\text{DerefPtr}(\sigma, ty, (l, \mu)) = (n, j)$ *then the location* (l, μ) *has been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm DerefPtr. □

Lemma 5.3.71 (DerefPtrHLI Location Access)**.** *Given memory σ, location (l, μ), and type ty,*
if $\text{DerefPtrHLI}(\sigma,\ ty, (l, \mu)) = ([\alpha, \bar{l}, \bar{j}, i], j)$ *then the location* (l, μ) *has been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm DerefPtrHLI. □

Lemma 5.3.72 (Retrieve_vals Location Access)**.** *Given number α, location list $\bar{l}$, type ty, and memory σ,*
if $\text{Retrieve_vals}(\alpha, \bar{l}, ty, \sigma) = ([v_0, ...v_{\alpha'-1}], j)$ *then all locations in $\bar{l}$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm Retrieve_vals. □

Lemma 5.3.73 (CheckFreeable Location Access)**.** *Given environment γ, location list $\bar{l}$, tag list $\bar{j}$, and memory σ,*
if $\text{CheckFreeable}(\gamma, \bar{l}, \bar{j}, \sigma) = j$ *then all locations in $\bar{l}$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm CheckFreeable. □

Lemma 5.3.74 (UpdateBytesFree Location Access)**.** *Given location list $\bar{l}$, byte representations $[\omega_0, ..., \omega_{\alpha-1}]$, and memory σ_1, if* $\text{UpdateBytesFree}(\sigma_1, \bar{l}, [\omega_0, ..., \omega_{\alpha-1}]) = \sigma_2$ *then all locations in $\bar{l}$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm UpdateBytesFree. □

Lemma 5.3.75 (UpdatePointerLocations Location Access)**.** *Given location list $\bar{l}$, tag list $\bar{j}$, and memory σ_1 if* $\text{UpdatePointerLocations}(\sigma_1, \bar{l}[1:\alpha-1], \bar{j}[1:\alpha-1], \bar{l}[0], \bar{j}[0]) = (\sigma_2, \bar{l}_1)$*, then all locations in $\bar{l}_1$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm UpdatePointerLocations. □

Lemma 5.3.76 (UpdateDerefVals Location Access)**.** *Given number α, location list $\bar{l}$, list of values $[v_0, ..., v_{\alpha-1}]$, type ty, and memory σ_1,*
if $\text{UpdateDerefVals}(\alpha, \bar{l}, [v_0, ..., v_{\alpha-1}], ty, \sigma_1) = \sigma_2$ *then all locations in $\bar{l}$ have been accessed.*

Proof. By Definition 5.3.8 and the definition of Algorithm UpdateDerefVals. □

5.3.4 Proof of Noninterference

Theorem 5.3.2 (Multiparty Noninterference)**.** *For every environment $\{\gamma^{\mathrm{p}}, \gamma_1^{\mathrm{p}}, \gamma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$; memory $\{\sigma^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ $\in \mathrm{Mem}$; location map$\{\Delta^{\mathrm{p}}, \Delta_1^{\mathrm{p}}, \Delta_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$; accumulator $\{\mathrm{acc}^{\mathrm{p}}, \mathrm{acc}_1^{\mathrm{p}}, \mathrm{acc}_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}} \in \mathbb{N}$; statement s, values $\{v^{\mathrm{p}}, v'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$; step evaluation code lists $\mathcal{D}, \mathcal{D}'$ and their corresponding lists of locations accessed $\mathcal{L}, \mathcal{L}'$, party $\mathrm{p} \in \{1...\mathrm{q}\}$;*
if $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, s) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}^{\mathrm{q}}, s))$
$\Downarrow_{\mathcal{D}}^{\mathcal{L}} ((1, \gamma_1^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}_1^1, v^1) \parallel ... \parallel (\mathrm{q}, \gamma_1^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}_1^{\mathrm{q}}, v^{\mathrm{q}}))$
and $\Sigma \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, s) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}^{\mathrm{q}}, s))$
$\Downarrow_{\mathcal{D}'}^{\mathcal{L}'} ((1, \gamma_1'^1, \sigma_1'^1, \Delta_1'^1, \mathrm{acc}_1'^1, v'^1) \parallel ... \parallel (\mathrm{q}, \gamma_1'^{\mathrm{q}}, \sigma_1'^{\mathrm{q}}, \Delta_1'^{\mathrm{q}}, \mathrm{acc}_1'^{\mathrm{q}}, v'^{\mathrm{q}}))$
then $\{\gamma_1^{\mathrm{p}} = \gamma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\sigma_1^{\mathrm{p}} = \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\Delta_1^{\mathrm{p}} = \Delta_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{\mathrm{acc}_1^{\mathrm{p}} = \mathrm{acc}_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{v^{\mathrm{p}} = v'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\mathcal{D} = \mathcal{D}'$, $\mathcal{L} = \mathcal{L}'$, $\Pi \simeq_L \Sigma$.

Proof.
Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, x[e]) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, x[e])) \Downarrow_{\mathcal{D}_1::(\mathrm{ALL},[mpra])}^{\mathcal{L}_1::\mathcal{L}_2} ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, x[e]) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}^{\mathrm{q}}, x[e])) \Downarrow_{\mathcal{D}_1::(\mathrm{ALL},[mpra])}^{\mathcal{L}_1::\mathcal{L}_2} ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}}))$, by rule Multiparty Array Read Private Index we have $\{(e) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{(n^{\mathrm{p}}) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (B) $\{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \mathrm{const}\ a\ bty*)\}_{\mathrm{p}=1}^{\mathrm{q}}$, (C) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, i^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, i^{\mathrm{q}}))$, (D) $\{\sigma_1^{\mathrm{p}}(l^{\mathrm{p}}) = (\omega^{\mathrm{p}}, a\ \mathrm{const}\ bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, a\ \mathrm{const}\ bty*, a, 1))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (E) $\{\mathrm{DecodePtr}(a\ \mathrm{const}\ bty*, 1, \omega^{\mathrm{p}}) = [1, [(l_1^{\mathrm{p}}, 0)], [1], 1]\}_{\mathrm{p}=1}^{\mathrm{q}}$, (F) $\{\sigma_1^{\mathrm{p}}(l_1^{\mathrm{p}}) = (\omega_1^{\mathrm{p}}, a\ bty, \alpha, \mathrm{PermL}(\mathrm{Freeable}, a\ bty, a, \alpha))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (G) $\{\forall i \in \{0...\alpha-1\} \quad \mathrm{DecodeArr}(a\ bty, i, \omega_1^{\mathrm{p}}) = n_i^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (H) $\mathrm{MPC}_{ar}((i^1,$

$[n_0^1, ..., n_{\alpha-1}^1]), ..., (i^{\mathrm{q}}, [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}])) = (n^1, ..., n^{\mathrm{q}})$, and (I) $\mathcal{L}_2 = (1, [(l^1, 0), (l_1^1, 0), ..., (l_1^1, \alpha-1)]) \parallel ... \parallel (\mathrm{q}, [(l^{\mathrm{q}}, 0), (l_1^{\mathrm{q}}, 0), ..., (l_1^{\mathrm{q}}, \alpha - 1)])$.

Given (J) $\Sigma \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, x[e]) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}^{\mathrm{q}}, x[e])) \Downarrow^{\mathcal{L}_1' :: \mathcal{L}_2'}_{\mathcal{D}_1' :: \mathcal{D}_2' :: (\mathrm{ALL}, [d])} ((1, \gamma_2'^1, \sigma_2'^1, \Delta_2'^1, \mathrm{acc}^1, v'^1) \parallel ... \parallel (\mathrm{q}, \gamma_2'^{\mathrm{q}}, \sigma_2'^{\mathrm{q}}, \Delta_2'^{\mathrm{q}}, \mathrm{acc}^{\mathrm{q}}, v'^{\mathrm{q}}))$ and (A), by Lemma 5.2.87 we have (K) $d = mpra$.

Given (J) and (K), by SMC^2 rule Multiparty Array Read Private Index we have $\{(e) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $\{(n'^{\mathrm{p}}) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (L) $\{\gamma^{\mathrm{p}}(x) = (l'^{\mathrm{p}}, \mathrm{const}\ a'\ bty'*)\}_{\mathrm{p}=1}^{\mathrm{q}}$, (M) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow^{\mathcal{L}_1'}_{\mathcal{D}_1'} ((1, \gamma^1, \sigma_1'^1, \Delta_1'^1, \mathrm{acc}, i'^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1'^{\mathrm{q}}, \Delta_1'^{\mathrm{q}}, \mathrm{acc}, i'^{\mathrm{q}}))$, (N) $\{\sigma_1'^{\mathrm{p}}(l'^{\mathrm{p}}) = (\omega'^{\mathrm{p}},\ a'\ \mathrm{const}\ bty'*, 1, \mathrm{PermL}(\mathrm{Freeable}, a'\ \mathrm{const}\ bty'*, a',\ 1))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (O) $\{\mathrm{DecodePtr}(a'\ \mathrm{const}\ bty'*,\ 1,\ \omega'^{\mathrm{p}}) = [1,\ [(l_1'^{\mathrm{p}}, 0)],\ [1],\ 1]\}_{\mathrm{p}=1}^{\mathrm{q}}$, (P) $\{\sigma_1'^{\mathrm{p}}(l_1'^{\mathrm{p}}) = (\omega_1'^{\mathrm{p}}, a'\ bty', \alpha', \mathrm{PermL}(\mathrm{Freeable},\ a'\ bty', a', \alpha'))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (Q) $\{\forall i' \in \{0...\alpha' - 1\} \quad \mathrm{DecodeArr}(a'\ bty', i', \omega_1'^{\mathrm{p}}) = n_{i'}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (R) $\mathrm{MPC}_{ar}((i'^1,\ [n_0'^1,\ ..., n_{\alpha'-1}'^1]), ..., (i'^{\mathrm{q}}, [n_0'^{\mathrm{q}}, ..., n_{\alpha'-1}'^{\mathrm{q}}])) = (n'^1, ..., n'^{\mathrm{q}})$, and (S) $\mathcal{L}_2' = (1, [(l'^1, 0), (l_1'^1, 0), ..., (l_1'^1, \alpha' - 1)]) \parallel ... \parallel (\mathrm{q}, [(l'^{\mathrm{q}}, 0), (l_1'^{\mathrm{q}}, 0), ..., (l_1'^{\mathrm{q}}, \alpha' - 1)])$.

Given (B) and (L), by Definition 5.3.3 we have (T) $\{l^{\mathrm{p}} = l'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (U) $a = a'$, and (V) $bty = bty'$.

Given (C) and (M), by the inductive hypothesis we have (W) $\{\sigma_1^{\mathrm{p}} = \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (X) $\{\Delta_1^{\mathrm{p}} = \Delta_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (Y) $\{i^{\mathrm{p}} = i'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (Z) $\mathcal{L}_1 = \mathcal{L}_1'$, and (A1) $\mathcal{D}_1 = \mathcal{D}_1'$.

Given (D), (N), (W), and (T), by Definition 5.3.4 we have (B1) $\{\omega^{\mathrm{p}} = \omega'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (E), (O), (U), (V), and (B1), by Lemma 5.3.26 we have (C1) $\{l_1^{\mathrm{p}} = l_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (F), (P), (W), and (C1), by Definition 5.3.4 we have (D1) $\{\omega_1^{\mathrm{p}} = \omega_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (E1) $\alpha = \alpha'$.

Given (G), (Q), and (E1), we have $i = i'$. Given (U), (V), and (D1), by Lemma 5.3.27 we have (F1) $\forall i \in \{0...\alpha - 1\}\{n_i^{\mathrm{p}} = n_i'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (H), (R), (Y), and (F1), by Axiom 5.3.5 we have (G1) $\{n^{\mathrm{p}} = n'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (D) and (E), by Lemma 5.3.62 we have accessed locations (H1) $\{(\mathrm{p}, [(l^{\mathrm{p}}, 0)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (F) and (G), by Lemma 5.3.63 we have accessed locations (I1) $\{(\mathrm{p}, [(l_1^{\mathrm{p}}, 0), ..., (l_1^{\mathrm{p}}, \alpha - 1)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (H1) and (I1), by Lemmas 5.3.44 and 5.3.46 we have (I).

Given (N) and (O), by Lemma 5.3.62 we have accessed locations (J1) $\{(\mathrm{p}, [(l^{\mathrm{p}}, 0)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (P) and (Q), by Lemma 5.3.63 we have accessed locations (K1) $\{(\mathrm{p}, [(l_1'^1, 0), ..., (l_1'^1, \alpha' - 1)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (J1) and (K1), by Lemmas 5.3.44 and 5.3.46 we have (S).

Given (T), (C1), (E1), (I), and (S), by Definition 5.3.10 we have (L1) $\mathcal{L}_2 = \mathcal{L}_2'$. Given (Z) and (L1), by Lemma 5.3.47 we have (M1) $\mathcal{L}_1 :: \mathcal{L}_2 = \mathcal{L}_1' :: \mathcal{L}_2'$.

Given (A1) and $(\mathrm{ALL}, [mpra])$, by Lemma 5.3.38 we have (N1) $\mathcal{D}_1 :: (\mathrm{ALL}, [mpra]) = \mathcal{D}_1' :: (\mathrm{ALL}, [mpra])$.

Given (W), (X), (G1), (M1), and (N1), by Definition 5.3.2, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, x[e_1] = e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, x[e_1] = e_2)) \Downarrow_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpwa])}^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3} ((1, \gamma^1, \sigma_{3+\alpha-1}^1, \Delta_2^1, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_{3+\alpha-1}^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, x[e_1] = e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, x[e_1] = e_2)) \Downarrow_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpwa])}^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3} ((1, \gamma^1, \sigma_{3+\alpha-1}^1, \Delta_2^1, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_{3+\alpha-1}^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip}))$ by SMC^2 rule Multiparty Array Write Private Index, we have (B) $\{(e_1) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (C) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1)) \Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, i^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, i^{\mathrm{q}}))$, (D) $((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}\mathrm{acc}, e_2)) \Downarrow_{\mathcal{D}_2}^{\mathcal{L}_2} ((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}}))$, (E) $\{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \text{private const } bty*)\}_{\mathrm{p}=1}^{\mathrm{q}}$, (F) $\{\sigma_2^{\mathrm{p}}(l^{\mathrm{p}}) = (\omega^{\mathrm{p}}, \text{private const } bty*, 1, \mathrm{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (G) $\{\mathrm{DecodePtr}(\text{private const } bty*,\ 1, \omega^{\mathrm{p}}) = [1, [(l_1^{\mathrm{p}}, 0)], [1],\ 1]\}_{\mathrm{p}=1}^{\mathrm{q}}$, (H) $\{\sigma_2^{\mathrm{p}}(l_1^{\mathrm{p}}) = (\omega_1^{\mathrm{p}}, \text{private } bty, \alpha, \mathrm{PermL}(\text{Freeable, private } bty, \text{private}, \alpha))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (I) $\{\forall j \in \{0...\alpha-1\}\ \mathrm{DecodeArr}(\text{private } bty, j, \omega_1^{\mathrm{p}}) = n_j^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (J) $\mathrm{MPC}_{aw}((i^1, n^1, [n_0^1, ..., n_{\alpha-1}^1]), ..., (i^{\mathrm{q}}, n^{\mathrm{q}}, [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}])) = ([n_0'^1, ..., n_{\alpha-1}'^1], ..., [n_0'^{\mathrm{q}}, ..., n_{\alpha-1}'^{\mathrm{q}}])$, (K) $\{\forall j \in \{0...\alpha-1\}\ \mathrm{UpdateArr}(\sigma_{2+j}^{\mathrm{p}}, (l_1^{\mathrm{p}}, j), n_j'^{\mathrm{p}}, \text{ private } bty) = \sigma_{3+j}^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (L) $\mathcal{L}_3 = (1, [(l^{\mathrm{p}}, 0), (l_1^{\mathrm{p}}, 0), ..., (l_1^{\mathrm{p}}, \alpha-1)]) \parallel ... \parallel (\mathrm{q}, [(l^{\mathrm{p}}, 0), (l_1^{\mathrm{p}}, 0), ..., (l_1^{\mathrm{p}}, \alpha-1)])$.

Given (M) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, x[e_1] = e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, x[e_1] = e_2)) \Downarrow_{\mathcal{D}_1'::\mathcal{D}_2'::(\mathrm{ALL},[d])}^{\mathcal{L}_1'::\mathcal{L}_2'::\mathcal{L}_3'} ((1, \gamma^1, \sigma_{3+\alpha-1}'^1, \Delta_2'^1, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_{3+\alpha-1}'^{\mathrm{q}}, \Delta_2'^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip}))$ and (A), by Lemma 5.2.87 we have (N) $d = mpwa$.

Given (M) and (N), by SMC^2 rule Multiparty Array Write Private Index, we have (O) $\{(e_1) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (P) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1)) \Downarrow_{\mathcal{D}_1'}^{\mathcal{L}_1'} ((1, \gamma^1, \sigma_1'^1, \Delta_1'^1, \mathrm{acc}, i'^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1'^{\mathrm{q}}, \Delta_1'^{\mathrm{q}}, \mathrm{acc}, i'^{\mathrm{q}}))$, (Q) $((1, \gamma^1, \sigma_1'^1, \Delta_1'^1, \mathrm{acc}, e_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1'^{\mathrm{q}}, \Delta_1'^{\mathrm{q}}\mathrm{acc}, e_2)) \Downarrow_{\mathcal{D}_2'}^{\mathcal{L}_2'} ((1, \gamma^1, \sigma_2'^1, \Delta_2'^1, \mathrm{acc}, n'^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2'^{\mathrm{q}}, \Delta_2'^{\mathrm{q}},$

acc, n'^{q})), (R) $\{\gamma^{\mathrm{p}}(x) = (l'^{\mathrm{p}}, \text{ private const } bty'*)\}_{\mathrm{p}=1}^{\mathrm{q}}$, (S) $\{\sigma_2'^{\mathrm{p}}(l'^{\mathrm{p}}) = (\omega'^{\mathrm{p}}, \text{ private const } bty'*, 1, \text{PermL(Freeable, private const } bty'*, \text{private}, 1))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (T) $\{\text{DecodePtr(private const } bty'*, 1, \omega'^{\mathrm{p}}) = [1, [(l_1'^{\mathrm{p}}, 0)], [1], 1]\}_{\mathrm{p}=1}^{\mathrm{q}}$, (U) $\{\sigma_2'^{\mathrm{p}}(l_1'^{\mathrm{p}}) = (\omega_1'^{\mathrm{p}}, \text{private } bty', \alpha', \text{PermL(Freeable, private } bty', \text{ private}, \alpha'))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (V) $\{\forall j' \in \{0...\alpha'-1\}$ DecodeArr(private $bty', j', \omega_1'^{\mathrm{p}}) = n_j''^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (W) $\mathrm{MPC}_{aw}((i'^1, n'^1, [n_0''^1, ..., n_{\alpha'-1}''^1]), ..., (i'^{\mathrm{q}}, n'^{\mathrm{q}}, [n_0''^{\mathrm{q}}, ..., n_{\alpha'-1}''^{\mathrm{q}}])) = ([n_0'''^1, ..., n_{\alpha'-1}'''^1], ..., [n_0'''^{\mathrm{q}}, ..., n_{\alpha'-1}'''^{\mathrm{q}}])$, (X) $\{\forall j' \in \{0...\alpha'-1\}$ UpdateArr($\sigma_{2+j'}'^{\mathrm{p}}$, $(l_1'^{\mathrm{p}}, j')$, $n_{j'}'''^{\mathrm{p}}$, private $bty') = \sigma_{3+j'}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (Y) $\mathcal{L}_3' = (1, [(l'^{\mathrm{p}}, 0), (l_1'^{\mathrm{p}}, 0), ..., (l_1'^{\mathrm{p}}, \alpha'-1)]) \parallel ... \parallel (\mathrm{q}, [(l'^{\mathrm{p}}, 0), (l_1'^{\mathrm{p}}, 0), ..., (l_1'^{\mathrm{p}}, \alpha'-1)])$.

Given (C) and (P), by the inductive hypothesis we have (Z) $\{\sigma_1^{\mathrm{p}} = \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (A1) $\{\Delta_1^{\mathrm{p}} = \Delta_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (B1) $\{i^{\mathrm{p}} = i'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (C1) $\mathcal{L}_1 = \mathcal{L}_1'$, and (D1) $\mathcal{D}_1 = \mathcal{D}_1'$.

Given (D), (Q), (Z), and (A1), by the inductive hypothesis we have (E1) $\{\sigma_2^{\mathrm{p}} = \sigma_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (F1) $\{\Delta_2^{\mathrm{p}} = \Delta_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (G1) $\{n^{\mathrm{p}} = n'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (H1) $\mathcal{L}_2 = \mathcal{L}_2'$, and (I1) $\mathcal{D}_2 = \mathcal{D}_2'$.

Given (E) and (R), by Definition 5.3.3 we have (J1) $\{l^{\mathrm{p}} = l'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (K1) $bty = bty'$.

Given (F), (S), (E1) and (J1), by Definition 5.3.4 we have (L1) $\{\omega^{\mathrm{p}} = \omega'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (G), (T), (K1), and (L1), by Lemma 5.3.26 we have (M1) $\{l_1^{\mathrm{p}} = l_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (H), (U), (E1), and (M1), by Definition 5.3.4 we have (N1) $\{\omega_1^{\mathrm{p}} = \omega_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (O1) $\alpha = \alpha'$.

Given (I), (V), (O1), we have (P1) $j = j'$. Given (I), (V), (K1), (O1), (P1), and (N1), by Lemma 5.3.27 we have (Q1) $\forall j \in \{0...\alpha-1\}\{n_j^{\mathrm{p}} = n_j''^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (J), (W), (B1), (G1), and (Q1), by Axiom 5.3.6 we have (R1) $\{n'^{\mathrm{p}} = n'''^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (K), (X), (P1), (O1), (M1), (E1), (L1), and (R1), by Lemma 5.3.35 we have (S1) $\forall j, j' \in \{0...\alpha-1\}$ such that $j = j'$, $\sigma_{2+j} = \sigma_{2+j'}'$ and (T1) $\sigma_{3+j} = \sigma_{3+j'}'$.

Given (F) and (G), by Lemma 5.3.62 we have accessed locations (U1) $\{(\mathrm{p}, [(l^{\mathrm{p}}, 0)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (H) and (I), by Lemma 5.3.63 we have accessed locations (V1) $\{(\mathrm{p}, [(l_1^{\mathrm{p}}, 0), ..., (l_1^{\mathrm{p}}, \alpha-1)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (K), by Lemma 5.3.67 we have accessed locations (W1) $\{(\mathrm{p}, [(l_1^{\mathrm{p}}, 0), ..., (l_1^{\mathrm{p}}, \alpha-1)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (U1), (V1), and (W1), by Lemmas 5.3.44

and 5.3.46 we have (L).

Given (S) and (T), by Lemma 5.3.62 we have accessed locations (X1) $\{(\mathrm{p}, [(l'^{\mathrm{p}}, 0)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (U) and (V), by Lemma 5.3.63 we have accessed locations (Y1) $\{(\mathrm{p}, [(l_1'^{\mathrm{p}}, 0), ..., (l_1'^{\mathrm{p}}, \alpha' - 1)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (X), by Lemma 5.3.67 we have accessed locations (Z1) $\{(\mathrm{p}, [(l_1'^{\mathrm{p}}, 0), ..., (l_1'^{\mathrm{p}}, \alpha' - 1)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (X1), (Y1), and (Z1), by Lemmas 5.3.44 and 5.3.46 we have (Y).

Given (J1), (M1), (O1), (L), and (Y), by Definition 5.3.10 we have (A2) $\mathcal{L}_3 = \mathcal{L}_3'$. Given (C1), (H1), and (A2), by Lemma 5.3.47 we have (B2) $\mathcal{L}_1 :: \mathcal{L}_2 :: \mathcal{L}_3 = \mathcal{L}_1' :: \mathcal{L}_2' :: \mathcal{L}_3'$.

Given (D1), (I1), and $(\mathrm{ALL}, [mpwa])$, by Lemma 5.3.38 we have (C2) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{ALL}, [mpwa]) = \mathcal{D}_1' :: \mathcal{D}_2' :: (\mathrm{ALL}, [mpwa])$.

Given (T1), (F1), (C2), and (B2), by Definition 5.3.2, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1 \ bop \ e_2) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1 \ bop \ e_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpb])}$
$((1, \gamma_2^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n_3^1) \ \| ... \| \ (\mathrm{q}, \gamma_2^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n_3^{\mathrm{q}}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1 \ bop \ e_2) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1 \ bop \ e_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpb])}$ $((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n_3^1) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n_3^{\mathrm{q}}))$, by SMC^2 rule Multiparty Binary Operation we have $\{(e_1, e_2) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $bop \in \{\cdot, +, -, \div\}$, (B) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n_1^1) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n_1^{\mathrm{q}}))$, (C) $((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, e_2) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, e_2)) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2}$ $((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n_2^1) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n_2^{\mathrm{q}}))$, and (D) $\mathrm{MPC}_b(bop, [n_1^1, ..., n_1^{\mathrm{q}}], [n_2^1, ..., n_2^{\mathrm{q}}]) = (n_3^1, ..., n_3^{\mathrm{q}})$.

Given (E) $\Sigma \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1 \ bop \ e_2) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1 \ bop \ e_2)) \Downarrow^{\mathcal{L}_1'::\mathcal{L}_2'}_{\mathcal{D}_1'::\mathcal{D}_2'::(\mathrm{ALL},[d])} ((1, \gamma^1, \sigma_2'^1, \Delta_2'^1, \mathrm{acc}, n_3'^1) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2'^{\mathrm{q}}, \Delta_2'^{\mathrm{q}}, \mathrm{acc}, n_3'^{\mathrm{q}}))$ and (A), by Lemma 5.2.87 we have (F) $d = mpb$.

Given (E) and (F), by SMC^2 rule Multiparty Binary Operation we have $\{(e_1, e_2) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, $bop \in \{\cdot, +, -, \div\}$,
(G) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1)) \Downarrow^{\mathcal{L}_1'}_{\mathcal{D}_1'} ((1, \gamma^1, \sigma_1'^1, \Delta_1'^1, \mathrm{acc}, n_1'^1) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1'^{\mathrm{q}}, \Delta_1'^{\mathrm{q}}, \mathrm{acc}, n_1'^{\mathrm{q}}))$,
(H) $((1, \gamma^1, \sigma_1'^1, \Delta_1'^1, \mathrm{acc}, e_2) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1'^{\mathrm{q}}, \Delta_1'^{\mathrm{q}}, \mathrm{acc}, e_2)) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((1, \gamma^1, \sigma_2'^1, \Delta_2'^1, \mathrm{acc}, n_2'^1) \ \| ... \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2'^{\mathrm{q}}, \Delta_2'^{\mathrm{q}}, \mathrm{acc}, n_2'^{\mathrm{q}}))$, and (I) $\mathrm{MPC}_b(bop, [n_1'^1, ..., n_1'^{\mathrm{q}}], [n_2'^1, ..., n_2'^{\mathrm{q}}]) = (n_3'^1, ..., n_3'^{\mathrm{q}})$.

Given (B) and (G), by the inductive hypothesis we have (J) $\{\sigma_1^{\mathrm{p}} = \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (K) $\{\Delta_1^{\mathrm{p}} = \Delta_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (L) $\{n_1^{\mathrm{p}} = n_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (M) $\mathcal{D}_1 = \mathcal{D}_1'$, (N) $\mathcal{L}_1 = \mathcal{L}_1'$.

Given (C), (H), (J), and (K), by the inductive hypothesis we have (O) $\{\sigma_2^{\mathrm{p}} = \sigma_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (P) $\{\Delta_2^{\mathrm{p}} = \Delta_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (Q) $\{n_2^{\mathrm{p}} = n_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (R) $\mathcal{D}_2 = \mathcal{D}_2'$, (S) $\mathcal{L}_2 = \mathcal{L}_2'$.

Given (D), (I), (L), and (Q), by Axiom 5.3.7 we have (T) $\{n_3^{\mathrm{p}} = n_3'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (M), (R), and $(\mathrm{ALL}, [mpb])$, by Lemma 5.3.38 we have (U) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{ALL}, [mpb]) = \mathcal{D}_1' :: \mathcal{D}_2' :: (\mathrm{ALL}, [mpb])$.

Given (N) and (S), by Lemma 5.3.47 we have (V) $\mathcal{L}_1 :: \mathcal{L}_2 = \mathcal{L}_1' :: \mathcal{L}_2'$.

Given (O), (P), (T), (U), and (V), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}^1, e_1 \ bop \ e_2) \ \| \ldots \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}^{\mathrm{q}}, e_1 \ bop \ e_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpcmp])}$ $((1, \gamma_2^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}^1, n_3^1) \ \| \ldots \| \ (\mathrm{q}, \gamma_2^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}^{\mathrm{q}}, n_3^{\mathrm{q}}))$

This case is similar to Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e_1 \ bop \ e_2) \ \| \ldots \| \ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e_1 \ bop \ e_2))$ $\Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{ALL},[mpb])} ((1, \gamma_2^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, n_3^1) \ \| \ldots \| \ (\mathrm{q}, \gamma_2^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, n_3^{\mathrm{q}}))$. The main difference is using Axiom 5.3.8 in place of Axiom 5.3.7.

Case $\Pi \triangleright ((\mathrm{p}, \gamma, \ \sigma, \ \Delta, \ \mathrm{acc}, \ e_1 + e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bp])} ((\mathrm{p}, \gamma, \ \sigma_2, \ \Delta_2, \ \mathrm{acc}, \ n_3) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma, \ \sigma, \ \Delta, \ \mathrm{acc}, \ e_1 + e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bp])} ((\mathrm{p}, \gamma, \ \sigma_2, \ \Delta_2, \ \mathrm{acc}, \ n_3) \parallel C_2)$ by SMC^2 rule Public Addition, we have $(e_1, e_2) \nvdash \gamma$, (B) $((\mathrm{p}, \gamma, \ \sigma, \ \Delta, \ \mathrm{acc}, \ e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma, \ \sigma_1, \ \Delta_1, \ \mathrm{acc}, \ n_1) \parallel C_1)$, (C) $((\mathrm{p}, \gamma, \ \sigma_1, \ \Delta_1, \ \mathrm{acc}, \ e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma, \ \sigma_2, \ \Delta_2, \ \mathrm{acc}, \ n_2) \parallel C_2)$, and (D) $n_1 + n_2 = n_3$.

Given (E) $\Sigma \triangleright ((\mathrm{p}, \gamma, \ \sigma, \ \Delta, \ \mathrm{acc}, \ e_1 + e_2) \parallel C) \Downarrow^{\mathcal{L}_1'::\mathcal{L}_2'}_{\mathcal{D}_1'::\mathcal{D}_2'::(\mathrm{p},[d])} ((\mathrm{p}, \gamma, \ \sigma_2', \ \Delta_2', \ \mathrm{acc}, \ n_3') \parallel C_2')$ and (A), by Lemma 5.2.87 we have (F) $d = bp$.

Given (E) and (F), by SMC^2 rule Public Addition, we have $(e_1, e_2) \nvdash \gamma$, (G) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ n'_1) \parallel C'_1)$, (H) $((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ e'_2) \parallel C'_1) \Downarrow^{\mathcal{L}'_2}_{\mathcal{D}'_2} ((\mathrm{p}, \gamma,\ \sigma'_2,\ \Delta'_2,\ \mathrm{acc},\ n'_2) \parallel C'_2)$, and (I) $n'_1 + n'_2 = n'_3$.

Given (B) and (G), by the inductive hypothesis we have (J) $\sigma_1 = \sigma'_1$, (K) $\Delta_1 = \Delta'_1$, (L) $n_1 = n'_1$, (M) $\mathcal{D}_1 = \mathcal{D}'_1$, (N) $\mathcal{L}_1 = \mathcal{L}'_1$, and (O) $C_1 = C'_1$.

Given (C), (H), (J), (K), and (O), by the inductive hypothesis we have (P) $\sigma_2 = \sigma'_2$, (Q) $\Delta_2 = \Delta'_2$, (R) $n_2 = n'_2$, (S) $\mathcal{D}_2 = \mathcal{D}'_2$, (T) $\mathcal{L}_2 = \mathcal{L}'_2$, and (U) $C_2 = C'_2$.

Given (D), (I), (L), and (R), we have (V) $n_3 = n'_3$.

Given (M), (S), and $(\mathrm{p}, [bp])$, by Lemma 5.3.38 we have (W) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [bp]) = \mathcal{D}'_1 :: \mathcal{D}'_2 :: (\mathrm{p}, [bp])$.

Given (N) and (T), by Lemma 5.3.47 we have (X) $\mathcal{L}_1 :: \mathcal{L}_2 = \mathcal{L}'_1 :: \mathcal{L}'_2$.

Given (P), (Q), (U), (V), (W), and (X), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 \cdot e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [bm])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e_1 + e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [bp])} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, n_3) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 - e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [bs])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e_1 + e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [bp])} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, n_3) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 \div e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [bd])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_3) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e_1 + e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[bp])} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, n_3) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$ by SMC^2 rule Public Less Than True, we have $(e_1, e_2) \nvdash \gamma$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1)$, (C) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n_2) \parallel C_2)$, and (D) $(n_1 < n_2) = 1$.

Given (E) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}'_1::\mathcal{L}'_2}_{\mathcal{D}'_1::\mathcal{D}'_2::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_2,\ \Delta'_2,\ \mathrm{acc},\ 1) \parallel C'_2)$ and (A), by Lemma 5.2.87 we have (F) $d = ltt$.

Given (E) and (F), by SMC^2 rule Public Less Than True we have $(e_1, e_2) \nvdash \gamma$, (G) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ n'_1) \parallel C'_1)$, (H) $((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ e_2) \parallel C'_1) \Downarrow^{\mathcal{L}'_2}_{\mathcal{D}'_2} ((\mathrm{p}, \gamma,\ \sigma'_2,\ \Delta_2,\ \mathrm{acc},\ n'_2) \parallel C'_2)$, and (I) $(n'_1 < n'_2) = 1$.

Given (B) and (G), by the inductive hypothesis we have (J) $\sigma_1 = \sigma'_1$, (K) $\Delta_1 = \Delta'_1$, (L) $n_1 = n'_1$, (M) $\mathcal{D}_1 = \mathcal{D}'_1$, (N) $\mathcal{L}_1 = \mathcal{L}'_1$, and (O) $C_1 = C'_1$.

Given (C), (H), (J), (K), and (O), by the inductive hypothesis we have (P) $\sigma_2 = \sigma'_2$, (Q) $\Delta_2 = \Delta'_2$, (R) $n_2 = n'_2$, (S) $\mathcal{D}_2 = \mathcal{D}'_2$, (T) $\mathcal{L}_2 = \mathcal{L}'_2$, and (U) $C_2 = C'_2$.

Given (D), (I), (L), and (R), we have (V) $(n_1 < n_2) = (n'_1 < n'_2) = 1$.

Given (M), (S), and $(\mathrm{p}, [ltt])$, by Lemma 5.3.38 we have (W) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ltt]) = \mathcal{D}'_1 :: \mathcal{D}'_2 :: (\mathrm{p}, [ltt])$.

Given (N) and (T), by Lemma 5.3.47 we have (X) $\mathcal{L}_1 :: \mathcal{L}_2 = \mathcal{L}'_1 :: \mathcal{L}'_2$.

Given (P), (Q), (U), (V), (W), and (X), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltf])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 == e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[eqt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 == e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[eqf])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 ! = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[net])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 ! = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[nef])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 0) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1 < e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ltt])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ 1) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[dv])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[dv])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ by SMC^2 rule Public Declaration, we have $(ty = \mathrm{public}\ bty)$, $\mathrm{acc} = 0$ (B) $l = \phi()$, (C) $\gamma_1 = \gamma[x\ \rightarrow\ (l,\ ty)]$, (D) $\omega = \mathrm{EncodeVal}(ty, \mathrm{NULL})$, and (E) $\sigma_1 = \sigma[l\ \rightarrow\ (\omega,\ ty,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, ty, \mathrm{public}, 1))]$.

Given (F) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[dv])} ((\mathrm{p}, \gamma_1',\ \sigma_1',\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ and (A), by Lemma 5.2.87 we have (G) $d = dv$.

Given (F) and (G), by SMC^2 rule Public Declaration, we have $(ty = \mathrm{public}\ bty)$, $\mathrm{acc} = 0$ (H) $l' = \phi()$, (I)

$\gamma_1' = \gamma[x \ \rightarrow \ (l', \ ty)]$, (J) $\omega' = \text{EncodeVal}(ty, \text{NULL})$, and (K) $\sigma_1' = \sigma[l' \ \rightarrow \ (\omega', \ ty, \ 1, \ \text{PermL}(\text{Freeable}, ty, \text{public}, 1))]$.

Given (B) and (H), by Axiom 5.3.4 we have (L) $l = l'$.

Given (C), (I), and (L), by Definition 5.3.3 we have (M) $\gamma_1 = \gamma_1'$.

Given (D) and (J), by Lemma 5.3.30 we have (N) $\omega = \omega'$.

Given (E), (K), (L), and (N), by Definition 5.3.4 we have (O) $\sigma_1 = \sigma_1'$.

Given (E), by Lemma 5.3.51 we have accessed (P) $(\text{p}, [(l, 0)])$. Given (K), by Lemma 5.3.51 we have accessed (Q) $(\text{p}, [(l', 0)])$. Given (P), (Q), and (L), we have (R) $(\text{p}, [(l, 0)]) = (\text{p}, [(l', 0)])$.

Given (A), (F), and (G) we have (S) $(\text{p}, [dv])$ = $(\text{p}, [dv])$.

Given (M), (O), (R), and (S), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma, \ \sigma, \ \Delta, \ \text{acc}, \ ty \ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[d1])} ((\text{p}, \gamma_1, \ \sigma_1, \ \Delta, \ \text{acc}, \ \text{skip}) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\text{p}, \gamma, \ \sigma, \ \Delta, \ \text{acc}, \ ty \ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[dv])} ((\text{p}, \gamma_1, \ \sigma_1, \ \Delta, \ \text{acc}, \ \text{skip}) \parallel C)$.

Case $\Pi \triangleright ((\text{p}, \gamma, \ \sigma, \ \Delta, \ \text{acc}, \ s_1; \ s_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[ss])} ((\text{p}, \gamma_2, \ \sigma_2, \ \Delta_2, \ \text{acc}, \ v_2) \parallel C_2)$

Given (A) $\Pi \triangleright ((\text{p}, \gamma, \ \sigma, \ \Delta, \ \text{acc}, \ s_1; \ s_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[ss])} ((\text{p}, \gamma_2, \ \sigma_2, \ \Delta_2, \ \text{acc}, \ v_2) \parallel C_2)$ by SMC^2 rule Statement Sequencing, we have (B) $((\text{p}, \gamma, \ \sigma, \ \Delta, \ \text{acc}, \ s_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma_1, \ \sigma_1, \ \Delta_1, \ \text{acc}, \ v_1) \parallel C_1)$, and (C) $((\text{p}, \gamma_1, \ \sigma_1, \ \Delta_1, \ \text{acc}, \ s_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\text{p}, \gamma_2, \ \sigma_2, \ \Delta_2, \ \text{acc}, \ v_2) \parallel C_2)$.

Given (D) $\Sigma \triangleright ((\text{p}, \gamma, \ \sigma, \ \Delta, \ \text{acc}, \ s_1; \ s_2) \parallel C) \Downarrow^{\mathcal{L}_1'::\mathcal{L}_2'}_{\mathcal{D}_1'::\mathcal{D}_2'::(\text{p},[ss])} ((\text{p}, \gamma_2', \ \sigma_2', \ \Delta_2', \ \text{acc}, \ v_2') \parallel C_2')$ and (A), by Lemma 5.2.87 we have (E) $d = ss$.

Given (D) and (E) by SMC^2 rule Statement Sequencing, we have (F) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ s_1) \parallel C) \Downarrow^{\mathcal{L}_1'}_{\mathcal{D}_1'} ((\mathrm{p}, \gamma_1',\ \sigma_1',\ \Delta_1',\ \mathrm{acc},\ v_1') \parallel C_1')$, and (G) $((\mathrm{p}, \gamma_1',\ \sigma_1',\ \Delta_1',\ \mathrm{acc},\ s_2) \parallel C_1') \Downarrow^{\mathcal{L}_2'}_{\mathcal{D}_2'} ((\mathrm{p}, \gamma_2',\ \sigma_2',\ \Delta_2',\ \mathrm{acc},\ v_2') \parallel C_2')$.

Given (B) and (F), by the inductive hypothesis we have (H) $\gamma_1 = \gamma_1'$, (I) $\sigma_1 = \sigma_1'$, (J) $\Delta_1 = \Delta_1'$, (K) $v_1 = v_1'$, (L) $\mathcal{D}_1 = \mathcal{D}_1'$, (M) $\mathcal{L}_1 = \mathcal{L}_1'$, and (N) $C_1 = C_1'$.

Given (C), (G), (H), (I), (J), and (N), by the inductive hypothesis we have (O) $\gamma_2 = \gamma_2'$, (P) $\sigma_2 = \sigma_2'$, (Q) $\Delta_2 = \Delta_2'$, (R) $v_2 = v_2'$, (S) $\mathcal{D}_2 = \mathcal{D}_2'$, (T) $\mathcal{L}_2 = \mathcal{L}_2'$, and (U) $C_2 = C_2'$.

Given (L), (S), and $(\mathrm{p}, [ss])$, by Lemma 5.3.38 we have (V) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ss]) = \mathcal{D}_1' :: \mathcal{D}_2' :: (\mathrm{p}, [ss])$.

Given (M) and (T), by Lemma 5.3.47 we have (W) $\mathcal{L}_1 :: \mathcal{L}_2 = \mathcal{L}_1' :: \mathcal{L}_2'$.

Given (O), (P), (Q), (R), (U), (V), and (W), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \{s\}) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\mathrm{p},[sb])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \{s\}) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\mathrm{p},[sb])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Statement Block, we have (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ s) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v) \parallel C_1)$.

Given (C) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \{s\}) \parallel C) \Downarrow^{\mathcal{L}_1'}_{\mathcal{D}_1'::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma_1',\ \Delta_1',\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1')$ and (A), by Lemma 5.2.87 we have (D) $d = sb$.

Given (C) and (D), by SMC^2 rule Statement Block, we have (E) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ s) \parallel C) \Downarrow^{\mathcal{L}_1'}_{\mathcal{D}_1'} ((\mathrm{p}, \gamma_1',\ \sigma_1',\ \Delta_1',\ \mathrm{acc},\ v') \parallel C_1')$.

Given (B) and (E), by the inductive hypothesis we have (F) $\gamma_1 = \gamma_1'$, (G) $\sigma_1 = \sigma_1'$, (H) $\Delta_1 = \Delta_1'$, (I) $v_1 = v_1'$, (J) $\mathcal{D}_1 = \mathcal{D}_1'$, (K) $\mathcal{L}_1 = \mathcal{L}_1'$, and (L) $C_1 = C_1'$.

Given (J) and $(\mathrm{p}, [sb])$, by Lemma 5.3.38 we have (M) $\mathcal{D}_1 :: (\mathrm{p}, [sb]) = \mathcal{D}_1' :: (\mathrm{p}, [sb])$.

Given (G), (H), (L), (K), and (M), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (e)) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\mathrm{p},[ep])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (e)) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\mathrm{p},[ep])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v) \parallel C_1)$ by SMC^2 rule Parentheses, we have (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ v) \parallel C_1)$.

Given (C) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (e)) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ v') \parallel C'_1)$ and (A), by Lemma 5.2.87 we have (D) $d = ep$.

Given (C) and (D), by SMC^2 rule Parentheses, we have (E) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ v')$ $\parallel C'_1)$.

Given (B) and (E), by the inductive hypothesis we have (F) $\sigma_1 = \sigma'_1$, (G) $\Delta_1 = \Delta'_1$, (H) $v = v'$, (I) $\mathcal{D}_1 = \mathcal{D}'_1$, (J) $\mathcal{L}_1 = \mathcal{L}'_1$, and (K) $C_1 = C'_1$.

Given (I) and $(\mathrm{p}, [ep])$, by Lemma 5.3.38 we have (L) $\mathcal{D}_1 :: (\mathrm{p}, [ep]) = \mathcal{D}'_1 :: (\mathrm{p}, [ep])$.

Given (F), (G), (H), (J), (K), and (L), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ds])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ds])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$ by SMC^2 rule Declaration Assignment, we have (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$, and (C) $((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ x = e) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma_1,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Given (D) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x = e) \parallel C) \Downarrow^{\mathcal{L}'_1::\mathcal{L}'_2}_{\mathcal{D}'_1::\mathcal{D}'_2::(\mathrm{p},[d])} ((\mathrm{p}, \gamma'_1,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_2)$ and (A), by Lemma 5.2.87 we have (E) $d = ds$.

Given (D) and (E) by SMC^2 rule Declaration Assignment, we have (F) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma'_1,\ \sigma'_1,$ $\Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_1)$, and (G) $((\mathrm{p}, \gamma'_1,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ x = e) \parallel C'_1) \Downarrow^{\mathcal{L}'_2}_{\mathcal{D}'_2} ((\mathrm{p}, \gamma'_1,\ \sigma'_2,\ \Delta'_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_2)$.

Given (B) and (F), by the inductive hypothesis we have (H) $\gamma_1 = \gamma_1'$, (I) $\sigma_1 = \sigma_1'$, (J) $\Delta_1 = \Delta_1'$, (K) $\mathcal{D}_1 = \mathcal{D}_1'$, (L) $\mathcal{L}_1 = \mathcal{L}_1'$, and (M) $C_1 = C_1'$.

Given (C), (G), (H), (I), (J), and (N), by the inductive hypothesis we have (N) $\sigma_2 = \sigma_2'$, (O) $\Delta_2 = \Delta_2'$, (P) $\mathcal{D}_2 = \mathcal{D}_2'$, (Q) $\mathcal{L}_2 = \mathcal{L}_2'$, and (R) $C_2 = C_2'$.

Given (K), (P), and $(\mathrm{p}, [ds])$, by Lemma 5.3.38 we have (S) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [ds]) = \mathcal{D}_1' :: \mathcal{D}_2' :: (\mathrm{p}, [ds])$.

Given (L) and (Q), by Lemma 5.3.47 we have (T) $\mathcal{L}_1 :: \mathcal{L}_2 = \mathcal{L}_1' :: \mathcal{L}_2'$.

Given (H), (N), (O), (S), and (T), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x[e_1] = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[das])} ((\mathrm{p}, \gamma_1,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ds])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[r])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ v) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[r])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ v) \parallel C)$ by SMC^2 rule Read Public Variable, we have (B) $\gamma(x) = (l,\ \text{public}\ bty)$, (C) $\sigma(l) = (\omega,\ \text{public}\ bty,\ 1,\ \mathrm{PermL}(\text{Freeable}, \text{public}\ bty, \text{public}, 1))$, and (D) $\mathrm{DecodeVal}(\text{public}\ bty,\ \omega) = v$.

Given (E) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l',0)])}_{(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ v') \parallel C)$ and (A), by Lemma 5.2.87 we have (F) $d = r$.

Given (E) and (F), by SMC^2 rule Read Public Variable, we have (G) $\gamma(x) = (l',\ \text{public}\ bty')$, (H) $\sigma(l') = (\omega',\ \text{public}\ bty',\ 1,\ \mathrm{PermL}(\text{Freeable}, \text{public}\ bty', \text{public}, 1))$, and (I) $\mathrm{DecodeVal}(\text{public}\ bty',\ \omega') = v'$.

Given (B) and (G), by Definition 5.3.3 we have (J) $l = l'$, and (K) $bty = bty'$.

Given (C), (H), and (J), by Definition 5.3.4 we have (L) $\omega = \omega'$.

Given (D), (I), (K), and (L), by Lemma 5.3.29 we have (M) $v = v'$.

Given (C) and (D), by Lemma 5.3.64 we have accessed location $(\mathrm{p}, [(l, 0)])$. Given (H) and (I), by Lemma 5.3.64 we have accessed location $(\mathrm{p}, [(l', 0)])$. Given (J), we have (N) $(\mathrm{p}, [(l, 0)]) = (\mathrm{p}, [(l', 0)])$.

Given (A), (E), (F), (M), and (N), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[r1])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ v) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[r])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ v) \parallel C)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[w])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[w])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Write Public Variable, we have $(e) \nvdash \gamma$, (B) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1)$, (C) $\gamma(x) = (l,\ \mathrm{public}\ bty)$, and (D) $\mathrm{UpdateVal}(\sigma_1,\ l,\ n,\ \mathrm{public}\ bty) = \sigma_2$.

Given (E) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}'_1::(\mathrm{p},[(l',0)])}_{\mathcal{D}'_1::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_1)$ and (A), by Lemma 5.2.87 we have (F) $d = w$.

Given (E) and (F), by SMC^2 rule Write Public Variable, we have $(e) \nvdash \gamma$, (G) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma, \sigma'_1, \Delta'_1, \mathrm{acc}, n') \parallel C'_1)$, (H) $\gamma(x) = (l',\ \mathrm{public}\ bty')$, and (I) $\mathrm{UpdateVal}(\sigma'_1,\ l',\ n',\ \mathrm{public}\ bty') = \sigma'_2$.

Given (B) and (G), by the inductive hypothesis we have (J) $\sigma_1 = \sigma'_1$, (K) $\Delta_1 = \Delta'_1$, (L) $n = n'$, (M) $\mathcal{D}_1 = \mathcal{D}'_1$, (N) $\mathcal{L}_1 = \mathcal{L}'_1$, and (O) $C_1 = C'_1$.

Given (C) and (H), by Definition 5.3.3 we have (P) $l = l'$ and (Q) $bty = bty'$.

Given (D), (I), (J), (P), (L), and (Q), by Lemma 5.3.34 we have (R) $\sigma_2 = \sigma'_2$.

Given (M) and $(\mathrm{p}, [w])$, by Lemma 5.3.38 we have (S) $\mathcal{D}_1 :: (\mathrm{p}, [w]) = \mathcal{D}_1' :: (\mathrm{p}, [w])$.

Given (D), by Lemma 5.3.66 we have accessed location $(\mathrm{p}, [(l, 0)])$. Given (I), by Lemma 5.3.66 we have accessed location $(\mathrm{p}, [(l', 0)])$. Given (P), we have (T) $(\mathrm{p}, [(l, 0)]) = (\mathrm{p}, [(l', 0)])$. Given (N) and (T), by Lemma 5.3.47 we have (U) $\mathcal{L}_1 :: (\mathrm{p}, [(l, 0)]) = \mathcal{L}_1' :: (\mathrm{p}, [(l', 0)])$.

Given (R), (K), (O), (S), and (U), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [w1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [w])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [w2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [w2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Write Private Variable Public Value, we have $(e) \nvdash \gamma$, (B) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1)$, (C) $\gamma(x) = (l,\ \mathrm{public}\ bty)$, and (D) $\mathrm{UpdateVal}(\sigma_1,\ l,\ \mathrm{encrypt}(n),\ \mathrm{public}\ bty) = \sigma_2$.

Given (E) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1' :: (\mathrm{p}, [(l',0)])}_{\mathcal{D}_1' :: (\mathrm{p}, [d])} ((\mathrm{p}, \gamma,\ \sigma_2',\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1')$ and (A), by Lemma 5.2.87 we have (F) $d = w2$.

Given (E) and (F), by SMC^2 rule Write Private Variable Public Value, we have $(e) \nvdash \gamma$, (G) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1'}_{\mathcal{D}_1'} ((\mathrm{p}, \gamma, \sigma_1', \Delta_1', \mathrm{acc}, n') \parallel C_1')$, (H) $\gamma(x) = (l',\ \mathrm{public}\ bty')$, and (I) $\mathrm{UpdateVal}(\sigma_1',\ l',\ \mathrm{encrypt}(n'),\ \mathrm{public}\ bty') = \sigma_2'$.

Given (B) and (G), by the inductive hypothesis we have (J) $\sigma_1 = \sigma_1'$, (K) $\Delta_1 = \Delta_1'$, (L) $n = n'$, (M) $\mathcal{D}_1 = \mathcal{D}_1'$, (N) $\mathcal{L}_1 = \mathcal{L}_1'$, and (O) $C_1 = C_1'$.

Given (C) and (H), by Definition 5.3.3 we have (P) $l = l'$ and (Q) $bty = bty'$.

Given (L) and $\text{encrypt}(n)$ and $\text{encrypt}(n')$, by Axiom 5.3.1 we have (R) $\text{encrypt}(n) = \text{encrypt}(n')$.

Given (D), (I), (J), (P), (R), and (Q), by Lemma 5.3.34 we have (S) $\sigma_2 = \sigma_2'$.

Given (M) and $(\text{p}, [w])$, by Lemma 5.3.38 we have (T) $\mathcal{D}_1 :: (\text{p}, [w]) = \mathcal{D}_1' :: (\text{p}, [w])$.

Given (D), by Lemma 5.3.66 we have accessed location $(\text{p}, [(l, 0)])$. Given (I), by Lemma 5.3.66 we have accessed location $(\text{p}, [(l', 0)])$. Given (P), we have (U) $(\text{p}, [(l, 0)]) = (\text{p}, [(l', 0)])$. Given (N) and (U), by Lemma 5.3.47 we have (V) $\mathcal{L}_1 :: (\text{p}, [(l, 0)]) = \mathcal{L}_1' :: (\text{p}, [(l', 0)])$.

Given (S), (K), (O), (T), and (V), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{smcinput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[inp])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{skip}) \parallel C_2)$

Given (A) $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{smcinput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\text{p},[inp])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \text{acc},\ \text{skip}) \parallel C_2)$ by SMC^2 rule SMC Input Public Value, we have $(e) \nvdash \gamma$, $\text{acc} = 0$, (B) $((\text{p}, \gamma, \sigma, \Delta, \text{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, n) \parallel C_1)$, (C) $\gamma(x) = (l, \text{public } bty)$, (D) $\text{InputValue}(x, n) = n_1$, (E) $((\text{p}, \gamma, \sigma_1, \Delta_1, \text{acc}, x = n_1) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\text{p}, \gamma, \sigma_2, \Delta_2, \text{acc}, \text{skip}) \parallel C_2)$.

Given (F) $\Sigma \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ \text{smcinput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1'::\mathcal{L}_2'}_{\mathcal{D}_1'::\mathcal{D}_2'::(\text{p},[d])} ((\text{p}, \gamma,\ \sigma_2',\ \Delta_2',\ \text{acc},\ \text{skip}) \parallel C_2')$ and (A), by Lemma 5.2.87 we have (G) $d = inp$.

Given (F) and (G), by SMC^2 rule SMC Input Public Value, we have $(e) \nvdash \gamma$, $\text{acc} = 0$, (H) $((\text{p}, \gamma, \sigma, \Delta, \text{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1'}_{\mathcal{D}_1'} ((\text{p}, \gamma, \sigma_1', \Delta_1', \text{acc}, n') \parallel C_1')$, (I) $\gamma(x) = (l', \text{public } bty')$, (J) $\text{InputValue}(x, n') = n_1'$, (K) $((\text{p}, \gamma, \sigma_1', \Delta_1', \text{acc}, x = n_1') \parallel C_1') \Downarrow^{\mathcal{L}_2'}_{\mathcal{D}_2'} ((\text{p}, \gamma, \sigma_2', \Delta_2', \text{acc}, \text{skip}) \parallel C_2')$.

Given (B) and (H), by the inductive hypothesis we have (L) $\sigma_1 = \sigma_1'$, (M) $\Delta_1 = \Delta_1'$, (N) $n = n'$, (O) $\mathcal{D}_1 = \mathcal{D}_1'$, (P) $\mathcal{L}_1 = \mathcal{L}_1'$, and (Q) $C_1 = C_1'$.

Given (C) and (I), by Definition 5.3.3 we have (R) $l = l'$, and (S) $bty = bty'$.

Given (D), (J), and (N), by Axiom 5.3.2 we have (T) $n_1 = n_1'$.

Given (E), (K), (L), (M), (Q), and (T), by the inductive hypothesis we have (U) $\sigma_2 = \sigma'_2$, (V) $\Delta_2 = \Delta'_2$, (W) $\mathcal{D}_2 = \mathcal{D}'_2$, (X) $\mathcal{L}_2 = \mathcal{L}'_2$, and (Y) $C_2 = C'_2$.

Given (O), (W), and $(\mathrm{p}, [inp])$, by Lemma 5.3.38 we have (Z) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [inp]) = \mathcal{D}'_1 :: \mathcal{D}'_2 :: (\mathrm{p}, [inp])$.

Given (P) and (X), by Lemma 5.3.47 we have (A1) $\mathcal{L}_1 :: \mathcal{L}_2 = \mathcal{L}'_1 :: \mathcal{L}'_2$.

Given (U), (V), (Y), (Z), and (A1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[inp2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[inp])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e_1,\ e_1)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[inp3])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_3)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e_1,\ e_1)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[inp3])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_3)$ by SMC^2 rule SMC Input Private Array, we have $(e_1, e_2) \nvdash \gamma$, $\mathrm{acc} = 0$, (B) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1)$, (C) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, \alpha) \parallel C_2)$, (D) $\gamma(x) = (l, \text{private const } bty*)$, (E) $\mathrm{InputArray}(x, n, \alpha) = [m_0, ..., m_\alpha]$, and (F) $((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, x = [m_0, ..., m_\alpha]) \parallel C_2) \Downarrow^{\mathcal{L}_3}_{\mathcal{D}_3} ((\mathrm{p}, \gamma, \sigma_3, \Delta_3, \mathrm{acc}, \mathrm{skip}) \parallel C_3)$.

Given (G) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e_1,\ e_1)) \parallel C) \Downarrow^{\mathcal{L}'_1::\mathcal{L}'_2::\mathcal{L}'_3}_{\mathcal{D}'_1::\mathcal{D}'_2::\mathcal{D}'_3::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_3,\ \Delta'_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_3)$ and (A), by Lemma 5.2.87 we have (H) $d = inp3$.

Given (G) and (H), by SMC^2 rule SMC Input Private Array, we have $(e_1, e_2) \nvdash \gamma$, $\mathrm{acc} = 0$, (I) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e_1) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma, \sigma'_1, \Delta'_1, \mathrm{acc}, n') \parallel C'_1)$, (J) $((\mathrm{p}, \gamma, \sigma'_1, \Delta'_1, \mathrm{acc}, e_2) \parallel C'_1) \Downarrow^{\mathcal{L}'_2}_{\mathcal{D}'_2} ((\mathrm{p}, \gamma, \sigma'_2, \Delta'_2, \mathrm{acc}, \alpha') \parallel C'_2)$, (K) $\gamma(x) = (l', \text{private const } bty'*)$, (L) $\mathrm{InputArray}(x, n', \alpha') = [m'_0, ..., m'_{\alpha'}]$, and (M) $((\mathrm{p}, \gamma, \sigma'_2, \Delta'_2, \mathrm{acc}, x = [m'_0, ..., m'_{\alpha'}]) \parallel C'_2) \Downarrow^{\mathcal{L}'_3}_{\mathcal{D}'_3} ((\mathrm{p}, \gamma, \sigma'_3, \Delta'_3, \mathrm{acc}, \mathrm{skip}) \parallel C'_3)$.

Given (B) and (I), by the inductive hypothesis we have (N) $\sigma_1 = \sigma_1'$, (O) $\Delta_1 = \Delta_1'$, (P) $n = n'$, (Q) $\mathcal{D}_1 = \mathcal{D}_1'$, (R) $\mathcal{L}_1 = \mathcal{L}_1'$, and (S) $C_1 = C_1'$.

Given (C), (J), (N), (O), and (S), by the inductive hypothesis we have (T) $\sigma_2 = \sigma_2'$, (U) $\Delta_2 = \Delta_2'$, (V) $\alpha = \alpha'$, (W) $\mathcal{D}_2 = \mathcal{D}_2'$, (X) $\mathcal{L}_2 = \mathcal{L}_2'$, and (Y) $C_2 = C_2'$.

Given (D) and (K), by Definition 5.3.3 we have (Z) $l = l'$, and (A1) $bty = bty'$.

Given (E), (L), (P), and (V), by Axiom 5.3.3 we have (B1) $[m_0, ..., m_{n_1}] = [m_0', ..., m_{n_1'}']$.

Given (F), (M), (T), (U), (Y), and (B1), by the inductive hypothesis we have (C1) $\sigma_3 = \sigma_3'$, (D1) $\Delta_3 = \Delta_3'$, (E1) $\mathcal{D}_3 = \mathcal{D}_3'$, (F1) $\mathcal{L}_3 = \mathcal{L}_3'$, and (G1) $C_3 = C_3'$.

Given (O), (W), (E1) and $(\mathrm{p}, [inp3])$, by Lemma 5.3.38 we have (H1) $\mathcal{D}_1 :: \mathcal{D}_2 :: \mathcal{D}_3 :: (\mathrm{p}, [inp3]) = \mathcal{D}_1' :: \mathcal{D}_2' :: \mathcal{D}_3' :: (\mathrm{p}, [inp3])$.

Given (R), (X), and (F1), by Lemma 5.3.47 we have (I1) $\mathcal{L}_1 :: \mathcal{L}_2 :: \mathcal{L}_3 = \mathcal{L}_1' :: \mathcal{L}_2' :: \mathcal{L}_3'$.

Given (C1), (D1), (G1), (H1), and (I1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[inp1])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_3)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcinput}(x,\ e_1,\ e_1)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[inp3])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc}, \mathrm{skip}) \parallel C_3)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[out])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[out])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule SMC Output Public Value, we have $(e) \nvdash \gamma$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$, (C) $\gamma(x) = (l, \mathrm{public}\ bty)$, (D) $\sigma_1(l) = (\omega,\ \mathrm{public}\ bty,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{public}\ bty, \mathrm{public}, 1))$, (E) $\mathrm{DecodeVal}(\mathrm{public}\ bty,\ \omega) = n_1$, and (F) $\mathrm{OutputValue}(x, n, n_1)$.

Given (G) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1' :: (\mathrm{p}, [(l', 0)])}_{\mathcal{D}_1' :: (\mathrm{p}, [d])} ((\mathrm{p}, \gamma,\ \sigma_1',\ \Delta_1',\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1')$ and (A), by Lemma 5.2.87 we have (H) $d = out$.

Given (G) and (H), by SMC^2 rule SMC Output Public Value, we have $(e) \nvdash \gamma$, (I) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1'}_{\mathcal{D}_1'} ((\mathrm{p}, \gamma,\ \sigma_1',\ \Delta_1',\ \mathrm{acc},\ n') \parallel C_1')$, (J) $\gamma(x) = (l', \mathrm{public}\ bty')$, (K) $\sigma_1'(l') = (\omega',\ \mathrm{public}\ bty',\ 1,\ \mathrm{PermL}(\mathrm{Freeable},\ \mathrm{public}\ bty', \mathrm{public}, 1))$, (L) $\mathrm{DecodeVal}(\mathrm{public}\ bty',\ \omega') = n_1'$, and (M) $\mathrm{OutputValue}(x, n', n_1')$.

Given (B) and (I), by the inductive hypothesis we have (N) $\sigma_1 = \sigma_1'$, (O) $\Delta_1 = \Delta_1'$, (P) $n = n'$, (Q) $\mathcal{D}_1 = \mathcal{D}_1'$, (R) $\mathcal{L}_1 = \mathcal{L}_1'$, and (S) $C_1 = C_1'$.

Given (C) and (J), by Definition 5.3.3 we have (T) $l = l'$, and (U) $bty = bty'$.

Given (D), (K), (N), and (T), by Definition 5.3.4 we have (V) $\omega = \omega'$.

Given (E), (L), (U), and (V), by Lemma 5.3.29 we have (W) $n_1 = n_1'$.

Given (F), (M), (P), and (W), by Lemma 5.3.1 we have identical output going to the same parties.

Given (Q) and $(\mathrm{p}, [out])$, by Lemma 5.3.38 we have (X) $\mathcal{D}_1 :: (\mathrm{p}, [out]) = \mathcal{D}_1' :: (\mathrm{p}, [out])$.

Given (D) and (E), by Lemma 5.3.64 we have accessed location $(\mathrm{p}, [(l, 0)])$. Given (K) and (L), by Lemma 5.3.64 we have accessed location $(\mathrm{p}, [(l', 0)])$. Given (R) and (T), by Lemma 5.3.47 we have (Y) $\mathcal{L}_1 :: (\mathrm{p}, [(l, 0)]) = \mathcal{L}_1' :: (\mathrm{p}, [(l', 0)])$.

Given (N), (O), (S), (X), and (Y), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l, 0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [out2])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e)) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l, 0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [out])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[out3])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[out3])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C_2)$ by SMC^2 rule SMC Output Private Array, we have $(e_1, e_2) \nvdash \gamma$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$, (C) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma, \sigma_2, \Delta_2, \mathrm{acc}, \alpha) \parallel C_2)$, (D) $\gamma(x) = (l, \text{private const } bty*)$, (E) $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \mathrm{PermL}(\text{Freeable, private const } bty*, \text{private}, 1))$, (F) $\mathrm{DecodePtr}(\text{private const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ \text{private } bty,\ 1]$, (G) $\sigma_2(l_1) = (\omega_1,\ \text{private } bty,\ \alpha,\ \mathrm{PermL}(\text{Freeable, private } bty, \text{private}, \alpha))$, (H) $\forall i \in \{0, ..., \alpha - 1\} \quad \mathrm{DecodeArr}(\text{private } bty,\ i,\ \omega_1) = m_i$, and (I) $\mathrm{OutputArray}(x,\ n,\ [m_0,\ ...,\ m_{\alpha-1}])$.

Given (J) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}'_1::\mathcal{L}'_2::(\mathrm{p},[(l',0),(l'_1,0),...,(l'_1,\alpha-1)])}_{\mathcal{D}'_1::\mathcal{D}'_2::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_2,\ \Delta'_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C'_2)$ and (A), by Lemma 5.2.87 we have (K) $d = out3$.

Given (J) and (K), by SMC^2 rule SMC Output Private Array, we have $(e_1, e_2) \nvdash \gamma$, (L) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1)\text{-} \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ n') \parallel C'_1)$, (M) $((\mathrm{p}, \gamma, \sigma'_1, \Delta'_1, \mathrm{acc}, e_2) \parallel C'_1) \Downarrow^{\mathcal{L}'_2}_{\mathcal{D}'_2} ((\mathrm{p}, \gamma, \sigma'_2, \Delta'_2, \mathrm{acc}, \alpha') \parallel C'_2)$, (N) $\gamma(x) = (l', \text{private const } bty'*)$, (O) $\sigma'_2(l') = (\omega',\ \text{private const } bty'*,\ 1,\ \mathrm{PermL}(\text{Freeable, private const } bty'*, \text{private}, 1))$, (P) $\mathrm{DecodePtr}(\text{private const } bty'*,\ 1,\ \omega') = [1,\ [(l'_1, 0)],\ [1],\ \text{private } bty',\ 1]$, (Q) $\sigma'_2(l'_1) = (\omega'_1, \text{private } bty',\ \alpha',\ \mathrm{PermL}(\text{Freeable, private } bty', \text{private}, \alpha'))$, (R) $\forall i' \in \{0, ..., \alpha' - 1\}$ $\mathrm{DecodeArr}(\text{private } bty',\ i',\ \omega'_1) = m'_{i'}$, and (S) $\mathrm{OutputArray}(x,\ n',\ [m'_0,\ ...,\ m'_{\alpha'-1}])$.

Given (B) and (L), by the inductive hypothesis we have (T) $\sigma_1 = \sigma'_1$, (U) $\Delta_1 = \Delta'_1$, (V) $n = n'$, (W) $\mathcal{D}_1 = \mathcal{D}'_1$, (X) $\mathcal{L}_1 = \mathcal{L}'_1$, and (Y) $C_1 = C'_1$.

Given (C) and (M), by the inductive hypothesis we have (Z) $\sigma_2 = \sigma'_2$, (A1) $\Delta_2 = \Delta'_2$, (B1) $\alpha = \alpha'$, (C1) $\mathcal{D}_2 = \mathcal{D}'_2$, (D1) $\mathcal{L}_2 = \mathcal{L}'_2$, and (E1) $C_2 = C'_2$.

Given (D) and (N), by Definition 5.3.3 we have (F1) $l = l'$, and (G1) $bty = bty'$.

Given (E), (O), (Z), and (F1), by Definition 5.3.4 we have (H1) $\omega = \omega'$.

Given (F), (P), (G1), and (H1), by Lemma 5.3.26 we have (I1) $l_1 = l'_1$.

Given (G), (Q), (Z), and (I1), by Definition 5.3.4 we have (J1) $\omega_1 = \omega_1'$ and (K1) $\alpha = \alpha'$.

Given (R), (H), (K1), we have $i = i'$. Given (G1) and (J1), by Lemma 5.3.27 we have (L1) $\forall i \in \{0...\alpha - 1\} m_i = m_i'$.

Given (I), (S), (V), (K1), and (L1), by Lemma 5.3.2 we have identical output going to the same parties.

Given (W), (C1) and $(\mathrm{p}, [out3])$, by Lemma 5.3.38 we have (M1) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [out3]) = \mathcal{D}_1' :: \mathcal{D}_2' :: (\mathrm{p}, [out3])$.

Given (E) and (F), by Lemma 5.3.62 we have accessed locations (N1) $(\mathrm{p}, [(l, 0)])$. Given (G) and (H), by Lemma 5.3.63 we have accessed locations (O1) $(\mathrm{p}, [(l_1, 0), ..., (l_1, \alpha - 1)])$. Given (N1) and (O1), by Lemmas 5.3.44 and 5.3.45 we have (P1) $(\mathrm{p}, [(l, 0), (l_1, 0), ..., (l_1, \alpha - 1)])$ Given (O) and (P), by Lemma 5.3.62 we have accessed locations (Q1) $(\mathrm{p}, [(l', 0)])$. Given (Q) and (R), by Lemma 5.3.63 we have accessed locations (R1) $(\mathrm{p}, [(l_1', 0), ..., (l_1', \alpha' - 1)])$. Given (Q1) and (R1), by Lemmas 5.3.44 and 5.3.45 we have (S1) $(\mathrm{p}, [(l', 0), (l_1', 0), ..., (l_1', \alpha' - 1)])$.

Given (X), (D1), (F1), (I1), (P1), and (S1) by Lemma 5.3.47 we have (T1) $\mathcal{L}_1 :: \mathcal{L}_2 :: (\mathrm{p}, [(l, 0), (l_1, 0), ..., (l_1, \alpha - 1)]) = \mathcal{L}_1' :: \mathcal{L}_2' :: (\mathrm{p}, [(l', 0), (l_1', 0), ..., (l_1', \alpha' - 1)])$.

Given (Z), (A1), (E1), (M1), and (T1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[out1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{smcoutput}(x,\ e_1,\ e_2)) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[out3])}$ $((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x(\overline{p})\{s\}) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[fpd])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x(\overline{p})\{s\}) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[fpd])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ by SMC^2 rule Function Definition, we have $\mathrm{acc} = 0$, $x \notin \gamma$, (B) $l = \phi()$, (C) $\mathrm{GetFunTypeList}(\overline{p}) = \overline{ty}$, (D) $\gamma_1 = \gamma[x \rightarrow (l,\ \overline{ty} \rightarrow ty)]$, (E) $\mathrm{CheckPublicEffects}(s,\ x,\ \gamma, \sigma) = n$, (F) $\mathrm{EncodeFun}(s,\ n,\ \overline{p}) = \omega$, and (G) $\sigma_1 = \sigma[l \rightarrow (\omega,\ \overline{ty} \rightarrow$

$ty,\ 1,\ \text{PermL_Fun(public)})]$.

Given (H) $\Sigma \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x(\overline{p})\{s\})\ \|\ C) \Downarrow^{(\text{p},[(l',0)])}_{(\text{p},[d])} ((\text{p}, \gamma_1',\ \sigma_1',\ \Delta,\ \text{acc},\ \text{skip})\ \|\ C)$ and (A), by Lemma 5.2.87 we have (I) $d = fpd$.

Given (H) and (I), by SMC^2 rule Function Definition, we have $\text{acc} = 0$, $x \notin \gamma$, (J) $l' = \phi()$, (K) $\text{GetFunTypeList}(\overline{p}) = \overline{ty}'$, (L) $\gamma_1' = \gamma[x \rightarrow (l',\ \overline{ty}' \rightarrow ty)]$, (M) $\text{CheckPublicEffects}(s,\ x,\ \gamma, \sigma) = n'$, (N) $\text{EncodeFun}(s,\ n',\ \overline{p}) = \omega'$, and (O) $\sigma_1' = \sigma[l' \rightarrow (\omega',\ \overline{ty}' \rightarrow ty,\ 1,\ \text{PermL_Fun(public)})]$.

Given (B) and (J), by Axiom 5.3.4 we have (P) $l = l'$.

Given (C) and (K), by Lemma 5.3.3 we have (Q) $\overline{ty} = \overline{ty}'$.

Given (D), (L), (P), and (Q), by Definition 5.3.3 we have (R) $\gamma_1 = \gamma_1'$.

Given (E) and (M), by Lemma 5.3.5 we have (S) $n = n'$.

Given (F), (N), and (S), by Lemma 5.3.33 we have (T) $\omega = \omega'$.

Given (G), (O), (P), (Q), and (T), by Definition 5.3.4 we have (U) $\sigma_1 = \sigma_1'$.

Given (G), by Lemma 5.3.51 we have accessed (V) $(\text{p}, [(l, 0)])$. Given (O), by Lemma 5.3.51 we have accessed (W) $(\text{p}, [(l', 0)])$. Given (V), (W), and (P), we have (X) $(\text{p}, [(l, 0)]) = (\text{p}, [(l', 0)])$.

Given (A), (H), and (I) we have (Y) $(\text{p}, [fpd])$ = $(\text{p}, [fpd])$.

Given (R), (U), (X), and (Y), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x(\overline{p}))\ \|\ C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[df])} ((\text{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \text{acc},\ \text{skip})\ \|\ C)$

This case is similar to Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ty\ x(\overline{p})\{s\})\ \|\ C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[fpd])} ((\text{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \text{acc},\ \text{skip})\ \|\ C)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x(\overline{p})\{s\}) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[fd])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x(\overline{p})\{s\}) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[fd])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ by SMC^2 rule Pre-Declared Function Definition, we have $\mathrm{acc} = 0$, $x \in \gamma$, (B) $\gamma(x) = (l,\ \overline{ty} \rightarrow ty)$, (C) $\mathrm{CheckPublicEffects}(s,\ x,\ \gamma,\ \sigma) = n$, (D) $\sigma = \sigma_1[l \rightarrow (\mathrm{NULL},\ \overline{ty} \rightarrow ty,\ 1,\ \mathrm{PermL_Fun(public)})]$, (E) $\mathrm{EncodeFun}(s,\ n,\ \overline{p}) = \omega$, and (F) $\sigma_2 = \sigma_1[l \rightarrow (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \mathrm{PermL_Fun(public)})]$.

Given (G) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x(\overline{p})\{s\}) \parallel C) \Downarrow^{(\mathrm{p},[(l',0)])}_{(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_2,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ and (A), by Lemma 5.2.87 we have (H) $d = fd$.

Given (G) and (H), by SMC^2 rule Pre-Declared Function Definition, we have $\mathrm{acc} = 0$, $x \in \gamma$, (I) $\gamma(x) = (l',\ \overline{ty}' \rightarrow ty)$, (J) $\mathrm{CheckPublicEffects}(s,\ x,\ \gamma, \sigma) = n'$,
(K) $\sigma = \sigma'_1[l' \rightarrow (\mathrm{NULL},\ \overline{ty}' \rightarrow ty,\ 1,\ \mathrm{PermL_Fun(public)})]$, (L) $\mathrm{EncodeFun}(s,\ n',\ \overline{p}) = \omega'$, and (M) $\sigma'_2 = \sigma'_1[l' \rightarrow (\omega',\ \overline{ty}' \rightarrow ty,\ 1,\ \mathrm{PermL_Fun(public)})]$.

Given (B) and (I), by Definition 5.3.3 we have (N) $l = l'$ and (O) $\overline{ty} = \overline{ty}'$.

Given (C) and (J), by Lemma 5.3.5 we have (P) $n = n'$.

Given (D), (K), (N), and (O), by Definition 5.3.4 we have (Q) $\sigma_1 = \sigma'_1$.

Given (E), (L), and (P), by Lemma 5.3.33 we have (R) $\omega = \omega'$.

Given (F), (M), (N), (O), (Q), and (R), by Definition 5.3.4 we have (S) $\sigma_2 = \sigma'_2$.

Given (D) and (F), by Lemma 5.3.52 we have accessed (T) $(\mathrm{p}, [(l, 0)])$. Given (K) and (M), by Lemma 5.3.52 we have accessed (U) $(\mathrm{p}, [(l', 0)])$. Given (T), (U), and (N), we have (V) $(\mathrm{p}, [(l, 0)]) = (\mathrm{p}, [(l', 0)])$.

Given (A), (G), and (H) we have (W) $(\mathrm{p}, [fd])$ = $(\mathrm{p}, [fd])$.

Given (S), (V), and (W), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x(\overline{e})) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])::\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[fc1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x(\overline{e})) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])::\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[fc1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$ by SMC^2 rule Function Call Without Public Side Effects, we have (B) $\gamma(x) = (l,\ \overline{ty} \rightarrow ty)$, (C) $\sigma(l) = (\omega,\ \overline{ty} \rightarrow ty,\ 1,\ \mathrm{PermL_Fun(public)})$, (D) $\mathrm{DecodeFun}(\omega) = (s,\ n,\ \overline{p})$, (E) $\mathrm{GetFunParamAssign}(\overline{p}, \overline{e}) = s_1$, (F) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ s_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$, (G) $n = 0$, and (H) $((\mathrm{p}, \gamma_1,\ \sigma_1, \Delta_1,\ \mathrm{acc},\ s) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma_2, \sigma_2, \Delta_2,\ \mathrm{acc},\ \mathrm{skip})$-$\parallel C_2)$.

Given (I) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x(\overline{e})) \parallel C) \Downarrow^{(\mathrm{p},[(l',0)])::\mathcal{L}'_1::\mathcal{L}'_2}_{\mathcal{D}'_1::\mathcal{D}'_2::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_2,\ \Delta'_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_2)$ and (A), by Lemma 5.2.87 we have (J) $d = fc1$.

Given (I) and (J), by SMC^2 rule Function Call Without Public Side Effects, we have (K) $\gamma(x) = (l',\ \overline{ty}' \rightarrow ty')$, (L) $\sigma(l') = (\omega',\ \overline{ty}' \rightarrow ty',\ 1,\ \mathrm{PermL_Fun(public)})$, (M) $\mathrm{DecodeFun}(\omega') = (s',\ n',\ \overline{p}')$, (N) $\mathrm{GetFunParamAssign}(\overline{p}', \overline{e}) = s'_1$, (O) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ s'_1) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma'_1,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_1)$, (P) $n' = 0$, and (Q) $((\mathrm{p}, \gamma'_1,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ s) \parallel C'_1) \Downarrow^{\mathcal{L}'_2}_{\mathcal{D}'_2} ((\mathrm{p}, \gamma'_2,\ \sigma'_2,\ \Delta'_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_2)$.

Given (B) and (K), by Definition 5.3.3 we have (R) $l = l'$, (S) $\overline{ty} = \overline{ty}'$, and (T) $ty = ty'$.

Given (C), (L), and (R), by Definition 5.3.4 we have (U) $\omega = \omega'$.

Given (D), (M), and (U), by Lemma 5.3.28 we have (V) $s = s'$, (W) $n = n'$, and (X) $\overline{p} = \overline{p}'$.

Given (E), (N), and (X), by Lemma 5.3.4 we have (Y) $s_1 = s'_1$.

Given (F), (O), and (Y), by the inductive hypothesis we have (Z) $\gamma_1 = \gamma'_1$, (A1) $\sigma_1 = \sigma'_1$, (B1) $\Delta_1 = \Delta'_1$, (C1) $\mathcal{D}_1 = \mathcal{D}'_1$, (D1) $\mathcal{L}_1 = \mathcal{L}'_1$, and (E1) $C_1 = C'_1$.

Given (H), (Q), (Z), (A1), (B1), (E1), by the inductive hypothesis we have (F1) $\gamma_2 = \gamma'_2$, (G1) $\sigma_2 = \sigma'_2$, (H1) $\Delta_2 = \Delta'_2$, (I1) $\mathcal{D}_2 = \mathcal{D}'_2$, (J1) $\mathcal{L}_2 = \mathcal{L}'_2$, and (K1) $C_2 = C'_2$.

Given (C1), (I1), and $(\mathrm{p}, [fc1])$, by Lemma 5.3.38 we have (L1) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [fc1]) = \mathcal{D}'_1 :: \mathcal{D}'_2 :: (\mathrm{p}, [fc1])$.

Given (C) and (D), by Lemma 5.3.65 we have accessed (M1) $(\mathrm{p}, [(l, 0)])$. Given (L) and (M), by Lemma 5.3.65 we have accessed (N1) $(\mathrm{p}, [(l', 0)])$. Given (M1), (N1), and (R), we have (O1) $(\mathrm{p}, [(l, 0)]) = (\mathrm{p}, [(l', 0)])$. Given (D1), (J1), and (O1), by Lemma 5.3.47 we have (P1) $\mathcal{L}_1 :: \mathcal{L}_2 :: (\mathrm{p}, [(l, 0)]) = \mathcal{L}'_1 :: \mathcal{L}'_2 :: (\mathrm{p}, [(l', 0)])$.

Given (G1), (H1), (K1), (L1), and (P1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x(\overline{e})) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])::\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[fc])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x(\overline{e})) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])::\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[fc1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{sizeof}(ty)) \parallel C) \Downarrow^{\epsilon}_{(\mathrm{p},[ty])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{sizeof}(ty)) \parallel C) \Downarrow^{\epsilon}_{(\mathrm{p},[ty])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$ by SMC^2 rule Size of Type, we have (B) $n = \tau(ty)$ and $(ty) \nvdash \gamma$.

Given (C) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{sizeof}(ty)) \parallel C) \Downarrow^{\epsilon}_{(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n') \parallel C)$ and (A), by Lemma 5.2.87 we have (D) $d = ty$.

Given (C) and (D), by SMC^2 rule Size of Type, we have (E) $n' = \tau(ty)$ and $(ty) \nvdash \gamma$.

Given (B) and (E), by Lemma 5.3.6 we have (F) $n = n'$.

Given (A), (C), and (D), we have (G) $(\mathrm{p}, [ty]) = (\mathrm{p}, [ty])$.

Given (F) and (G), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \&x) \parallel C) \Downarrow^{\epsilon}_{(\mathrm{p},[loc])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l, 0)) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \&x) \parallel C) \Downarrow^{\epsilon}_{(\mathrm{p},[loc])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l, 0)) \parallel C)$ by SMC^2 rule Address Of, we have (B) $\gamma(x) = (l,\ ty)$.

Given (C) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \&x) \parallel C) \Downarrow^{\epsilon}_{(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l', 0)) \parallel C)$ and (A), by Lemma 5.2.87 we have (D) $d = loc$.

Given (C) and (D), by SMC^2 rule Address Of, we have (E) $\gamma(x) = (l',\ ty')$.

Given (B) and (E), by Definition 5.3.3 we have (F) $l = l'$ and $ty = ty'$.

Given (A), (C), and (D), we have (G) $(\mathrm{p}, [loc]) = (\mathrm{p}, [loc])$.

Given (F) and (G), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\mathrm{p},[cv])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\mathrm{p},[cv])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1)$ by SMC^2 rule Cast Public Value, we have $(e) \nvdash \gamma$, $(ty = \text{public } bty)$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$, and (C) $n_1 = \mathrm{Cast}(\text{public},\ ty,\ n)$.

Given (D) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ n'_1) \parallel C'_1)$ and (A), by Lemma 5.2.87 we have (E) $d = cv$.

Given (D) and (E), by SMC^2 rule Cast Public Value, we have $(e) \nvdash \gamma$, $(ty = \text{public } bty)$, (F) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ n') \parallel C'_1)$, and (G) $n'_1 = \mathrm{Cast}(\text{public},\ ty,\ n')$.

Given (B) and (F), by the inductive hypothesis we have (H) $\sigma_1 = \sigma'_1$, (I) $\Delta_1 = \Delta'_1$, (J) $n = n'$, (K) $\mathcal{D}_1 = \mathcal{D}'_1$, (L) $\mathcal{L}_1 = \mathcal{L}'_1$, and (M) $C_1 = C'_1$.

Given (C), (G), and (J), by Lemma 5.3.7 we have (N) $n_1 = n'_1$.

Given (K) and $(\mathrm{p}, [cv])$, by Lemma 5.3.38 we have (O) $\mathcal{D}_1 :: (\mathrm{p}, [cv]) = \mathcal{D}'_1 :: (\mathrm{p}, [cv])$.

Given (H), (I), (N), (M), (L), and (O), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\mathrm{p},[cv1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1::(\mathrm{p},[cv])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_1) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[cl1])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_1,\ \mathrm{acc},\ (l,0)) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[cl1])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_1,\ \mathrm{acc},\ (l,0)) \parallel C_1)$ by SMC^2 rule Cast Private Location, we have $(ty = \text{private } bty*)$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ (l,0)) \parallel C_1)$, (C) $\sigma_1 = \sigma_2\big[l \rightarrow \big(\omega,\ \text{void},\ n, \mathrm{PermL}(\text{Freeable},\ ty, \text{private}, n)\big)\big]$, and (D) $\sigma_3 = \sigma_2\big[l \rightarrow \big(\omega,\ ty,\ \frac{n}{\tau(ty)}, \mathrm{PermL}(\text{Freeable},\ ty, \text{private},\ \frac{n}{\tau(ty)})\big)\big]$.

Given (E) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}'_1::(\mathrm{p},[(l',0)])}_{\mathcal{D}'_1::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_3,\ \Delta'_1,\ \mathrm{acc},\ (l',0)) \parallel C'_1)$ and (A), by Lemma 5.2.87 we have (F) $d = cl1$.

Given (E) and (F), by SMC^2 rule Cast Private Location, we have $(ty = \text{private } bty*)$, (G) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ (l',0)) \parallel C'_1)$, (H) $\sigma'_1 = \sigma'_2\big[l' \rightarrow \big(\omega',\ \text{void},\ n', \mathrm{PermL}(\text{Freeable},\ \text{void}, \text{private}, n')\big)\big]$, and (I) $\sigma'_3 = \sigma'_2\big[l' \rightarrow \big(\omega',\ ty,\ \frac{n'}{\tau(ty)}, \mathrm{PermL}(\text{Freeable},\ ty, \text{private},\ \frac{n'}{\tau(ty)})\big)\big]$.

Given (B) and (G), by the inductive hypothesis we have (J) $\sigma_1 = \sigma'_1$, (K) $\Delta_1 = \Delta'_1$, (L) $l = l'$, (M) $\mathcal{D}_1 = \mathcal{D}'_1$, (N) $\mathcal{L}_1 = \mathcal{L}'_1$, and (O) $C_1 = C'_1$.

Given (C), (H), (J), and (L), by Definition 5.3.4 we have (P) $\sigma_2 = \sigma'_2$, (Q) $\omega = \omega'$, and (R) $n = n'$.

Given (D), (I), (P), (L), (Q), and (R), by Definition 5.3.4 we have (S) $\sigma_3 = \sigma'_3$.

Given (C) and (D), by Lemma 5.3.52 we have accessed (T) $(\mathrm{p}, [(l,0)])$. Given (H) and (I), by Lemma 5.3.52 we have accessed (U) $(\mathrm{p}, [(l',0)])$. Given (T), (U), and (L), we have (V) $(\mathrm{p}, [(l,0)]) = (\mathrm{p}, [(l',0)])$. Given (N) and (V), by Lemma 5.3.47 we have (W) $\mathcal{L}_1 :: (\mathrm{p}, [(l,0)]) = \mathcal{L}'_1 :: (\mathrm{p}, [(l',0)])$.

Given (A), (E), (F), and (M), we have (X) $\mathcal{D}_1 :: (\mathrm{p}, [cl1]) = \mathcal{D}'_1 :: (\mathrm{p}, [cl1])$.

Given (S), (K), (L), (O), (W), and (X), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[cl])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_1,\ \mathrm{acc},\ (l,0)) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (ty)\ e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[cl1])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_1,\ \mathrm{acc},\ (l,0)) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{free}(e)) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,0)])}_{(\mathrm{p},[fre])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{free}(e)) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,0)])}_{(\mathrm{p},[fre])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ by SMC^2 rule Single Location Free, we have $\mathrm{acc} = 0$, (B) $\gamma(x) = (l,\ \text{public}\ bty*)$, (C) $\sigma(l) = (\omega, \text{public}\ bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, \text{public}\ bty*, \text{public}, 1))$, (D) $\mathrm{DecodePtr}(\text{public}\ bty*,\ 1,\ \omega) = [1,\ [(l_1,0)],\ [1],\ 1]$, (E) $\mathrm{CheckFreeable}(\gamma, [(l_1,0)], [1], \sigma) = 1$, and (F) $\mathrm{Free}(\sigma,\ l_1) = (\sigma_1, (l_1, 0))$.

Given (G) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{free}(e)) \parallel C) \Downarrow^{(\mathrm{p},[(l',0),(l'_1,0)])}_{(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ and (A), by Lemma 5.2.87 we have (H) $d = fre$.

Given (G) and (H), by SMC^2 rule Single Location Free, we have $\mathrm{acc} = 0$, (I) $\gamma(x) = (l',\ \text{public}\ bty'*)$, (J) $\sigma(l') = (\omega', \text{public}\ bty'*, 1, \mathrm{PermL}(\mathrm{Freeable}, \text{public}\ bty'*, \text{public}, 1))$, (K) $\mathrm{DecodePtr}(\text{public}\ bty'*,\ 1,\ \omega') = [1,\ [(l'_1,0)],\ [1],\ 1]$, (L) $\mathrm{CheckFreeable}(\gamma, [(l'_1,0)], [1], \sigma) = 1$, and (M) $\mathrm{Free}(\sigma,\ l'_1) = (\sigma'_1, (l'_1, 0))$.

Given (B) and (I), by Definition 5.3.3 we have (N) $l = l'$ and (O) $bty = bty'$.

Given (C), (J), and (N), by Definition 5.3.4 we have (P) $\omega = \omega'$.

Given (D), (K), (O), (P), by Lemma 5.3.26 we have (Q) $l_1 = l'_1$.

Given (F), (M), and (Q), by Lemma 5.3.8 we have (R) $\sigma_1 = \sigma'_1$.

Given (C) and (D), by Lemma 5.3.62 we have accessed (S) $(\mathrm{p}, [(l,0)])$. Given (F), by Lemma 5.3.48 we have accessed location (T) $(\mathrm{p}, [(l_1,0)]$ Given (J) and (K), by Lemma 5.3.62 we have accessed (U) $(\mathrm{p}, [(l',0)])$. Given (M), by

Lemma 5.3.48 we have accessed location (V) $(\mathrm{p}, [(l_1', 0)]$

Given (S), (T), (U), and (V), by Lemmas 5.3.44 and 5.3.45 we have (W) $(\mathrm{p}, [(l, 0), (l_1, 0)])$ and (X) $(\mathrm{p}, [(l', 0), (l_1', 0)])$. Given (W), (X), (N), and (Q) by Definition 5.3.10 we have (Y) $(\mathrm{p}, [(l, 0), (l_1, 0)]) = (\mathrm{p}, [(l', 0), (l_1', 0)])$.

Given (A), (G), and (H), we have (Z) $(\mathrm{p}, [\mathit{fre}]) = (\mathrm{p}, [\mathit{fre}])$.

Given (R), (Y), and (Z), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{pfree}(x)) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,0)])}_{(\mathrm{p},[\mathit{pfre}])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{free}(e)) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,0)])]}_{\mathcal{D}_1::(\mathrm{p},[\mathit{fre}])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{pmalloc}(e,\ \mathit{ty})) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[\mathit{malp}])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ (l, 0)) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{pmalloc}(e,\ \mathit{ty})) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[\mathit{malp}])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ (l, 0)) \parallel C_1)$ by SMC^2 rule Private Malloc, we have $(e) \nvdash \gamma$, $\mathrm{acc} = 0$, $(\mathit{ty} = \text{private } \mathit{bty}*) \vee (\mathit{ty} = \text{private } \mathit{bty})$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$, (C) $l = \phi()$, and (D) $\sigma_2 = \sigma_1\big[l \to \big(\mathrm{NULL},\ \mathrm{void}*,\ n \cdot \tau(\mathit{ty}), \mathrm{PermL}(\mathrm{Freeable}, \mathrm{void}*, \mathrm{private}, n \cdot \tau(\mathit{ty}))\big)\big]$.

Given (E) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{pmalloc}(e,\ \mathit{ty})) \parallel C) \Downarrow^{\mathcal{L}_1'::(\mathrm{p},[(l',0)])}_{\mathcal{D}_1'::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma_2',\ \Delta_1',\ \mathrm{acc},\ (l', 0)) \parallel C_1')$ by Lemma 5.2.87 we have (F) $d = \mathit{malp}$.

Given (E) and (F), by SMC^2 rule Private Malloc, we have $(e) \nvdash \gamma$, $\mathrm{acc} = 0$, $(\mathit{ty} = \text{private } \mathit{bty}*) \vee (\mathit{ty} = \text{private } \mathit{bty})$, (G) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1'}_{\mathcal{D}_1'} ((\mathrm{p}, \gamma,\ \sigma_1',\ \Delta_1,\ \mathrm{acc},\ n') \parallel C_1')$, (H) $l' = \phi()$, and (I) $\sigma_2' = \sigma_1'\big[l' \to \big(\mathrm{NULL},\ \mathrm{void}*,\ n' \cdot \tau(\mathit{ty}), \mathrm{PermL}(\mathrm{Freeable}, \mathrm{void}*, \mathrm{private}, n' \cdot \tau(\mathit{ty}))\big)\big]$.

Given (B) and (G), by the inductive hypothesis we have (J) $\sigma_1 = \sigma_1'$, (K) $\Delta_1 = \Delta_1'$, (L) $n = n'$, (M) $\mathcal{D}_1 = \mathcal{D}_1'$, (N) $\mathcal{L}_1 = \mathcal{L}_1'$, and (O) $C_1 = C_1'$.

Given (C) and (H), by Axiom 5.3.4 we have (P) $l = l'$.

Given (D), (I), (J), (P), and (L), by Definition 5.3.4 we have (Q) $\sigma_2 = \sigma_2'$.

Given (D), by Lemma 5.3.51 we have accessed location (R) $(\mathrm{p}, [(l, 0)])$. Given (I), by Lemma 5.3.51 we have accessed location (S) $(\mathrm{p}, [(l', 0)])$. Given (N), (P), (R), and (S), by Lemma 5.3.47 we have (T) $\mathcal{L}_1 :: (\mathrm{p}, [(l, 0)]) = \mathcal{L}_1' :: (\mathrm{p}, [(l', 0)])$.

Given (O) and $(\mathrm{p}, [malp])$, by Lemma 5.3.38 we have (U) $\mathcal{D}_1 :: (\mathrm{p}, [malp]) = \mathcal{D}_1' :: (\mathrm{p}, [malp])$.

Given (Q), (K), (P), (T), (U) and (O), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{malloc}(e)) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [mal])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ (l, 0)) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{pmalloc}(e,\ ty)) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [malp])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ (l, 0)) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, ++\ x) \parallel C) \Downarrow^{(\mathrm{p}, [(l,0)])}_{(\mathrm{p}, [pin3])} ((\mathrm{p}, \gamma, \sigma_1, \Delta, \mathrm{acc}, n_2) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, ++\ x) \parallel C) \Downarrow^{(\mathrm{p}, [(l,0)])}_{(\mathrm{p}, [pin3])} ((\mathrm{p}, \gamma, \sigma_1, \Delta, \mathrm{acc}, n_2) \parallel C)$, by SMC^2 rule Pre-Increment Private Int Variable, we have (B) $\gamma(x) = (l, \mathrm{private\ int})$, (C) $\sigma(l) = (\omega, \mathrm{private\ int}, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private\ int}, \mathrm{private}, 1))$, (D) $\mathrm{DecodeVal}(\mathrm{private\ int}, \omega) = n_1$, (E) $n_2 = n_1 + \mathrm{encrypt}(1)$, and (F) $\mathrm{UpdateVal}(\sigma, l, n_2, \mathrm{private\ int}) = \sigma_1$.

Given (G) $\Sigma \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, ++\ x) \parallel C) \Downarrow^{(\mathrm{p}, [(l',0)])}_{(\mathrm{p}, [d])} ((\mathrm{p}, \gamma, \sigma_1', \Delta, \mathrm{acc}, v_2') \parallel C)$ and (A), by Lemma 5.2.87 we have (H) $d = pin3$.

Given (G) and (H), by SMC^2 rule Pre-Increment Private Int Variable, we have (I) $\gamma(x) = (l', \mathrm{private\ int})$, (J) $\sigma(l') = (\omega', \mathrm{private\ int}, 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private\ int}, \mathrm{private}, 1))$, (K) $\mathrm{DecodeVal}(\mathrm{private\ int}, \omega') = n_1'$, (L) $n_2' = n_1' + \mathrm{encrypt}(1)$, and (M) $\mathrm{UpdateVal}(\sigma, l', n_2', \mathrm{private\ int}) = \sigma_1'$.

Given (B) and (I), by Definition 5.3.3 we have (N) $l = l'$.

Given (C), (J), and (N), by Definition 5.3.4 we have (O) $\omega = \omega'$.

Given (D), (K), and (O), by Lemma 5.3.29 we have (P) $n_1 = n_1'$.

By Axiom 5.3.1, we have (Q) $\text{encrypt}(1) = \text{encrypt}(1)$. Given (E), (L), (P), and (Q), we have (R) $n_2 = n_2'$.

Given (F), (M), (N), and (R), by Lemma 5.3.34 we have (S) $\sigma_1 = \sigma_1'$.

Given (A), (G), and (H), we have (T) $(\text{p}, [pin3]) = (\text{p}, [pin3])$.

Given (C) and (D), by Lemma 5.3.64 and Lemma 5.3.66 we have accessed location (U) $(\text{p}, [(l, 0)])$. Given (J) and (K), by Lemma 5.3.64 and Lemma 5.3.66 we have accessed location (V) $(\text{p}, [(l', 0)])$. Given (U), (V), and (N), we have (W) $(\text{p}, [(l, 0)]) = (\text{p}, [(l', 0)])$.

Given (S), (R), (T), and (W) by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ n_1) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\text{p}, \gamma, \sigma, \Delta, \text{acc}, ++\ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin3])} ((\text{p}, \gamma, \sigma_1, \Delta, \text{acc}, n_2) \parallel C)$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin5])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ [\alpha,\ \bar{l}_1,\ \bar{j},\ i]) \parallel C)$

Given (A) $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin5])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ [\alpha,\ \bar{l}_1,\ \bar{j},\ i]) \parallel C)$ by SMC^2 rule Pre-Increment Private Pointer Multiple Locations, we have (B) $\gamma(x) = (l,\ \text{private}\ bty*)$, (C) $\sigma(l) = (\omega,\ \text{private}\ bty*,\ \alpha,\ \text{PermL}(\text{Freeable}, \text{private}\ bty*, \text{private}, \alpha))$, (D) $\text{DecodePtr}(\text{private}\ bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l}, \bar{j},\ i]$, (E) $\text{IncrementList}(\bar{l},\ \tau(\text{private}\ bty*), \sigma) = (\bar{l}_1, 1)$, and (F) $\text{UpdatePtr}(\sigma,\ (l, 0),\ [\alpha,\ \bar{l}_1,\ \bar{j},\ i],\ \text{private}\ bty*) = (\sigma_1,\ 1)$.

Given (G) $\Sigma \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C) \Downarrow^{(\text{p},[(l',0)])}_{(\text{p},[d])} ((\text{p}, \gamma,\ \sigma_1',\ \Delta,\ \text{acc},\ [\alpha',\ \bar{l}_1',\ \bar{j}',\ i']) \parallel C)$ and (A), by Lemma 5.2.87 we have (H) $d = pin5$.

Given (G) and (H), by SMC^2 rule Pre-Increment Private Pointer Multiple Locations, we have (I) $\gamma(x) = (l',\ \text{private}\ bty'*)$, (J) $\sigma(l') = (\omega',\ \text{private}\ bty'*,\ \alpha',\ \text{PermL}(\text{Freeable}, \text{private}\ bty'*, \text{private}, \alpha'))$,
(K) $\text{DecodePtr}(\text{private}\ bty'*,\ \alpha',\ \omega') = [\alpha',\ \bar{l}',\ \bar{j}',\ i']$, (L) $\text{IncrementList}(\bar{l}',\ \tau(\text{private}\ bty'*), \sigma) = (\bar{l}'_1, 1)$, and (M) $\text{UpdatePtr}(\sigma,\ (l', 0),\ [\alpha',\ \bar{l}'_1,\ \bar{j}',\ i'],\ \text{private}\ bty'*) = (\sigma'_1,\ 1)$.

Given (B) and (I), by Definition 5.3.3 we have (N) $l = l'$ and (O) $bty = bty'$.

Given (C), (J), and (N), by Definition 5.3.4 we have (P) $\omega = \omega'$ and (Q) $\alpha = \alpha'$.

Given (D), (K), (O), (P), and (Q), by Lemma 5.3.26 we have (R) $\bar{l} = \bar{l}'$, (S) $\bar{j} = \bar{j}'$, and (T) $i = i'$.

Given (E), (L), (R), and (O), by Lemma 5.3.9 we have (U) $\bar{l}_1 = \bar{l}'_1$.

Given (F), (M), (N), (O), (Q), (S), (T), and (U), by Lemma 5.3.36 we have (V) $\sigma_1 = \sigma'_1$.

Given (A), (G), and (H), we have (W) $(\text{p}, [pin5]) = (\text{p}, [pin5])$.

Given (C) and (D), by Lemma 5.3.62 we have accessed location (X) $(\text{p}, [(l, 0)])$. Given (J) and (K), by Lemma 5.3.62 we have accessed location (Y) $(\text{p}, [(l', 0)])$. Given (X), (Y), and (N), we have (Z) $(\text{p}, [(l, 0)]) = (\text{p}, [(l', 0)])$.

Given (V), (Q), (U), (S), (T), (W) and (Z) by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin4])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ [n,\ \bar{l}_1,\ \bar{j},\ 1]) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin5])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ [\alpha,\ \bar{l}_1,\ \bar{j},\ i]) \parallel C)$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin2])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ (l_2, \mu_2)) \parallel C)$

Given (A) $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin2])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ (l_2, \mu_2)) \parallel C)$ by SMC^2 rule Pre-Increment Public Pointer Higher Level Indirection Single Location, we have $i > 1$, (B) $\gamma(x) = (l,\ \text{public}\ bty*)$,

(C) $\sigma(l) = (\omega,\ \text{public } bty*,\ 1,\ \text{PermL}(\text{Freeable}, \text{public } bty*, \text{public}, 1))$, (D) $\text{DecodePtr}(\text{public } bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i]$, (E) $((l_2, \mu_2), 1) = \text{GetLocation}((l_1, \mu_1), \tau(\text{public } bty*), \sigma)$, and (F) $\text{UpdatePtr}(\sigma,\ (l, 0),\ [1,\ [(l_2, \mu_2)],\ [1],\ i],\ \text{public } bty) = (\sigma_1,\ 1)$.

Given (G) $\Sigma \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C) \Downarrow^{(\text{p},[(l',0)])}_{(\text{p},[d])} ((\text{p}, \gamma,\ \sigma'_1,\ \Delta,\ \text{acc},\ (l'_2, \mu'_2)) \parallel C)$ and (A), by Lemma 5.2.87 we have (H) $d = pin2$.

Given (G) and (H), by SMC^2 rule Pre-Increment Public Pointer Higher Level Indirection Single Location, we have $i' > 1$, (I) $\gamma(x) = (l',\ \text{public } bty'*)$, (J) $\sigma(l') = (\omega',\ \text{public } bty'*,\ 1,\ \text{PermL}(\text{Freeable}, \text{public } bty'*, \text{public}, 1))$, (K) $\text{DecodePtr}(\text{public } bty'*,\ 1,\ \omega') = [1,\ [(l'_1, \mu'_1)],\ [1],\ i']$, (L) $((l'_2, \mu'_2), 1) = \text{GetLocation}((l'_1, \mu'_1), \tau(\text{public } bty'*), \sigma)$, and (M) $\text{UpdatePtr}(\sigma,\ (l', 0),\ [1, [(l'_2, \mu'_2)],\ [1],\ i'],\ \text{public } bty') = (\sigma'_1,\ 1)$.

Given (B) and (I), by Definition 5.3.3 we have (N) $l = l'$ and (O) $bty = bty'$.

Given (C), (J), and (N), by Definition 5.3.4 we have (P) $\omega = \omega'$.

Given (D), (K), (O), and (P), by Lemma 5.3.26 we have (Q) $l_1 = l'_1$, (R) $\mu_1 = \mu'_1$, and (S) $i = i'$.

Given (E), (L), (O), (Q), and (R), by Lemma 5.3.10 we have (T) $l_2 = l'_2$ and (U) $\mu_2 = \mu'_2$.

Given (F), (M), (N), (O), (S), (T), and (U), by Lemma 5.3.36 we have (V) $\sigma_1 = \sigma'_1$.

Given (A), (G), and (H), we have (W) $(\text{p}, [pin2]) = (\text{p}, [pin2])$.

Given (C) and (D), by Lemma 5.3.62 we have accessed location (X) $(\text{p}, [(l, 0)])$. Given (J) and (K), by Lemma 5.3.62 we have accessed location (Y) $(\text{p}, [(l', 0)])$. Given (X), (Y), and (N), we have (Z) $(\text{p}, [(l, 0)]) = (\text{p}, [(l', 0)])$.

Given (V), (T), (U), (W) and (Z) by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ ++\ x) \parallel C) \Downarrow^{(\text{p},[(l,0)])}_{(\text{p},[pin1])} ((\text{p}, \gamma,\ \sigma_1,\ \Delta,\ \text{acc},\ (l_2, \mu_2)) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin2])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin6])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin2])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin7])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ++\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[pin2])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[iet])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[iet])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$ by SMC^2 rule Public If Else True, we have $(e) \nvdash \gamma, n \neq 0$, (B)$((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$, and (C) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ s_1) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma_1,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Given (D) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \parallel C) \Downarrow^{\mathcal{L}'_1::\mathcal{L}'_2}_{\mathcal{D}'_1::\mathcal{D}'_2::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_2,\ \Delta'_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_2)$ and (A), by Lemma 5.2.87 we have (E) $d = iet$.

Given (D) and (E), by SMC^2 rule Public If Else True, we have $(e) \nvdash \gamma, n' \neq 0$ (F)$((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1}$ $((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ n') \parallel C'_1)$, and (G) $((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ s'_1) \parallel C'_1) \Downarrow^{\mathcal{L}'_2}_{\mathcal{D}'_2} ((\mathrm{p}, \gamma'_1,\ \sigma'_2,\ \Delta'_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_2)$.

Given (B) and (F), by the inductive hypothesis we have (H) $\sigma_1 = \sigma'_1$, (I) $\Delta_1 = \Delta'_1$, (J) $n = n'$, (K) $\mathcal{D}_1 = \mathcal{D}'_1$, (L) $\mathcal{L}_1 = \mathcal{L}'_1$, and (M) $C_1 = C'_1$.

Given (C), (G), (H), (I), and (M), by the inductive hypothesis we have (N) $\gamma_2 = \gamma'_2$, (O) $\sigma_2 = \sigma'_2$, (P) $\Delta_2 = \Delta'_2$, (Q) $\mathcal{D}_2 = \mathcal{D}'_2$, (R) $\mathcal{L}_2 = \mathcal{L}'_2$, and (S) $C_2 = C'_2$.

Given (K), (Q), and $(\mathrm{p}, [iet])$, by Lemma 5.3.38 we have (T) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [iet]) = \mathcal{D}'_1 :: \mathcal{D}'_2 :: (\mathrm{p}, [iet])$.

Given (L) and (R), by Lemma 5.3.47 we have (U) $\mathcal{L}_1 :: \mathcal{L}_2 = \mathcal{L}'_1 :: \mathcal{L}'_2$.

Given (O), (P), (S), (T), and (U), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p},\ \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[ief])} ((\mathrm{p},\ \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[iet])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{while}\ (e)\ s) \parallel C) \Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wle])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{while}\ (e)\ s) \parallel C) \Downarrow^{\mathcal{L}}_{\mathcal{D}::(\mathrm{p},[wle])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule While End, we have $(e) \nvdash \gamma, n\ =\ 0$, and (B) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \Downarrow^{\mathcal{L}}_{\mathcal{D}} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1)$.

Given (C) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{while}\ (e)\ s) \parallel C) \Downarrow^{\mathcal{L}'}_{\mathcal{D}'::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_1)$ and (A), by Lemma 5.2.87 we have (D) $d = wle$.

Given (C) and (D), by SMC^2 rule While End, we have $(e) \nvdash \gamma, n'\ =\ 0$, and (E) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \Downarrow^{\mathcal{L}'}_{\mathcal{D}'}$ $((\mathrm{p}, \gamma, \sigma'_1, \Delta'_1, \mathrm{acc}, n') \parallel C'_1)$.

Given (B) and (E), by the inductive hypothesis we have (F) $\sigma_1 = \sigma'_1$, (G) $\Delta_1 = \Delta'_1$, (H) $n = n'$, (I) $\mathcal{D} = \mathcal{D}'$, (J) $\mathcal{L} = \mathcal{L}'$, and (K) $C_1 = C'_1$.

Given (I) and $(\mathrm{p}, [wle])$, by Lemma 5.3.38 we have (L) $\mathcal{D} :: (\mathrm{p}, [wle]) = \mathcal{D}' :: (\mathrm{p}, [wle])$.

Given (F), (G), (J), (L), and (K), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{while}\ (e)\ s) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wlc])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{while}\ (e)\ s) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ \mathrm{while}\ (e)\ s) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wlc])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{while}\ (e)\ s) \parallel C_2)$ by SMC^2

rule While Continue, we have $(e) \nvdash \gamma, n \neq 0$, (B) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, n) \parallel C_1)$, and (C) $((\mathrm{p}, \gamma, \sigma_1, \Delta_1, \mathrm{acc}, s) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma_1, \sigma_2, \Delta_2, \mathrm{acc}, \mathrm{skip}) \parallel C_2)$.

Given (D) $\Sigma \triangleright ((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, \mathrm{while}\ (e)\ s) \parallel C) \Downarrow^{\mathcal{L}'_1::\mathcal{L}'_2}_{\mathcal{D}'_1::\mathcal{D}'_2::(\mathrm{p},[d])} ((\mathrm{p}, \gamma, \sigma'_2, \Delta'_2, \mathrm{acc}, \mathrm{while}\ (e)\ s) \parallel C'_2)$ and (A), by Lemma 5.2.87 we have (E) $d = wlc$.

Given (D) and (E), by SMC^2 rule While Continue, we have $(e) \nvdash \gamma, n \neq 0$, (F) $((\mathrm{p}, \gamma, \sigma, \Delta, \mathrm{acc}, e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma, \sigma'_1, \Delta'_1, \mathrm{acc}, n') \parallel C'_1)$, and (G) $((\mathrm{p}, \gamma, \sigma'_1, \Delta'_1, \mathrm{acc}, s) \parallel C'_1) \Downarrow^{\mathcal{L}'_2}_{\mathcal{D}'_2} ((\mathrm{p}, \gamma'_1, \sigma'_2, \Delta'_2, \mathrm{acc}, \mathrm{skip}) \parallel C'_2)$.

Given (B) and (F), by the inductive hypothesis we have (H) $\sigma_1 = \sigma'_1$, (I) $\Delta_1 = \Delta'_1$, (J) $n = n'$, (K) $\mathcal{D}_1 = \mathcal{D}'_1$, (L) $\mathcal{L}_1 = \mathcal{L}'_1$, and (M) $C_1 = C'_1$.

Given (C), (G), (H), (I), and (M), by the inductive hypothesis we have (N) $\gamma_1 = \gamma'_1$, (O) $\sigma_2 = \sigma'_2$, (P) $\Delta_2 = \Delta'_2$, (Q) $\mathcal{D}_2 = \mathcal{D}'_2$, (R) $\mathcal{L}_2 = \mathcal{L}'_2$, and (S) $C_2 = C'_2$.

Given (K), (Q), and $(\mathrm{p}, [wlc])$, by Lemma 5.3.38 we have (T) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wlc]) = \mathcal{D}'_1 :: \mathcal{D}'_2 :: (\mathrm{p}, [wlc])$.

Given (L) and (R), by Lemma 5.3.47 we have (U) $\mathcal{L}_1 :: \mathcal{L}_2 = \mathcal{L}'_1 :: \mathcal{L}'_2$.

Given (O), (P), (S), (T), and (U), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3::\mathcal{L}_4::\mathcal{L}_5::\mathcal{L}_6::\mathcal{L}_7}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[iep])}$ $((1, \gamma^1, \sigma^1_6, \Delta^1_3, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_6, \Delta^{\mathrm{q}}_3, \mathrm{acc}, \mathrm{skip}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2))$ $\Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3::\mathcal{L}_4::\mathcal{L}_5::\mathcal{L}_6::\mathcal{L}_7}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[iep])} ((1, \gamma^1, \sigma^1_6, \Delta^1_3, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_6, \Delta^{\mathrm{q}}_3, \mathrm{acc}, \mathrm{skip}))$ by SMC^2 rule Private If Else (Variable Tracking), we have $\{(e) \vdash \gamma^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$, (B) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((1, \gamma^1, \sigma^1_1, \Delta^1_1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}_1, \Delta^{\mathrm{q}}_1, \mathrm{acc}, n^{\mathrm{q}}))$, (C) $\{\mathrm{Extract}(s_1, s_2, \gamma^{\mathrm{p}}) = (x_{list}, 0)\}^{\mathrm{q}}_{\mathrm{p}=1}$,
(D) $\{\mathrm{InitializeVariables}(x_{list}, \gamma^{\mathrm{p}}, \sigma^{\mathrm{p}}_1, n^{\mathrm{p}}, \mathrm{acc}+1) = (\gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_2, \bar{l}^{\mathrm{p}}_2)\}^{\mathrm{q}}_{\mathrm{p}=1}$, (E) $((1, \gamma^1_1, \sigma^1_2, \Delta^1_1, \mathrm{acc}+1, s_1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}_1, \sigma^{\mathrm{q}}_2, \Delta^{\mathrm{q}}_1, \mathrm{acc}+1, s_1)) \Downarrow^{\mathcal{L}_3}_{\mathcal{D}_2} ((1, \gamma^1_2, \sigma^1_3, \Delta^1_2, \mathrm{acc}+1, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}_2, \sigma^{\mathrm{q}}_3, \Delta^{\mathrm{q}}_2, \mathrm{acc}+1, \mathrm{skip}))$,
(F) $\{\mathrm{RestoreVariables}(x_{list}, \gamma^{\mathrm{p}}_1, \sigma^{\mathrm{p}}_3, \mathrm{acc}+1) = (\sigma^{\mathrm{p}}_4, \bar{l}^{\mathrm{p}}_4)\}^{\mathrm{q}}_{\mathrm{p}=1}$, (G) $((1, \gamma^1_1, \sigma^1_4, \Delta^1_2, \mathrm{acc}+1, s_2) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}_1, \sigma^{\mathrm{q}}_4, \Delta^{\mathrm{q}}_2, \mathrm{acc}+1, s_2)) \Downarrow^{\mathcal{L}_5}_{\mathcal{D}_3} ((1, \gamma^1_3, \sigma^1_5, \Delta^1_3, \mathrm{acc}+1, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}_3, \sigma^{\mathrm{q}}_5, \Delta^{\mathrm{q}}_3, \mathrm{acc}+1, \mathrm{skip}))$,

(H) $\{\text{ResolveVariables_Retrieve}(x_{list},\ \text{acc}+1, \gamma_1^{\text{p}}, \sigma_5^{\text{p}}) = ([(v_{t1}^{\text{p}}, v_{e1}^{\text{p}}), ..., (v_{tm}^{\text{p}}, v_{em}^{\text{p}})],\ n_1^{\text{p}}, \bar{l}_6^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (I) $\text{MPC}_{resolve}([n_1^1, ..., n_1^{\text{q}}],\ [[(v_{t1}^1, v_{e1}^1), ..., (v_{tm}^1, v_{em}^1)],\ ...,\ [(v_{t1}^{\text{q}}, v_{e1}^{\text{q}}), ..., (v_{tm}^{\text{q}}, v_{em}^{\text{q}})]]) = [[v_1^1,\ ...,\ v_m^1],\ ...,\ [v_1^{\text{q}},\ ...,\ v_m^{\text{q}}]]$ (J) $\{\text{ResolveVariables_Store}(x_{list},\ \text{acc}+1, \gamma_1^{\text{p}}, \sigma_5^{\text{p}},\ [v_1^{\text{p}}, ..., v_m^{\text{p}}]) = (\sigma_6^{\text{p}}, \bar{l}_7^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (K) $\mathcal{L}_2 = (1, \bar{l}_2^1) \parallel ... \parallel (\text{q}, \bar{l}_2^{\text{q}})$, (L) $\mathcal{L}_4 = (1, \bar{l}_4^1) \parallel ... \parallel (\text{q}, \bar{l}_4^{\text{q}})$, (M) $\mathcal{L}_6 = (1, \bar{l}_6^1) \parallel ... \parallel (\text{q}, \bar{l}_6^{\text{q}})$, and (N) $\mathcal{L}_7 = (1, \bar{l}_7^1) \parallel ... \parallel (\text{q}, \bar{l}_7^{\text{q}})$.

Given (O) $\Sigma \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2)) \Downarrow_{\mathcal{D}_1'::\mathcal{D}_2'::\mathcal{D}_3'::(\text{p},[d])}^{\mathcal{L}_1'::\mathcal{L}_2'::\mathcal{L}_3'::\mathcal{L}_4'::\mathcal{L}_5'::\mathcal{L}_6'::\mathcal{L}_7'} ((1, \gamma^1, \sigma_6'^1, \Delta_3'^1, \text{acc}, \text{skip}) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma_6'^{\text{q}}, \Delta_3'^{\text{q}}, \text{acc}, \text{skip}))$ and (A), by Lemma 5.2.87 we have (P) $d = iep$.

Given (O) and (P), by SMC^2 rule Private If Else (Variable Tracking), we have $\{(e) \vdash \gamma^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (Q) $((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, e) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, e)) \Downarrow_{\mathcal{D}_1'}^{\mathcal{L}_1'} ((1, \gamma^1, \sigma_1'^1, \Delta_1'^1, \text{acc}, n'^1) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma_1'^{\text{q}}, \Delta_1'^{\text{q}}, \text{acc}, n'^{\text{q}}))$, (R) $\{\text{Extract}(s_1, s_2, \gamma^{\text{p}}) = (x_{list}', 0)\}_{\text{p}=1}^{\text{q}}$, (S) $\{\text{InitializeVariables}(x_{list}',\ \gamma^{\text{p}}, \sigma_1'^{\text{p}}, n'^{\text{p}},\ \text{acc}+1) = (\gamma_1'^{\text{p}}, \sigma_2'^{\text{p}}, \bar{l}_2'^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (T) $((1, \gamma_1'^1, \sigma_2'^1, \Delta_1'^1, \text{acc}+1, s_1) \parallel ... \parallel (\text{q}, \gamma_1'^{\text{q}}, \sigma_2'^{\text{q}}, \Delta_1'^{\text{q}}, \text{acc}+1, s_1)) \Downarrow_{\mathcal{D}_2'}^{\mathcal{L}_3'} ((1, \gamma_2'^1, \sigma_3'^1, \Delta_2'^1, \text{acc}+1, \text{skip}) \parallel ... \parallel (\text{q}, \gamma_2'^{\text{q}}, \sigma_3'^{\text{q}}, \Delta_2'^{\text{q}}, \text{acc}+1, \text{skip}))$, (U) $\{\text{RestoreVariables}(x_{list}',\ \gamma_1'^{\text{p}},\ \sigma_3'^{\text{p}},\ \text{acc}+1) = (\sigma_4'^{\text{p}}, \bar{l}_4'^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (V) $((1, \gamma_1'^1, \sigma_4'^1, \Delta_2'^1, \text{acc}+1, s_2) \parallel ... \parallel (\text{q}, \gamma_1'^{\text{q}}, \sigma_4'^{\text{q}}, \Delta_2'^{\text{q}}, \text{acc}+1, s_2)) \Downarrow_{\mathcal{D}_3'}^{\mathcal{L}_5'} ((1, \gamma_3'^1, \sigma_5'^1, \Delta_3'^1, \text{acc}+1, \text{skip}) \parallel ... \parallel (\text{q}, \gamma_3'^{\text{q}}, \sigma_5'^{\text{q}}, \Delta_3'^{\text{q}}, \text{acc}+1, \text{skip}))$, (W) $\{\text{ResolveVariables_Retrieve}(x_{list}',\ \text{acc}+1, \gamma_1'^{\text{p}}, \sigma_5'^{\text{p}}) = ([(v_{t1}'^{\text{p}}, v_{e1}'^{\text{p}}), ..., (v_{tm}'^{\text{p}}, v_{em}'^{\text{p}})],\ n_1'^{\text{p}}, \bar{l}_6'^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (X) $\text{MPC}_{resolve}([n_1'^1, ..., n_1'^{\text{q}}],\ [[(v_{t1}'^1, v_{e1}'^1), ..., (v_{tm}'^1, v_{em}'^1)],\ ...,\ [(v_{t1}'^{\text{q}}, v_{e1}'^{\text{q}}), ..., (v_{tm}'^{\text{q}},\ v_{em}'^{\text{q}})]]) = [[v_1'^1, ..., v_m'^1], ..., [v_1'^{\text{q}}, ..., v_m'^{\text{q}}]]$ (Y) $\{\text{ResolveVariables_Store}(x_{list}',\ \text{acc}+1, \gamma_1'^{\text{p}}, \sigma_5'^{\text{p}}, [v_1'^{\text{p}}, ..., v_m'^{\text{p}}]) = (\sigma_6'^{\text{p}}, \bar{l}_7'^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (Z) $\mathcal{L}_2' = (1, \bar{l}_2'^1) \parallel ... \parallel (\text{q}, \bar{l}_2'^{\text{q}})$, (A1) $\mathcal{L}_4' = (1, \bar{l}_4'^1) \parallel ... \parallel (\text{q}, \bar{l}_4'^{\text{q}})$, (B1) $\mathcal{L}_6' = (1, \bar{l}_6'^1) \parallel ... \parallel (\text{q}, \bar{l}_6'^{\text{q}})$, and (C1) $\mathcal{L}_7' = (1, \bar{l}_7'^1) \parallel ... \parallel (\text{q}, \bar{l}_7'^{\text{q}})$.

Given (B) and (Q), by the inductive hypothesis we have (D1) $\{\sigma_1^{\text{p}} = \sigma_1'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (E1) $\{\Delta_1^{\text{p}} = \Delta_1'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (F1) $\{n^{\text{p}} = n'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (G1) $\mathcal{D}_1 = \mathcal{D}_1'$, and (H1) $\mathcal{L}_1 = \mathcal{L}_1'$.

Given (C) and (R), by Lemma 5.3.16 we have (I1) $x_{list} = x_{list}'$.

Given (D), (S), (D1), and (F1), by Lemma 5.3.17 and we have (J1) $\{\gamma_1^{\text{p}} = \gamma_1'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (K1) $\{\sigma_2^{\text{p}} = \sigma_2'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, and (L1) $\{\bar{l}_2^{\text{p}} = \bar{l}_2'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$.

Given (L1), (K), and (Z), by Lemma 5.3.53 and Definition 5.3.10 we have (M1) $\mathcal{L}_2 = \mathcal{L}_2'$.

Given (E), (T), (J1), (K1), and (E1), by the inductive hypothesis we have (N1) $\{\gamma_2^{\text{p}} = \gamma_2'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (O1) $\{\sigma_3^{\text{p}} = \sigma_3'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (P1) $\{\Delta_2^{\text{p}} = \Delta_2'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (Q1) $\mathcal{D}_2 = \mathcal{D}_2'$, and (R1) $\mathcal{L}_3 = \mathcal{L}_3'$.

Given (F), (U), (I1), (J1), and (O1), by Lemma 5.3.18 we have (S1) $\{\sigma_4^{\mathrm{p}} = \sigma_4'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (T1) $\{\bar{l}_4^{\mathrm{p}} = \bar{l}_4'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (T1), (L), and (A1), by Lemma 5.3.54 and Definition 5.3.10 we have (U1) $\mathcal{L}_4 = \mathcal{L}_4'$.

Given (G), (V), (J1), (S1), and (P1), by the inductive hypothesis we have (V1) $\{\gamma_3^{\mathrm{p}} = \gamma_3'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (W1) $\{\sigma_5^{\mathrm{p}} = \sigma_5'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (X1) $\{\Delta_3^{\mathrm{p}} = \Delta_3'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (Y1) $\mathcal{D}_3 = \mathcal{D}_3'$, and (Z1) $\mathcal{L}_5 = \mathcal{L}_5'$.

Given (H), (W), (J1), (W1), (F1), and (I1), by Lemma 5.3.19 we have (A2) $\{[(v_{t1}^{\mathrm{p}}, v_{e1}^{\mathrm{p}}), ..., (v_{tm}^{\mathrm{p}}, v_{em}^{\mathrm{p}})] = [(v_{t1}'^{\mathrm{p}}, v_{e1}'^{\mathrm{p}}), ..., (v_{tm}'^{\mathrm{p}}, v_{em}'^{\mathrm{p}})]\}_{\mathrm{p}=1}^{\mathrm{q}}$, (B2) $\{n_1^{\mathrm{p}} = n_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (C2) $\{\bar{l}_6^{\mathrm{p}} = \bar{l}_6'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (M), (B1), and (C2), by Lemma 5.3.55 and Definition 5.3.10 we have (D2) $\mathcal{L}_6 = \mathcal{L}_6'$.

Given (I), (X), (B2), and (A2), by Axiom 5.3.10 we have $[[v_1^1, ..., v_m^1], ..., [v_1^{\mathrm{q}}, ..., v_m^{\mathrm{q}}]] = [[v_1'^1, ..., v_m'^1], ..., [v_1'^{\mathrm{q}}, ..., v_m'^{\mathrm{q}}]]$ and therefore (E2) $\{[v_1^{\mathrm{p}}, ..., v_m^{\mathrm{p}}] = [v_1'^{\mathrm{p}}, ..., v_m'^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (J), (Y), (I1), (J1), (W1), and (E2), by Lemma 5.3.20 we have (F2) $\{\sigma_6^{\mathrm{p}} = \sigma_6'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (G2) $\{\bar{l}_7^{\mathrm{p}} = \bar{l}_7'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (N), (C1), and (G2), by Lemma 5.3.56 and Definition 5.3.10 we have (H2) $\mathcal{L}_7 = \mathcal{L}_7'$.

Given (G1), (Q1), (Y1), and (P), by Lemma 5.3.38 we have (I2) $\mathcal{D}_1 :: \mathcal{D}_2 :: \mathcal{D}_3 :: (\mathrm{p}, [iep]) = \mathcal{D}_1 :: \mathcal{D}_2' :: \mathcal{D}_3'(\mathrm{p}, [iep])$.

Given (H1), (M1), (R1), (U1), (Z1), (D2), and (H2), by Lemma 5.3.47 we have (J2) $\mathcal{L}_1 :: \mathcal{L}_2 :: \mathcal{L}_3 :: \mathcal{L}_4 :: \mathcal{L}_5 :: \mathcal{L}_6 :: \mathcal{L}_7 = \mathcal{L}_1' :: \mathcal{L}_2' :: \mathcal{L}_3' :: \mathcal{L}_4' :: \mathcal{L}_5' :: \mathcal{L}_6' :: \mathcal{L}_7'$.

Given (F2), (X1), (J2), and (I2), by Definition 5.3.2, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3::\mathcal{L}_4::\mathcal{L}_5::\mathcal{L}_6::\mathcal{L}_7}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[iepd])}$ $((1, \gamma^1, \sigma_6^1, \Delta_6^1, \mathrm{acc}, \mathrm{skip})\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_6^{\mathrm{q}}, \Delta_6^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, \mathrm{if}\ (e)\ s_1\ \mathrm{else}\ s_2))$ $\Downarrow^{\mathcal{L}_1::\mathcal{L}_2::\mathcal{L}_3::\mathcal{L}_4::\mathcal{L}_5::\mathcal{L}_6::\mathcal{L}_7}_{\mathcal{D}_1::\mathcal{D}_2::\mathcal{D}_3::(\mathrm{p},[iepd])}$ $((1, \gamma^1, \sigma_6^1, \Delta_6^1, \mathrm{acc}, \mathrm{skip})\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_6^{\mathrm{q}}, \Delta_6^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip}))$ by SMC^2 rule Private If Else (Location Tracking), we have $\{(e) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (B) $((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, e)\ \|\ ...\ \|\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((1, \gamma^1, \sigma_1^1,$

$\Delta_1^1, \text{acc}, n^1) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma_1^{\text{q}}, \Delta_1^{\text{q}}, \text{acc}, n^{\text{q}}))$, (C) $\{\text{Extract}(s_1,\ s_2, \gamma^{\text{p}}) = (x_{list}, 1)\}_{\text{p}=1}^{\text{q}}$,
(D) $\{\text{Initialize}(\Delta_1^{\text{p}},\ x_{list},\ \gamma^{\text{p}}, \sigma_1^{\text{p}}, n^{\text{p}},\ \text{acc}+1) = (\gamma_1^{\text{p}}, \sigma_2^{\text{p}}, \Delta_2^{\text{p}}, \bar{l}_2^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (E) $((1, \gamma_1^1, \sigma_2^1, \Delta_2^1, \text{acc}+1, s_1) \parallel ... \parallel (\text{q}, \gamma_1^{\text{q}}, \sigma_2^{\text{q}}, \Delta_2^{\text{q}}, \text{acc}+1, s_1)) \Downarrow_{\mathcal{D}_2}^{\mathcal{L}_3} ((1, \gamma_2^1, \sigma_3^1, \Delta_3^1, \text{acc}+1, \text{skip}) \parallel ... \parallel (\text{q}, \gamma_2^{\text{q}}, \sigma_3^{\text{q}}, \Delta_3^{\text{q}}, \text{acc}+1, \text{skip}))$,
(F) $\{\text{Restore}(\sigma_3^{\text{p}},\ \Delta_3^{\text{p}},\ \text{acc}+1) = (\sigma_4^{\text{p}}, \Delta_4^{\text{p}}, \bar{l}_4^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (G) $((1, \gamma_1^1, \sigma_4^1, \Delta_4^1, \text{acc}+1, s_2) \parallel ... \parallel (\text{q}, \gamma_1^{\text{q}}, \sigma_4^{\text{q}}, \Delta_4^{\text{q}}, \text{acc}+1, s_2)) \Downarrow_{\mathcal{D}_3}^{\mathcal{L}_5} ((1, \gamma_3^1, \sigma_5^1, \Delta_5^1, \text{acc}+1, \text{skip}) \parallel ... \parallel (\text{q}, \gamma_3^{\text{q}}, \sigma_5^{\text{q}}, \Delta_5^{\text{q}}, \text{acc}+1, \text{skip}))$,
(H) $\{\text{Resolve_Retrieve}(\gamma_1^{\text{p}}, \sigma_5^{\text{p}}, \Delta_5^{\text{p}}, \text{acc}+1) = ([(v_{t1}^{\text{p}}, v_{e1}^{\text{p}}), ..., (v_{tm}^{\text{p}}, v_{em}^{\text{p}})], n_1^{\text{p}}, \bar{l}_6^{\text{p}})\}_{\text{p}=1}^{\text{q}}$,
(I) $\text{MPC}_{resolve}([n_1^1, ..., n_1^{\text{q}}], [[(v_{t1}^1, v_{e1}^1), ..., (v_{tm}^1, v_{em}^1)], ..., [(v_{t1}^{\text{q}}, v_{e1}^{\text{q}}), ..., (v_{tm}^{\text{q}}, v_{em}^{\text{q}})]]) = [[v_1^1, ..., v_m^1], ..., [v_1^{\text{q}}, ..., v_m^{\text{q}}]]$ (J) $\{\text{Resolve_Store}(\Delta_5^{\text{p}}, \sigma_5^{\text{p}}, \text{acc}+1, [v_1^{\text{p}}, ..., v_m^{\text{p}}]) = (\sigma_6^{\text{p}}, \Delta_6^{\text{p}}, \bar{l}_7^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (K) $\mathcal{L}_2 = (1, \bar{l}_2^1) \parallel ... \parallel (\text{q}, \bar{l}_2^{\text{q}})$, (L) $\mathcal{L}_4 = (1, \bar{l}_4^1) \parallel ... \parallel (\text{q}, \bar{l}_4^{\text{q}})$, (M) $\mathcal{L}_6 = (1, \bar{l}_6^1) \parallel ... \parallel (\text{q}, \bar{l}_6^{\text{q}})$, and (N) $\mathcal{L}_7 = (1, \bar{l}_7^1) \parallel ... \parallel (\text{q}, \bar{l}_7^{\text{q}})$.

Given (O) $\Sigma \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{if } (e)\ s_1 \text{ else } s_2)) \Downarrow_{\mathcal{D}_1'::\mathcal{D}_2'::\mathcal{D}_3'::(\text{p},[d])}^{\mathcal{L}_1'::\mathcal{L}_2'::\mathcal{L}_3'::\mathcal{L}_4'::\mathcal{L}_5'::\mathcal{L}_6'::\mathcal{L}_7'} ((1, \gamma^1, \sigma_6'^1, \Delta_6'^1, \text{acc}, \text{skip}) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma_6'^{\text{q}}, \Delta_6'^{\text{q}}, \text{acc}, \text{skip}))$ and (A), by Lemma 5.2.87 we have (P) $d = iepd$.

Given (O) and (P), by SMC^2 rule Private If Else (Location Tracking), we have $\{(e) \vdash \gamma^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (Q) $((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, e) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, e)) \Downarrow_{\mathcal{D}_1'}^{\mathcal{L}_1'} ((1, \gamma^1, \sigma_1'^1, \Delta_1'^1, \text{acc}, n'^1) \parallel ... \parallel (\text{q}, \gamma^{\text{q}}, \sigma_1'^{\text{q}}, \Delta_1'^{\text{q}}, \text{acc}, n'^{\text{q}}))$,
(R) $\{\text{Extract}(s_1,\ s_2,\ \gamma^{\text{p}}) = (x_{list}', 1)\}_{\text{p}=1}^{\text{q}}$, (S) $\{\text{Initialize}(\Delta_1'^{\text{p}},\ x_{list}',\ \gamma'^{\text{p}}, \sigma_1'^{\text{p}}, n'^{\text{p}},\ \text{acc}+1) = (\gamma_1'^{\text{p}}, \sigma_2'^{\text{p}}, \Delta_2'^{\text{p}}, \bar{l}_2'^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (T) $((1, \gamma_1'^1, \sigma_2'^1, \Delta_2'^1, \text{acc}+1, s_1) \parallel ... \parallel (\text{q}, \gamma_1'^{\text{q}}, \sigma_2'^{\text{q}}, \Delta_2'^{\text{q}}, \text{acc}+1, s_1)) \Downarrow_{\mathcal{D}_2'}^{\mathcal{L}_3'} ((1, \gamma_2'^1, \sigma_3'^1, \Delta_3'^1, \text{acc}+1, \text{skip}) \parallel ... \parallel (\text{q}, \gamma_2'^{\text{q}}, \sigma_3'^{\text{q}}, \Delta_3'^{\text{q}}, \text{acc}+1, \text{skip}))$, (U) $\{\text{Restore}(\sigma_3'^{\text{p}},\ \Delta_3'^{\text{p}},\ \text{acc}+1) = (\sigma_4'^{\text{p}}, \Delta_4'^{\text{p}}, \bar{l}_4'^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (V) $((1, \gamma_1'^1, \sigma_4'^1, \Delta_4'^1, \text{acc}+1, s_2) \parallel ... \parallel (\text{q}, \gamma_1'^{\text{q}}, \sigma_4'^{\text{q}}, \Delta_4'^{\text{q}}, \text{acc}+1, s_2)) \Downarrow_{\mathcal{D}_3'}^{\mathcal{L}_5'} ((1, \gamma_3'^1, \sigma_5'^1, \Delta_5'^1, \text{acc}+1, \text{skip}) \parallel ... \parallel (\text{q}, \gamma_3'^{\text{q}}, \sigma_5'^{\text{q}}, \Delta_5'^{\text{q}}, \text{acc}+1, \text{skip}))$, (W) $\{\text{Resolve_Retrieve}(\gamma_1'^{\text{p}}, \sigma_5'^{\text{p}}, \Delta_5'^{\text{p}}, \text{acc}+1) = ([(v_{t1}'^{\text{p}}, v_{e1}'^{\text{p}}), ..., (v_{tm}'^{\text{p}}, v_{em}'^{\text{p}})], n_1'^{\text{p}}, \bar{l}_6'^{\text{p}})\}_{\text{p}=1}^{\text{q}}$,
(X) $\text{MPC}_{resolve}([n_1'^1, ..., n_1'^{\text{q}}], [[(v_{t1}'^1, v_{e1}'^1), ..., (v_{tm}'^1, v_{em}'^1)], ..., [(v_{t1}'^{\text{q}}, v_{e1}'^{\text{q}}), ..., (v_{tm}'^{\text{q}}, v_{em}'^{\text{q}})]]) = [[v_1'^1, ..., v_m'^1], ..., [v_1'^{\text{q}}, ..., v_m'^{\text{q}}]]$
(Y) $\{\text{Resolve_Store}(\Delta_5'^{\text{p}}, \sigma_5'^{\text{p}}, \text{acc}+1, [v_1'^{\text{p}}, ..., v_m'^{\text{p}}]) = (\sigma_6'^{\text{p}}, \Delta_6'^{\text{p}}, \bar{l}_7'^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (Z) $\mathcal{L}_2' = (1, \bar{l}_2'^1) \parallel ... \parallel (\text{q}, \bar{l}_2'^{\text{q}})$,
(A1) $\mathcal{L}_4' = (1, \bar{l}_4'^1) \parallel ... \parallel (\text{q}, \bar{l}_4'^{\text{q}})$, (B1) $\mathcal{L}_6' = (1, \bar{l}_6'^1) \parallel ... \parallel (\text{q}, \bar{l}_6'^{\text{q}})$, and (C1) $\mathcal{L}_7' = (1, \bar{l}_7'^1) \parallel ... \parallel (\text{q}, \bar{l}_7'^{\text{q}})$.

Given (B) and (Q), by the inductive hypothesis we have (D1) $\{\sigma_1^{\text{p}} = \sigma_1'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (E1) $\{\Delta_1^{\text{p}} = \Delta_1'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (F1) $\{n^{\text{p}} = n'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (G1) $\mathcal{D}_1 = \mathcal{D}_1'$, and (H1) $\mathcal{L}_1 = \mathcal{L}_1'$.

Given (C) and (R), by Lemma 5.3.16 we have (I1) $x_{list} = x_{list}'$.

Given (D), (S), (D1), (E1), and (F1), by Lemma 5.3.21 we have (J1) $\{\gamma_1^{\text{p}} = \gamma_1'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (K1) $\{\sigma_2^{\text{p}} = \sigma_2'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (L1) $\{\Delta_2^{\text{p}} = \Delta_2'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, and (M1) $\{\bar{l}_2^{\text{p}} = \bar{l}_2'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$.

Given (M1), (K), and (Z), by Lemma 5.3.57 and Definition 5.3.10 we have (N1) $\mathcal{L}_2 = \mathcal{L}'_2$.

Given (E), (T), (J1), (K1), and (L1), by the inductive hypothesis we have $\{\gamma_2^{\mathrm{p}} = \gamma_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (O1) $\{\sigma_3^{\mathrm{p}} = \sigma_3'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (P1) $\{\Delta_3^{\mathrm{p}} = \Delta_3'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (Q1) $\mathcal{D}_2 = \mathcal{D}'_2$, and (R1) $\mathcal{L}_3 = \mathcal{L}'_3$.

Given (F), (U), (P1), and (O1), by Lemma 5.3.22 we have (S1) $\{\sigma_4^{\mathrm{p}} = \sigma_4'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (T1) $\{\Delta_4^{\mathrm{p}} = \Delta_4'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (U1) $\{\bar{l}_4^{\mathrm{p}} = \bar{l}_4'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (U1), (L), and (A1), by Lemma 5.3.58 and Definition 5.3.10 we have (V1) $\mathcal{L}_4 = \mathcal{L}'_4$.

Given (G), (V), (J1), (S1), and (T1), by the inductive hypothesis we have $\{\gamma_3^{\mathrm{p}} = \gamma_3'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (W1) $\{\sigma_5^{\mathrm{p}} = \sigma_5'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (X1) $\{\Delta_5^{\mathrm{p}} = \Delta_5'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (Y1) $\mathcal{D}_3 = \mathcal{D}'_3$, and (Z1) $\mathcal{L}_5 = \mathcal{L}'_5$.

Given (H), (W), (J1), (W1), and (X1), by Lemma 5.3.23 we have (A2) $\{[(v_{t1}^{\mathrm{p}}, v_{e1}^{\mathrm{p}}), ..., (v_{tm}^{\mathrm{p}}, v_{em}^{\mathrm{p}})] = [(v_{t1}'^{\mathrm{p}}, v_{e1}'^{\mathrm{p}}), ..., (v_{tm}'^{\mathrm{p}}, v_{em}'^{\mathrm{p}})]\}_{\mathrm{p}=1}^{\mathrm{q}}$, (B2) $\{n_1^{\mathrm{p}} = n_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (C2) $\{\bar{l}_6^{\mathrm{p}} = \bar{l}_6'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (M), (B1), and (C2), by Lemma 5.3.59 and Definition 5.3.10 we have (D2) $\mathcal{L}_6 = \mathcal{L}'_6$.

Given (I), (X), (B2), and (A2), by Axiom 5.3.10 we have $[[v_1^1, ..., v_m^1], ..., [v_1^{\mathrm{q}}, ..., v_m^{\mathrm{q}}]] = [[v_1'^1, ..., v_m'^1], ..., [v_1'^{\mathrm{q}}, ..., v_m'^{\mathrm{q}}]]$ and therefore (E2) $\{[v_1^{\mathrm{p}}, ..., v_m^{\mathrm{p}}] = [v_1'^{\mathrm{p}}, ..., v_m'^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (J), (Y), (X1), (W1), and (E2), by Lemma 5.3.24 we have (F2) $\{\sigma_6^{\mathrm{p}} = \sigma_6'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (G2) $\{\Delta_6^{\mathrm{p}} = \Delta_6'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, and (H2) $\{\bar{l}_7^{\mathrm{p}} = \bar{l}_7'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (N), (C1), and (H2), by Lemma 5.3.60 and Definition 5.3.10 we have (I2) $\mathcal{L}_7 = \mathcal{L}'_7$.

Given (G1), (Q1), (Y1), and (P), by Lemma 5.3.38 we have (J2) $\mathcal{D}_1 :: \mathcal{D}_2 :: \mathcal{D}_3 :: (\mathrm{p}, [iepd]) = \mathcal{D}_1 :: \mathcal{D}'_2 :: \mathcal{D}'_3(\mathrm{p}, [iepd])$.

Given (H1), (N1), (R1), (V1), (Z1), (D2), and (I2), by Lemma 5.3.47 we have (K2) $\mathcal{L}_1 :: \mathcal{L}_2 :: \mathcal{L}_3 :: \mathcal{L}_4 :: \mathcal{L}_5 :: \mathcal{L}_6 :: \mathcal{L}_7 = \mathcal{L}'_1 :: \mathcal{L}'_2 :: \mathcal{L}'_3 :: \mathcal{L}'_4 :: \mathcal{L}'_5 :: \mathcal{L}'_6 :: \mathcal{L}'_7$.

Given (F2), (G2), (J2), and (K2), by Definition 5.3.2, we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,0)])}_{\mathcal{D}_1::(\mathrm{p},[da])} ((\mathrm{p}, \gamma_1,\ \sigma_3,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,0)])}_{\mathcal{D}_1::(\mathrm{p},[da])} ((\mathrm{p}, \gamma_1,\ \sigma_3,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Public Array Declaration we have $((ty = \text{public } bty) \wedge ((bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{int}))) \vee (ty = \text{char})$, $(e) \nvdash \gamma$, $\alpha > 0$, $\mathrm{acc} = 0$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C)$, $\Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \alpha) \parallel C_1)$, (C) $l = \phi()$, (D) $l_1 = \phi()$, (E) $\omega_1 = \mathrm{EncodeArr}(\text{public } bty, 0, \alpha, \mathrm{NULL})$, (F) $\gamma_1 = \gamma[x \rightarrow (l,\ \text{public const } bty*)]$, (G) $\omega = \mathrm{EncodePtr}(\text{public const } bty*,\ [1,\ [(l_1, 0)],\ [1],\ 1])$, (H) $\sigma_2 = \sigma_1[l \rightarrow (\omega,\ \text{public const } bty*,\ 1,\ \mathrm{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))]$, and (I) $\sigma_3 = \sigma_2[l_1 \rightarrow (\omega_1,\ \text{public } bty,\ \alpha,\ \mathrm{PermL}(\text{Freeable}, \text{public } bty, \text{public}, \alpha))]$.

Given (J) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x[e]) \parallel C) \Downarrow^{\mathcal{L}'_1::(\mathrm{p},[(l',0),(l'_1,0)])}_{\mathcal{D}'_1::(\mathrm{p},[d])} ((\mathrm{p}, \gamma'_1,\ \sigma'_3,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_1)$ and (A), by Lemma 5.2.87 we have (K) $d = da$.

Given (J) and (K), by SMC^2 rule Public Array Declaration we have $((ty = \text{public } bty) \wedge ((bty = \text{float}) \vee (bty = \text{char}) \vee (bty = \text{int}))) \vee (ty = \text{char})$, $(e) \nvdash \gamma$, $\alpha' > 0$, $\mathrm{acc} = 0$, (L) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C)$, $\Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta,\ \mathrm{acc},\ \alpha') \parallel C'_1)$, (M) $l' = \phi()$, (N) $l'_1 = \phi()$, (O) $\omega'_1 = \mathrm{EncodeArr}(\text{public } bty, 0, \alpha', \mathrm{NULL})$, (P) $\gamma'_1 = \gamma[x \rightarrow (l',\ \text{public const } bty*)]$, (Q) $\omega' = \mathrm{EncodePtr}(\text{public const } bty*,\ [1,\ [(l'_1, 0)],\ [1],\ 1])$, (R) $\sigma'_2 = \sigma'_1[l' \rightarrow (\omega',\ \text{public const } bty*,\ 1, \mathrm{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))]$, and (S) $\sigma'_3 = \sigma'_2[l'_1 \rightarrow (\omega'_1,\ \text{public } bty,\ \alpha', \mathrm{PermL}(\text{Freeable}, \text{public } bty, \text{public}, \alpha'))]$.

Given (B) and (L), by the inductive hypothesis we have (T) $\sigma_1 = \sigma'_1$, (U) $\alpha = \alpha'$, (V) $\mathcal{D}_1 = \mathcal{D}'_1$, (W) $\mathcal{L}_1 = \mathcal{L}'_1$, and (X) $C_1 = C'_1$.

Given (C), (D), (M), and (N), by Axiom 5.3.4 we have (Y) $l = l'$ and (Z) $l_1 = l'_1$.

Given (E) , (O), and (U), by Lemma 5.3.31 we have (A1) $\omega_1 = \omega'_1$.

Given (F), (P), and (Y), by Definition 5.3.3 we have (B1) $\gamma_1 = \gamma'_1$.

Given (G), (Q), and (Z), by Lemma 5.3.32 we have (C1) $\omega = \omega'$.

Given (H), (R), (T), (Y), and (C1), by Definition 5.3.4 we have (D1) $\sigma_2 = \sigma_2'$.

Given (I), (S), (D1), (Z), and (A1), by Definition 5.3.4 we have (E1) $\sigma_3 = \sigma_3'$.

Given (X) and $(\mathrm{p}, [da])$, by Lemma 5.3.38 we have (F1) $\mathcal{D}_1 :: (\mathrm{p}, [da]) = \mathcal{D}_1' :: (\mathrm{p}, [da])$.

Given (H) and (I), by Lemma 5.3.51 we have accessed (G1) $(\mathrm{p}, [(l, 0)])$ and (H1) $(\mathrm{p}, [(l_1, 0)])$. Given (G1) and (H1), by Lemmas 5.3.44 and 5.3.45 we have (I1) $(\mathrm{p}, [(l, 0), (l_1, 0)])$. Given (R) and (S), by Lemma 5.3.51 we have accessed (J1) $(\mathrm{p}, [(l', 0)])$ and (K1) $(\mathrm{p}, [(l_1', 0)])$. Given (J1) and (K1), by Lemmas 5.3.44 and 5.3.45 we have (L1) $(\mathrm{p}, [(l', 0), (l_1', 0)])$. Given (I1), (L1), (Y), (Z), and (W), by Lemma 5.3.47 we have (M1) $\mathcal{L}_1 :: (\mathrm{p}, [(l, 0), (l_1, 0)]) = \mathcal{L}_1' :: (\mathrm{p}, [(l', 0), (l_1', 0)])$.

Given (B1), (E1), (X), (F1), and (M1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_1,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [da1])} ((\mathrm{p}, \gamma_1,\ \sigma_3,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_1,0)])}_{\mathcal{D}_1 :: (\mathrm{p}, [da])} ((\mathrm{p}, \gamma_1,\ \sigma_3,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_1,i)])}_{\mathcal{D}_1 :: (\mathrm{p}, [ra1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_i) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_1,i)])}_{\mathcal{D}_1 :: (\mathrm{p}, [ra1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_i) \parallel C_1)$ by SMC^2 rule Private Array Read Public Index we have $0 \leq i \leq \alpha - 1$, $(e) \nvdash \gamma$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ i) \parallel C_1)$, (C) $\gamma(x) = (l,\ \text{private const}\ bty*)$, (D) $\sigma_1(l) = (\omega,\ \text{private const}\ bty*, 1, \mathrm{PermL}(\text{Freeable}, \text{private const}\ bty*, \text{private}, 1))$, (E) $\mathrm{DecodePtr}(\text{private const}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$, (F) $\sigma_1(l_1) = (\omega_1, \text{private}\ bty, \alpha, \mathrm{PermL}(\text{Freeable}, \text{private}\ bty, \text{private}, \alpha))$, and (G) $\mathrm{DecodeArr}(\text{private}\ bty,\ i,\ \omega_1) = n_i$.

Given (H) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1' :: (\mathrm{p}, [(l',0),(l_1',i')])}_{\mathcal{D}_1' :: (\mathrm{p}, [d])} ((\mathrm{p}, \gamma,\ \sigma_1',\ \Delta_1',\ \mathrm{acc},\ n_{i'}') \parallel C_1')$ and (A), by Lemma 5.2.87 we have (I) $d = ra1$.

Given (H) and (I), by SMC^2 rule Private Array Read Public Index we have $0 \leq i' \leq \alpha' - 1$, $(e) \nvdash \gamma$, (J)

$((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ i') \parallel C'_1)$, (K) $\gamma(x) = (l',\ \text{private const } bty'*)$, (L) $\sigma'_1(l') = (\omega',\ \text{private const } bty'*, 1, \mathrm{PermL}(\mathrm{Freeable}, \text{private const } bty'*, \text{private}, 1))$, (M) $\mathrm{DecodePtr}(\text{private const } bty'*,\ 1,\ \omega') = [1,\ [(l'_1, 0)],\ [1],\ 1]$, (N) $\sigma'_1(l'_1) = (\omega'_1, \text{private } bty', \alpha', \mathrm{PermL}(\mathrm{Freeable}, \text{private } bty', \text{private}, \alpha'))$, and (O) $\mathrm{DecodeArr}(\text{private } bty', n'_{i'}$.

Given (B) and (J), by the inductive hypothesis we have (P) $\sigma_1 = \sigma'_1$, (Q) $\Delta_1 = \Delta'_1$, (R) $i = i'$, (S) $\mathcal{D}_1 = \mathcal{D}'_1$, (T) $\mathcal{L}_1 = \mathcal{L}'_1$, and (U) $C_1 = C'_1$.

Given (C) and (K), by Definition 5.3.3 we have (V) $l = l'$ and (W) $bty = bty'$.

Given (D), (L), (P), and (V), by Definition 5.3.4 we have (X) $\omega = \omega'$.

Given (E), (M), (W), and (X), by Lemma 5.3.26 we have (Y) $l_1 = l'_1$.

Given (F), (N), (P), and (Y), by Definition 5.3.4 we have (Z) $\omega_1 = \omega'_1$ and (A1) $\alpha = \alpha'$.

Given (G), (O), (W), (R), and (Z), by Lemma 5.3.27 we have (B1) $n_i = n'_{i'}$.

Given (S) and $(\mathrm{p}, [ra1])$, by Lemma 5.3.38 we have (C1) $\mathcal{D}_1 :: (\mathrm{p}, [ra1]) = \mathcal{D}'_1 :: (\mathrm{p}, [ra1])$.

Given (D) and (E), by Lemma 5.3.62 we have accessed location (D1) $(\mathrm{p}, [(l, 0)])$. Given (F) and (G), by Lemma 5.3.63 we have accessed location (E1) $(\mathrm{p}, [(l_1, i)])$. Given (D1) and (E1), by Lemmas 5.3.44 and 5.3.45 we have (F1) $(\mathrm{p}, [(l, 0), (l_1, i)])$.

Given (L) and (M), by Lemma 5.3.62 we have accessed location (G1) $(\mathrm{p}, [(l', 0)])$. Given (N) and (O), by Lemma 5.3.63 we have accessed location (H1) $(\mathrm{p}, [(l'_1, i')])$. Given (D1) and (E1), by Lemmas 5.3.44 and 5.3.45 we have (I1) $(\mathrm{p}, [(l', 0), (l'_1, i')])$.

Given (F1), (I1), (V), (Y), (R), and (T), by Lemma 5.3.47 we have
(J1) $\mathcal{L}_1 :: (\mathrm{p}, [(l, 0), (l_1, i)]) = \mathcal{L}'_1 :: (\mathrm{p}, [(l', 0), (l'_1, i')])$.

Given (P), (Q), (B1), (U), (C1), and (J1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::(\mathrm{p},[ra])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_i) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::(\mathrm{p},[ra1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n_i) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::(\mathrm{p},[rao])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::(\mathrm{p},[rao])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$ by SMC^2 rule Public Array Read Out of Bounds Public Index we have $(e) \nvdash \gamma$, $(i < 0) \vee (i \geq \alpha)$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ i) \parallel C_1)$, (C) $\gamma(x) = (l,\ \text{public const } bty*)$, (D) $\sigma_1(l) = (\omega,\ \text{public const } bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, \text{public const } bty*, \text{public}, 1))$, (E) $\mathrm{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$, (F) $\sigma_1(l_1) = (\omega_1, \text{public } bty, \alpha, \mathrm{PermL}(\mathrm{Freeable}, \text{public } bty, \text{public}, \alpha))$, (G) $\mathrm{ReadOOB}(i, \alpha, l_1, \text{public } bty, \sigma_1) = (n, 1, (l_2, \mu))$.

Given (H) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}'_1::(\mathrm{p},[(l',0),(l'_2,\mu')])}_{\mathcal{D}'_1::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ n') \parallel C'_1)$ and (A), by Lemma 5.2.87 we have (I) $d = rao$.

Given (H) and (I), by SMC^2 rule Public Array Read Out of Bounds Public Index we have $(e) \nvdash \gamma$, $(i < 0) \vee (i \geq \alpha)$, (J) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ i') \parallel C'_1)$, (K) $\gamma(x) = (l',\ \text{public const } bty'*)$, (L) $\sigma'_1(l') = (\omega',\ \text{public const } bty'*, 1, \mathrm{PermL}(\mathrm{Freeable}, \text{public const } bty'*, \text{public}, 1))$, (M) $\mathrm{DecodePtr}(\text{public const } bty'*,\ 1,\ \omega') = [1,\ [(l'_1, 0)],\ [1],\ 1]$, (N) $\sigma'_1(l'_1) = (\omega'_1, \text{public } bty', \alpha', \mathrm{PermL}(\mathrm{Freeable}, \text{public } bty', \text{public}, \alpha'))$, (O) $\mathrm{ReadOOB}(i', \alpha', l'_1, \text{public } bty', \sigma'_1) = (n', 1, (l'_2, \mu'))$.

Given (B) and (J), by the inductive hypothesis we have (P) $\sigma_1 = \sigma'_1$, (Q) $\Delta_1 = \Delta'_1$, (R) $i = i'$, (S) $\mathcal{D}_1 = \mathcal{D}'_1$, (T) $\mathcal{L}_1 = \mathcal{L}'_1$, and (U) $C_1 = C'_1$.

Given (C) and (K), by Definition 5.3.3 we have (V) $l = l'$ and (W) $bty = bty'$.

Given (D), (L), (P), and (V), by Definition 5.3.4 we have (X) $\omega = \omega'$.

Given (E), (M), (W), and (X), by Lemma 5.3.26 we have (Y) $l_1 = l'_1$.

Given (F), (N), (P), and (Y), by Definition 5.3.4 we have (Z) $\omega_1 = \omega_1'$ and (A1) $\alpha = \alpha'$.

Given (G), (O), (R), (A1), (Y), (X), and (P), by Lemma 5.3.11 we have (B1) $n = n'$ and (C1) $(l_2, \mu) = (l_2', \mu')$.

Given (S) and $(\mathrm{p}, [rao])$, by Lemma 5.3.38 we have (D1) $\mathcal{D}_1 :: (\mathrm{p}, [rao]) = \mathcal{D}_1' :: (\mathrm{p}, [rao])$.

Given (D) and (E) by Lemma 5.3.62 we have accessed location (E1) $(\mathrm{p}, [(l, 0)])$. Given (G), by Lemma 5.3.49 we have accessed location (F1) $(\mathrm{p}, [(l_2, \mu)])$. Given (E1) and (F1), by Lemmas 5.3.44 and 5.3.45 we have (G1) $(\mathrm{p}, [(l, 0), (l_2, \mu)])$.

Given (L) and (M) by Lemma 5.3.62 we have accessed location (H1) $(\mathrm{p}, [(l', 0)])$. Given (O), by Lemma 5.3.49 we have accessed location (I1) $(\mathrm{p}, [(l_2', \mu')])$. Given (H1) and (I1), by Lemmas 5.3.44 and 5.3.45 we have (J1) $(\mathrm{p}, [(l', 0), (l_2', \mu')])$.

Given (G1), (J1), (T), (V), and (C1), by Lemma 5.3.47 we have
(K1) $\mathcal{L}_1 :: (\mathrm{p}, [(l, 0), (l_2, \mu)]) = \mathcal{L}_1' :: (\mathrm{p}, [(l', 0), (l_2', \mu')])$.

Given (P), (Q), (B1), (U), (D1), and (K1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_2,\mu)])}_{\mathcal{D}_1 :: (\mathrm{p}, [rao1])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e]) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_2,\mu)])}_{\mathcal{D}_1 :: (\mathrm{p}, [rao])} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2 :: (\mathrm{p}, [(l,0),(l_2,\mu)])}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wao2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1 :: \mathcal{L}_2 :: (\mathrm{p}, [(l,0),(l_2,\mu)])}_{\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wao2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$ by SMC^2 rule Private Array Write Out of Bounds Public Index Private Value we have $(e_1) \nvdash \gamma$, $(e_2) \vdash \gamma$, $(i < 0) \vee (i \geq \alpha)$ (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ i) \parallel C_1)$, (C) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n) \parallel C_2)$, (D) $\gamma(x) = (l, \text{private const } bty*)$, (E) $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \mathrm{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$, (F) $\mathrm{DecodePtr}(\text{private const } bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1], 1]$, (G) $\sigma_2(l_1) = (\omega_1,\ \text{private } bty,\ \alpha,\ \mathrm{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha))$, and (H) $\mathrm{WriteOOB}(n,\ i,\ \alpha,\ l_1,$

private $bty,\ \sigma_2,\ \Delta_2, \text{acc}) = (\sigma_3,\ \Delta_3, 1, (l_2, \mu))$.

Given (I) $\Sigma \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1' :: \mathcal{L}_2' :: (\text{p}, [(l', 0), (l_2', \mu')])}_{\mathcal{D}_1' :: \mathcal{D}_2' :: (\text{p}, [wao2])} ((\text{p}, \gamma,\ \sigma_3',\ \Delta_3',\ \text{acc},\ \text{skip}) \parallel C_2')$ and (A), by Lemma 5.2.87 we have (J) $d = wao2$.

Given (I) and (J), by SMC^2 rule Private Array Write Out of Bounds Public Index Private Value we have $(e_1) \nvdash \gamma$, $(e_2) \vdash \gamma$, $(i' < 0) \vee (i' \geq \alpha')$, (K) $((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ e_1) \parallel C) \Downarrow^{\mathcal{L}_1'}_{\mathcal{D}_1'} ((\text{p}, \gamma,\ \sigma_1',\ \Delta_1',\ \text{acc},\ i') \parallel C_1')$,
(L) $((\text{p}, \gamma,\ \sigma_1',\ \Delta_1',\ \text{acc},\ e_2') \parallel C_1') \Downarrow^{\mathcal{L}_2'}_{\mathcal{D}_2'} ((\text{p}, \gamma,\ \sigma_2',\ \Delta_2',\ \text{acc},\ n') \parallel C_2')$, (M) $\gamma(x) = (l', \text{private const } bty'*)$,
(N) $\sigma_2'(l') = (\omega', \text{private const } bty'*, 1, \text{PermL}(\text{Freeable}, \text{private const } bty'*, \text{private}, 1))$,
(O) $\text{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1,\ [(l_1', 0)],\ [1],\ 1]$,
(P) $\sigma_2'(l_1') = (\omega_1',\ \text{private } bty',\ \alpha',\ \text{PermL}(\text{Freeable}, \text{private } bty', \text{private}, \alpha'))$, and
(Q) $\text{WriteOOB}(n',\ i',\ \alpha',\ l_1',\ \text{private } bty',\ \sigma_2',\ \Delta_2', \text{acc}) = (\sigma_3',\ \Delta_3', 1, (l_2', \mu'))$.

Given (B) and (K), by the inductive hypothesis we have (R) $\sigma_1 = \sigma_1'$, (S) $\Delta_1 = \Delta_1'$, (T) $i = i'$, (U) $\mathcal{D}_1 = \mathcal{D}_1'$, (V) $\mathcal{L}_1 = \mathcal{L}_1'$, and (W) $C_1 = C_1'$.

Given (C), (L), (R), (S), and (W), by the inductive hypothesis we have (X) $\sigma_2 = \sigma_2'$, (Y) $\Delta_2 = \Delta_2'$, (Z) $n = n'$, (A1) $\mathcal{D}_2 = \mathcal{D}_2'$, (B1) $\mathcal{L}_2 = \mathcal{L}_2'$, and (C1) $C_2 = C_2'$.

Given (D) and (M), by Definition 5.3.3 we have (D1) $l = l'$ and (E1) $bty = bty'$.

Given (E), (N), (X), and (D1), by Definition 5.3.4 we have (F1) $\omega = \omega'$.

Given (F), (O), (E1), and (F1), by Lemma 5.3.26 we have (G1) $l_1 = l_1'$.

Given (G), (P), (X), and (G1), by Definition 5.3.4 we have (H1) $\omega_1 = \omega_1'$ and (I1) $\alpha = \alpha'$.

Given (H), (Q), (Z), (T), (I1), (G1), (E1), (X), and (Y), by Lemma 5.3.12 we have (J1) $\sigma_3 = \sigma_3'$, (K1) $\Delta_3 = \Delta_3'$, and (L1) $(l_2, \mu) = (l_2', \mu')$.

Given (U), (A1), and $(\text{p}, [wao2])$, by Lemma 5.3.38 we have (M1) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\text{p}, [wao2]) = \mathcal{D}_1' :: \mathcal{D}_2' :: (\text{p}, [wao2])$.

Given (E) and (F) by Lemma 5.3.62 we have accessed location (N1) $(\text{p}, [(l, 0)])$. Given (H), by Lemma 5.3.50

we have accessed location (O1) $(\mathrm{p}, [(l_2, \mu)])$. Given (N1) and (O1), by Lemmas 5.3.44 and 5.3.45 we have (P1) $(\mathrm{p}, [(l, 0), (l_2, \mu)])$.

Given (N) and (O) by Lemma 5.3.62 we have accessed location (Q1) $(\mathrm{p}, [(l', 0)])$. Given (Q), by Lemma 5.3.50 we have accessed location (R1) $(\mathrm{p}, [(l'_2, \mu')])$. Given (Q1) and (R1), by Lemmas 5.3.44 and 5.3.45 we have (S1) $(\mathrm{p}, [(l', 0), (l'_2, \mu')])$.

Given (P1), (S1), (V), (B1), (D1), and (L1), by Lemma 5.3.47 we have (T1) $\mathcal{L}_1 :: \mathcal{L}_2 :: (\mathrm{p}, [(l, 0), (l_2, \mu)]) = \mathcal{L}'_1 :: \mathcal{L}'_2 :: (\mathrm{p}, [(l', 0), (l'_2, \mu')])$.

Given (J1), (K1), (C1), (M1), and (T1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wao])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wao2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wao1])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc}, \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_2,\mu)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wao2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$. We use Axiom 5.3.1 to prove that $\mathrm{encrypt}(n) = \mathrm{encrypt}(n')$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,0),\ldots,(l_1,\alpha-1)])}_{(\mathrm{p},[rea])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [n_0, ..., n_{\alpha-1}]) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,0),\ldots,(l_1,\alpha-1)])}_{(\mathrm{p},[rea])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [n_0, ..., n_{\alpha-1}]) \parallel C)$ by SMC^2 rule Read Entire Array we have (B) $\gamma(x) = (l, a \text{ const } bty*)$, (C) $\sigma(l) = (\omega, a \text{ const } bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, a \text{ const } bty*,\ a,\ 1))$, (D) $\mathrm{DecodePtr}(a \text{ const } bty*,\ 1,\ \omega) = [1,\ [(l_1,\ 0)],\ [1],\ 1]$, (E) $\sigma(l_1) = (\omega_1,\ a\ bty,\ \alpha, \mathrm{PermL}(\mathrm{Freeable}, a\ bty, a, \alpha))$, and (F) $\forall i \in \{0...\alpha - 1\} \quad \mathrm{DecodeArr}(a\ bty, i, \omega_1) = n_i$.

Given (G) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l',0),(l'_1,0),\ldots,(l'_1,\alpha'-1)])}_{(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [n'_0, ..., n'_{\alpha'-1}]) \parallel C)$ and

(A), by Lemma 5.2.87 we have (H) $d = rea$.

Given (G) and (H), by SMC^2 rule Read Entire Array we have (I) $\gamma(x) = (l', a' \text{ const } bty'*)$,
(J) $\sigma(l') = (\omega, a' \text{ const} bty'*, 1, \text{PermL}(\text{Freeable}, a' \text{ const } bty'*, a', 1))$,
(K) $\text{DecodePtr}(a' \text{ const } bty'*,\ 1,\ \omega') = [1, [(l'_1, 0)],\ [1],\ 1]$,
(L) $\sigma(l'_1) = (\omega'_1, a'\ bty', \alpha', \text{PermL}(\text{Freeable}, a'\ bty', a', \alpha'))$, and
(M) $\forall i' \in \{0...\alpha' - 1\}\ \text{DecodeArr}(a'\ bty', i', \omega'_1) = n'_{i'}$.

Given (B) and (I), by Definition 5.3.3 we have (N) $l = l'$, (O) $a = a'$ and (P) $bty = bty'$.

Given (C), (J), and (N), by Definition 5.3.4 we have (Q) $\omega = \omega'$.

Given (D), (K), (O), (P) and (Q), by Lemma 5.3.26 we have $[1,\ [(l_1, 0)],\ [1],\ 1] = [1,\ [(l'_1, 0)],\ [1],\ 1]$ and therefore (R) $l_1 = l'_1$.

Given (E), (L), and (R), by Definition 5.3.4 we have (S) $\omega_1 = \omega'_1$ and (T) $\alpha = \alpha'$.

Given (T), we have (U) $i = i'$ such that $i \in \{0...\alpha - 1\}$.

Given (O), (P), (U), and (S), by Lemma 5.3.27 we have (V) $\forall i, i' \in \{0...\alpha - 1\}$ such that $i = i'$, $n_i = n'_{i'}$. Therefore, we have (W) $[n_0, ..., n_{\alpha-1}] = [n'_0, ..., n'_{\alpha'-1}]$.

Given (C) and (D) by Lemma 5.3.62 we have accessed location (X) $(\text{p}, [(l, 0)])$. Given (E) and (F), by Lemma 5.3.63 we have accessed locations (Y) $(\text{p}, [(l_1, 0), ..., (l_1, \alpha - 1)])$. Given (X) and (Y), by Lemmas 5.3.44 and 5.3.45 we have (Z) $(\text{p}, [(l, 0), (l_1, 0), ..., (l_1, \alpha - 1)])$.

Given (J) and (K) by Lemma 5.3.62 we have accessed location (A1) $(\text{p}, [(l', 0)])$. Given (L) and (M), by Lemma 5.3.63 we have accessed locations (B1) $(\text{p}, [(l'_1, 0), ..., (l'_1, \alpha' - 1)])$. Given (A1) and (B1), by Lemmas 5.3.44 and 5.3.45 we have (C1) $(\text{p}, [(l', 0), (l'_1, 0), ..., (l'_1, \alpha' - 1)])$.

Given (N), (T), (R), (Z), and (C1), we have (D1) $(\text{p}, [(l, 0), (l_1, 0), ..., (l_1, \alpha - 1)]) = (\text{p}, [(l', 0), (l'_1, 0), ..., (l'_1, \alpha' - 1)])$.

Given (D1), (W), and (H), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::(\mathrm{p},[wea])} ((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1::(\mathrm{p},[wea])} ((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Public Array Write Entire Array we have $\alpha_e = \alpha$, $\mathrm{acc} = 0$, $(e) \nvdash \gamma$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ [n_0, ..., n_{\alpha_e-1}]) \parallel C_1)$, (C) $\gamma(x) = (l,\ \text{public const } bty*)$,

(D) $\sigma_1(l) = (\omega,\ \text{public const } bty*, 1, \mathrm{PermL}(\mathrm{Freeable}, \text{public const } bty*, \mathrm{public}, 1))$,

(E) $\mathrm{DecodePtr}(\text{public const } bty*,\ 1,\ \omega) = [1, [(l_1, 0)], [1], 1]$,

(F) $\sigma_1(l_1) = (\omega_1, \text{public } bty, \alpha, \mathrm{PermL}(\mathrm{Freeable}, \text{public } bty, \mathrm{public}, \alpha))$, and

(G) $\forall i \in \{0...\alpha - 1\}\ \mathrm{UpdateArr}(\sigma_{1+i},\ (l_1, i),\ n_i,\ \text{public } bty) = \sigma_{2+i}$.

Given (H) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}'_1::(\mathrm{p},[(l',0),(l'_1,0),...,(l'_1,\alpha'-1)])}_{\mathcal{D}'_1::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_{2+\alpha'-1},\ \Delta'_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_1)$ and (A), by Lemma 5.2.87 we have (I) $d = wea$.

Given (H) and (I), by SMC^2 rule Public Array Write Entire Array we have $\alpha'_e = \alpha'$, $\mathrm{acc} = 0$, $(e) \nvdash \gamma$, (J) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ [n'_0, ..., n'_{\alpha'_e-1}]) \parallel C'_1)$, (K) $\gamma(x) = (l',\ \text{public const } bty'*)$, (L) $\sigma'_1(l') = (\omega',\ \text{public const } bty'*, 1, \mathrm{PermL}(\mathrm{Freeable}, \text{public const } bty'*, \mathrm{public}, 1))$, (M) $\mathrm{DecodePtr}(\text{public const } bty'*,\ 1,\ \omega') = [1,\ [(l'_1, 0)],\ [1],\ 1]$, (N) $\sigma'_1(l'_1) = (\omega'_1, \text{public } bty', \alpha', \mathrm{PermL}(\mathrm{Freeable}, \text{public } bty', \mathrm{public}, \alpha'))$, and (O) $\forall i' \in \{0...\alpha' - 1\}\mathrm{UpdateArr}(\sigma'_{1+i'},\ (l'_1, i'),\ n'_i,\ \text{public } bty) = \sigma'_{2+i'}$.

Given (B) and (J), by the inductive hypothesis we have (P) $\sigma_1 = \sigma'_1$, (Q) $\Delta_1 = \Delta'_1$, (R) $[n_0, ..., n_{\alpha_e-1}] = [n'_0, ..., n'_{\alpha'_e-1}]$ and therefore (S) $\alpha_e = \alpha'_e$, (T) $\mathcal{D}_1 = \mathcal{D}'_1$, (U) $\mathcal{L}_1 = \mathcal{L}'_1$, and (V) $C_1 = C'_1$.

Given (C) and (K), by Definition 5.3.3 we have (W) $l = l'$ and (X) $bty = bty'$.

Given (D), (L), (P), and (W), by Definition 5.3.4 we have (Y) $\omega = \omega'$.

Given (E), (M), (X), and (Y), by Lemma 5.3.26 we have $[1,\ [(l_1, 0)],\ [1],\ 1] = [1,\ [(l'_1, 0)],\ [1],\ 1]$ and therefore (Z) $l_1 = l'_1$.

Given (F), (N), (P), and (Z), by Definition 5.3.4 we have (A1) $\omega_1 = \omega'_1$ and (B1) $\alpha = \alpha'$.

Given (B1), (S), $\alpha_e = \alpha$, and $\alpha'_e = \alpha'$, we have (C1) $i = i'$ such that $i \in \{0...\alpha - 1\}$.

Given (G), (O), (B1), (C1), (P), (Z), (X), (S), $\alpha_e = \alpha$, $\alpha'_e = \alpha'$, and (R), by Lemma 5.3.35 we have (D1) $\forall i, i' \in \{0...\alpha - 1\}$ such that $i = i'$, $\sigma_{1+i} = \sigma'_{1+i'}$ and (E1) $\sigma_{2+i} = \sigma'_{2+i'}$.

Given (D) and (E) by Lemma 5.3.62 we have accessed location (F1) $(\mathrm{p}, [(l, 0)])$. Given (G), by Lemma 5.3.67 we have accessed locations (G1) $(\mathrm{p}, [(l_1, 0), ..., (l_1, \alpha - 1)])$. Given (F1) and (G1), by Lemmas 5.3.44 and 5.3.45 we have (H1) $(\mathrm{p}, [(l, 0), (l_1, 0), ..., (l_1, \alpha - 1)])$.

Given (L) and (M) by Lemma 5.3.62 we have accessed location (I1) $(\mathrm{p}, [(l', 0)])$. Given (O), by Lemma 5.3.67 we have accessed locations (J1) $(\mathrm{p}, [(l'_1, 0), ..., (l'_1, \alpha' - 1)])$. Given (I1) and (J1), by Lemmas 5.3.44 and 5.3.45 we have (K1) $(\mathrm{p}, [(l', 0), (l'_1, 0), ..., (l'_1, \alpha' - 1)])$.

Given (U), (W), (Z), (B1), (H1), and (K1), by Lemma 5.3.47 we have (L1) $\mathcal{L}_1 :: (\mathrm{p}, [(l, 0), (l_1, 0), ..., (l_1, \alpha - 1)]) = \mathcal{L}'_1 :: (\mathrm{p}, [(l', 0), (l'_1, 0), ..., (l'_1, \alpha' - 1)])$.

Given (T) and (I), by Lemma 5.3.38 we have (M1) $\mathcal{D}_1 :: (\mathrm{p}, [wea]) = \mathcal{D}'_1 :: (\mathrm{p}, [wea])$.

Given (E1), (V), (L1), (M1), and (V), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wea1])} ((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wea])} ((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wea2])} ((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_1,0),...,(l_1,\alpha-1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wea])} ((\mathrm{p}, \gamma,\ \sigma_{2+\alpha-1},\ \Delta_1,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1)$. We use Axiom 5.3.1 to prove that $\mathrm{encrypt}(n) = \mathrm{encrypt}(n')$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wa])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wa])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$ by SMC^2 rule Public Array Write Public Value Public Index we have $(e_1, e_2) \nvdash \gamma$, $0 \leq i \leq \alpha - 1$, $\mathrm{acc} = 0$ (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ i) \parallel C_1)$, (C) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n) \parallel C_2)$, (D) $\gamma(x) = (l,\ \text{public const } bty*)$, (E) $\sigma_2(l) = (\omega, \text{public const } bty*, 1, \mathrm{PermL}(\text{Freeable}, \text{public const } bty*, \text{public}, 1))$, (F) $\mathrm{DecodePtr}(\text{public const } bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1],\ 1]$, (G) $\sigma_2(l_1) = (\omega_1, \text{public } bty, \alpha, \mathrm{PermL}(\text{Freeable}, \text{public } bty, \text{public}, \alpha))$, and (H) $\mathrm{UpdateArr}(\sigma_2,\ (l_1, i),\ n,\ \text{public } bty) = \sigma_3$.

Given (I) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}'_1::\mathcal{L}'_2::(\mathrm{p},[(l',0),(l'_1,i')])}_{\mathcal{D}'_1::\mathcal{D}'_2::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_3,\ \Delta'_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_2)$ and (A), by Lemma 5.2.87 we have (J) $d = wa$.

Given (I) and (J), by SMC^2 rule Public Array Write Public Value Public Index we have $(e_1, e_2) \nvdash \gamma$, $0 \leq i' \leq \alpha' - 1$, (K) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ i') \parallel C'_1)$, (L) $((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C'_1) \Downarrow^{\mathcal{L}'_2}_{\mathcal{D}'_2} ((\mathrm{p}, \gamma,\ \sigma'_2,\ \Delta'_2,\ \mathrm{acc},\ n') \parallel C'_2)$, (M) $\gamma(x) = (l',\ \text{public const } bty'*)$, (N) $\sigma'_2(l') = (\omega', \text{public const } bty'*, 1, \mathrm{PermL}(\text{Freeable}, \text{public const } bty'*, \text{public}, 1))$, (O) $\mathrm{DecodePtr}(\text{public const } bty'*, 1, \omega') = [1,\ [(l'_1, 0)],\ [1],\ 1]$, (P) $\sigma'_2(l'_1) = (\omega'_1, \text{public } bty', \alpha', \mathrm{PermL}(\text{Freeable}, \text{public } bty', \text{public}, \alpha'))$, and (Q) $\mathrm{UpdateArr}(\sigma'_2,\ (l'_1, i'),\ n',\ \text{public } bty') = \sigma'_3$.

Given (B) and (K), by the inductive hypothesis we have (R) $\sigma_1 = \sigma'_1$, (S) $\Delta_1 = \Delta'_1$, (T) $i = i'$, (U) $\mathcal{D}_1 = \mathcal{D}'_1$, (V) $\mathcal{L}_1 = \mathcal{L}'_1$, and (W) $C_1 = C'_1$.

Given (C), (L), (R), (S), and (W), by the inductive hypothesis we have (X) $\sigma_2 = \sigma'_2$, (Y) $\Delta_2 = \Delta'_2$, (Z) $n = n'$, (A1) $\mathcal{D}_2 = \mathcal{D}'_2$, (B1) $\mathcal{L}_2 = \mathcal{L}'_2$, and (C1) $C_2 = C'_2$.

Given (D) and (M), by Definition 5.3.3 we have (D1) $l = l'$ and (E1) $bty = bty'$.

Given (E), (N), (X), and (D1), by Definition 5.3.4 we have (F1) $\omega = \omega'$.

Given (F), (O), (E1), and (F1), by Lemma 5.3.26 we have (G1) $l_1 = l'_1$.

Given (G), (P), (X), and (G1), by Definition 5.3.4 we have (H1) $\omega_1 = \omega'_1$ and (I1) $\alpha = \alpha'$.

Given (H), (Q), (X), (G1), (T), (Z), and (E1), by Lemma 5.3.35 we have (J1) $\sigma_3 = \sigma'_3$.

Given (E) and (F) by Lemma 5.3.62 we have accessed location (K1) $(\mathrm{p}, [(l, 0)])$. Given (H), by Lemma 5.3.67 we have accessed location (L1) $(\mathrm{p}, [(l_1, i)])$. Given (K1) and (L1), by Lemmas 5.3.44 and 5.3.45 we have (M1) $(\mathrm{p}, [(l, 0), (l_1, i)])$.

Given (N) and (O) by Lemma 5.3.62 we have accessed location (N1) $(\mathrm{p}, [(l', 0)])$. Given (Q), by Lemma 5.3.67 we have accessed location (O1) $(\mathrm{p}, [(l'_1, i')])$. Given (N1) and (O1), by Lemmas 5.3.44 and 5.3.45 we have (P1) $(\mathrm{p}, [(l', 0), (l'_1, i')])$.

Given (V), (B1), (M1), (P1), (D1), (G1), and (T), by Lemma 5.3.47 we have (Q1) $\mathcal{L}_1 :: \mathcal{L}_2 :: (\mathrm{p}, [(l, 0), (l_1, i)]) = \mathcal{L}'_1 :: \mathcal{L}'_2 :: (\mathrm{p}, [(l', 0), (l'_1, i')])$.

Given (U), (A1), and $(\mathrm{p}, [wa])$, by Lemma 5.3.38 we have (R1) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wa]) = \mathcal{D}'_1 :: \mathcal{D}'_2 :: (\mathrm{p}, [wa])$.

Given (Y), (J1), (C1), (Q1), and (R1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wa2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wa2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$ by SMC^2 rule Private Array Write Private Value Public Index we have $(e_1) \nvdash \gamma$, $(e_2) \vdash \gamma$, $0 \leq i \leq \alpha - 1$,

(B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ i) \parallel C_1)$,

(C) $((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ e_2) \parallel C_1) \Downarrow^{\mathcal{L}_2}_{\mathcal{D}_2} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ n) \parallel C_2)$,

(D) $\gamma(x) = (l,\ \text{private const } bty*)$,

(E) $\sigma_2(l) = (\omega, \text{private const } bty*, 1, \mathrm{PermL}(\text{Freeable}, \text{private const } bty*, \text{private}, 1))$,

(F) $\mathrm{DecodePtr}(\text{private const } bty*, 1, \omega) = [1,\ [(l_1, 0)],\ [1], 1]$,

(G) $\sigma_2(l_1) = (\omega_1, \text{private } bty, \alpha, \mathrm{PermL}(\text{Freeable}, \text{private } bty, \text{private}, \alpha))$,

(H) $\mathrm{DynamicUpdate}(\Delta_2, \sigma_2, [(l_1, i)], \mathrm{acc}, \text{private } bty) = \Delta_3$, and

(I) $\mathrm{UpdateArr}(\sigma_2,\ (l_1, i),\ n,\ \text{private } bty) = \sigma_3$.

Given (J) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}'_1::\mathcal{L}'_2::(\mathrm{p},[(l',0),(l'_1,i')])}_{\mathcal{D}'_1::\mathcal{D}'_2::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_3,\ \Delta'_3,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_2)$ and (A),

by Lemma 5.2.87 we have (K) $d = wa2$.

Given (J) and (K), by SMC^2 rule Private Array Write Private Value Public Index we have $(e_1) \nvdash \gamma$, $(e_2) \vdash \gamma$, $0 \leq i' \leq \alpha' - 1$, (L) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e_1) \parallel C) \Downarrow^{\mathcal{L}_1'}_{\mathcal{D}_1'} ((\mathrm{p}, \gamma,\ \sigma_1',\ \Delta_1',\ \mathrm{acc},\ i') \parallel C_1')$, (M) $((\mathrm{p}, \gamma,\ \sigma_1',\ \Delta_1',\ \mathrm{acc},\ e_2) \parallel C_1') \Downarrow^{\mathcal{L}_2'}_{\mathcal{D}_2'} ((\mathrm{p}, \gamma,\ \sigma_2',\ \Delta_2',\ \mathrm{acc},\ n') \parallel C_2')$, (N) $\gamma(x) = (l',\ \text{private const } bty'*)$, (O) $\sigma_2'(l') = (\omega',\ \text{private const } bty'*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable},\ \text{private const } bty'*, \text{private}, 1))$, (P) $\mathrm{DecodePtr}(\text{private const } bty'*, 1, \omega') = [1,\ [(l_1', 0)], [1],\ 1]$, (Q) $\sigma_2'(l_1') = (\omega_1', \text{private } bty', \alpha',\ \mathrm{PermL}(\mathrm{Freeable}, \text{private } bty', \text{private}, \alpha'))$, (R) $\mathrm{DynamicUpdate}(\Delta_2',\ \sigma_2',\ [(l_1', i')],\ \mathrm{acc}, \text{private } bty') = \Delta_3'$, and (S) $\mathrm{UpdateArr}(\sigma_2',\ (l_1', i'),\ n',\ \text{private } bty') = \sigma_3'$.

Given (B) and (L), by the inductive hypothesis we have (T) $\sigma_1 = \sigma_1'$, (U) $\Delta_1 = \Delta_1'$, (V) $i = i'$, (W) $\mathcal{D}_1 = \mathcal{D}_1'$, (X) $\mathcal{L}_1 = \mathcal{L}_1'$, and (Y) $C_1 = C_1'$.

Given (C), (M), (T), (U), and (Y), by the inductive hypothesis we have (Z) $\sigma_2 = \sigma_2'$, (A1) $\Delta_2 = \Delta_2'$, (B1) $n = n'$, (C1) $\mathcal{D}_2 = \mathcal{D}_2'$, (D1) $\mathcal{L}_2 = \mathcal{L}_2'$, and (E1) $C_2 = C_2'$.

Given (D) and (N), by Definition 5.3.3 we have (F1) $l = l'$ and (G1) $bty = bty'$.

Given (E), (O), (Z), and (F1), by Definition 5.3.4 we have (H1) $\omega = \omega'$.

Given (F), (P), (G1), and (H1), by Lemma 5.3.26 we have (I1) $l_1 = l_1'$.

Given (G), (Q), (Z), and (I1), by Definition 5.3.4 we have (J1) $\omega_1 = \omega_1'$ and (K1) $\alpha = \alpha'$.

Given (H), (R), (A1), (Z), (I1), (V), and (G1), by Lemma 5.3.25 we have (L1) $\Delta_3 = \Delta_3'$.

Given (I), (S), (Z), (I1), (V), (B1), and (G1), by Lemma 5.3.35 we have (M1) $\sigma_3 = \sigma_3'$.

Given (E) and (F) by Lemma 5.3.62 we have accessed location (N1) $(\mathrm{p}, [(l, 0)])$. Given (I), by Lemma 5.3.67 we have accessed location (O1) $(\mathrm{p}, [(l_1, i)])$. Given (N1) and (O1), by Lemmas 5.3.44 and 5.3.45 we have (P1) $(\mathrm{p}, [(l, 0), (l_1, i)])$.

Given (O) and (P) by Lemma 5.3.62 we have accessed location (Q1) $(\mathrm{p}, [(l', 0)])$. Given (S), by Lemma 5.3.67 we have accessed location (R1) $(\mathrm{p}, [(l_1', i')])$. Given (Q1) and (R1), by Lemmas 5.3.44 and 5.3.45 we have (S1) $(\mathrm{p}, [(l', 0), (l_1', i')])$.

Given (X), (D1), (P1), (S1), (F1), (I1), and (V), by Lemma 5.3.47 we have (T1) $\mathcal{L}_1 :: \mathcal{L}_2 :: (\mathrm{p}, [(l, 0), (l_1, i)]) = \mathcal{L}'_1 :: \mathcal{L}'_2 :: (\mathrm{p}, [(l', 0), (l'_1, i')])$.

Given (Y), (E1), and (K), by Lemma 5.3.38 we have (U1) $\mathcal{D}_1 :: \mathcal{D}_2 :: (\mathrm{p}, [wa2]) = \mathcal{D}'_1 :: \mathcal{D}'_2 :: (\mathrm{p}, [wa2])$.

Given (E1), (L1), (M1), (T1), and (U1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wa1])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_2)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x[e_1]\ = e_2) \parallel C) \Downarrow^{\mathcal{L}_1::\mathcal{L}_2::(\mathrm{p},[(l,0),(l_1,i)])}_{\mathcal{D}_1::\mathcal{D}_2::(\mathrm{p},[wa2])} ((\mathrm{p}, \gamma,\ \sigma_3,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C_2)$. We use Axiom 5.3.1 to prove that $\mathrm{encrypt}(n) = \mathrm{encrypt}(n')$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[dp])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[dp])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ by SMC^2 rule Public Pointer Declaration we have $(ty = \mathrm{public}\ bty*) \vee ((ty = bty*) \wedge ((bty = \mathrm{char}) \vee (bty = \mathrm{void})))$, $\mathrm{acc} = 0$, (B) $l = \phi()$, (C) $\mathrm{GetIndirection}(*) = i$, (D) $\omega = \mathrm{EncodePtr}(\mathrm{public}\ bty*, [1, [(l_{default}, 0)], [1], i])$, (E) $\gamma_1 = \gamma[x \rightarrow (l,\ \mathrm{public}\ bty*)]$, and (F) $\sigma_1 = \sigma[l \rightarrow (\omega,\ \mathrm{public}\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{public}\ bty*, \mathrm{public}, 1))]$.

Given (G) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l',0)])}_{(\mathrm{p},[d])} ((\mathrm{p}, \gamma'_1,\ \sigma'_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$ and (A), by Lemma 5.2.87 we have (H) $d = dp$.

Given (G) and (H), by SMC^2 rule Public Pointer Declaration we have $(ty = \mathrm{public}\ bty*) \vee ((ty = bty*) \wedge ((bty = \mathrm{char}) \vee (bty = \mathrm{void})))$, $\mathrm{acc} = 0$, (I) $l' = \phi()$, (J) $\mathrm{GetIndirection}(*) = i'$, (K) $\omega' = \mathrm{EncodePtr}(\mathrm{public}\ bty*, [1, [(l_{default}, 0)], [1], i'])$, (L) $\gamma'_1 = \gamma[x \rightarrow (l',\ \mathrm{public}\ bty*)]$, and (M) $\sigma'_1 = \sigma[l' \rightarrow (\omega',\ \mathrm{public}\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{public}\ bty*, \mathrm{public}, 1))]$.

Given (B) and (I), by Axiom 5.3.4 we have (N) $l = l'$.

Given (C) and (J), by Lemma 5.3.13 we have (O) $i = i'$.

Given (D), (K), and (O), by Lemma 5.3.32 we have (P) $\omega = \omega'$.

Given (E), (L), and (N), by Definition 5.3.3 we have (Q) $\gamma_1 = \gamma_1'$.

Given (F), (M), (N), and (P), by Definition 5.3.4 we have (R) $\sigma_1 = \sigma_1'$.

Given (A), (G), and (H), we have (S) $(\mathrm{p}, [dp]) = (\mathrm{p}, [dp])$.

Given (F), by Lemma 5.3.51 we have accessed (T) $(\mathrm{p}, [(l, 0)])$. Given (M), by Lemma 5.3.51 we have accessed (U) $(\mathrm{p}, [(l', 0)])$. Given (T), (U), and (N), we have (V) $(\mathrm{p}, [(l, 0)]) = (\mathrm{p}, [(l', 0)])$.

Given (Q), (R), (S), and (V), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[dp1])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ ty\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[dp])} ((\mathrm{p}, \gamma_1,\ \sigma_1,\ \Delta,\ \mathrm{acc},\ \mathrm{skip}) \parallel C)$

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[rp])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_1, \mu_1)) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[rp])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_1, \mu_1)) \parallel C)$ by SMC^2 rule Pointer Read Single Location we have (B) $\gamma(x) = (l,\ a\ bty*)$, (C) $\sigma(l) = (\omega,\ a\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, a\ bty*, a, 1))$, and (D) $\mathrm{DecodePtr}(a\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ i]$.

Given (E) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l',0)])}_{(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_1', \mu_1')) \parallel C)$ and (A), by Lemma 5.2.87 we have (F) $d = rp$.

Given (E) and (F), by SMC^2 rule Pointer Read Single Location we have (G) $\gamma(x) = (l',\ a'\ bty'*)$, (H) $\sigma(l') = (\omega',\ a'\ bty'*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, a'\ bty'*, a', 1))$, and (I) $\mathrm{DecodePtr}(a'\ bty'*,\ 1,\ \omega') = [1,\ [(l_1', \mu_1')],\ [1],\ i']$.

Given (B) and (G), by Definition 5.3.3 we have (J) $l = l'$, (K) $a =$, and (L) $bty = bty'$.

Given (C), (H), and (J), by Definition 5.3.4 we have (M) $\omega = \omega'$.

Given (D), (I), (K), (L), and (M), by Lemma 5.3.26 we have $[1,\ [(l_1, \mu_1)],\ [1],\ i] = [1,\ [(l'_1, \mu'_1)],\ [1],\ i']$ and therefore (N) $(l_1, \mu_1) = (l'_1, \mu'_1)$.

Given (C) and (D), by Lemma 5.3.62 we have accessed location (O) $(\mathrm{p}, [(l, 0)])$.

Given (H) and (I), by Lemma 5.3.62 we have accessed location (P) $(\mathrm{p}, [(l', 0)])$.

Given (O), (P), and (J), we have (Q) $(\mathrm{p}, [(l, 0)]) = (\mathrm{p}, [(l', 0)])$.

Given (F), (N), and (Q), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[rp1])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_1, \mu_1)) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0)])}_{(\mathrm{p},[rp])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_1, \mu_1)) \parallel C)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[wp1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[wp1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Private Pointer Write we have $(e) \nvdash \gamma$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ (l_e, \mu_e)) \parallel C_1)$, (C) $\gamma(x) = (l,\ \mathrm{private}\ bty*)$, (D) $\sigma_1(l) = (\omega,\ \mathrm{private}\ bty*,\ \alpha, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty*, \mathrm{private}, \alpha))$, (E) $\mathrm{DecodePtr}(\mathrm{private}\ bty*,\ \alpha,\ \omega) = [\alpha,\ \bar{l},\ \bar{j},\ i]$, and (F) $\mathrm{UpdatePtr}(\sigma_1,\ (l, 0),\ [1,\ [(l_e, \mu_e)],\ [1],\ i], \mathrm{private}\ bty*) = (\sigma_2,\ 1)$.

Given (G) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}'_1::(\mathrm{p},[(l',0)])}_{\mathcal{D}'_1::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_2,\ \Delta'_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_1)$ and (A), by Lemma 5.2.87 we have (H) $d = wp1$.

Given (G) and (H), by SMC^2 rule Private Pointer Write we have $(e) \nvdash \gamma$,
(I) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ (l'_e, \mu'_e)) \parallel C'_1)$,

(J) $\gamma(x) = (l',\ \text{private}\ bty'*)$,

(K) $\sigma_1'(l') = (\omega',\ \text{private}\ bty'*,\ \alpha', \text{PermL}(\text{Freeable}, \text{private}\ bty'*, \text{private}, \alpha'))$,

(L) $\text{DecodePtr}(\text{private}\ bty'*,\ \alpha',\ \omega') = [\alpha',\ \bar{l}',\ \bar{j}',\ i']$, and

(M) $\text{UpdatePtr}(\sigma_1',\ (l', 0),\ [1,\ [(l_e', \mu_e')],\ [1],\ i'], \text{private}\ bty'*) = (\sigma_2',\ 1)$.

Given (B) and (I), by the inductive hypothesis we have (N) $\sigma_1 = \sigma_1'$, (O) $\Delta_1 = \Delta_1'$, (P) $(l_e, \mu_e) = (l_e', \mu_e')$, (Q) $\mathcal{D}_1 = \mathcal{D}_1'$, (R) $\mathcal{L}_1 = \mathcal{L}_1'$, and (S) $C_1 = C_1'$.

Given (C) and (J), by Definition 5.3.3 we have (T) $l = l'$ and (U) $bty = bty'$.

Given (D), (K), (N), and (T), by Definition 5.3.4 we have (V) $\omega = \omega'$ and (W) $\alpha = \alpha'$.

Given (E), (L), (U), (W), and (V), by Lemma 5.3.26 we have $\bar{l} = \bar{l}'$, $\bar{j} = \bar{j}'$, and (X) $i = i'$.

Given (F), (M), (N), (T), (P), (X), and (V), by Lemma 5.3.36 we have (Y) $\sigma_2 = \sigma_2'$.

Given (Q) and (H), by Lemma 5.3.38 we have (Z) $\mathcal{D}_1 :: (\text{p}, [wp1]) = \mathcal{D}_1' :: (\text{p}, [wp1])$.

Given (D) and (E), by Lemma 5.3.62 we have accessed location (A1) $(\text{p}, [(l, 0)])$. Given (F), by Lemma 5.3.68 we have accessed location (B1) $(\text{p}, [(l, 0)])$.

Given (K) and (L), by Lemma 5.3.62 we have accessed location (C1) $(\text{p}, [(l', 0)])$. Given (M), by Lemma 5.3.68 we have accessed location (D1) $(\text{p}, [(l', 0)])$

Given (R), (A1), (B1), (C1), (D1), and (T), by Lemma 5.3.47 we have (E1) $\mathcal{L}_1 :: (\text{p}, [(l, 0)]) = \mathcal{L}_1' :: (\text{p}, [(l', 0)])$.

Given (Y), (O), (S), (Z), and (E1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0)])}_{\mathcal{D}_1 :: (\text{p}, [wp])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\text{p}, \gamma,\ \sigma,\ \Delta,\ \text{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\text{p}, [(l,0)])}_{\mathcal{D}_1 :: (\text{p}, [wp1])} ((\text{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \text{acc},\ \text{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[wp2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)])}_{\mathcal{D}_1::(\mathrm{p},[wp1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp3])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp3])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Private Pointer Dereference Write Single Location Private Value we have $(e) \vdash \gamma$, $(bty = \mathrm{int}) \vee (bty = \mathrm{float})$, (B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc},\ n) \parallel C_1)$, (C) $\gamma(x) = (l,\ \mathrm{private}\ bty*)$, (D) $\sigma_1(l) = (\omega,\ \mathrm{private}\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty*, \mathrm{private}, 1))$, (E) $\mathrm{DecodePtr}(\mathrm{private}\ bty*,\ 1,\ \omega) = [1, [(l_1, \mu_1)], [1],\ 1]$, (F) $\mathrm{DynamicUpdate}(\Delta_1, \sigma_1, [(l_1, \mu_1)],\ \mathrm{acc}, \mathrm{private}\ bty) = (\Delta_2, \bar{l}_1)$, and (G) $\mathrm{UpdateOffset}(\sigma_1, (l_1, \mu_1), n, \mathrm{private}\ bty) = (\sigma_2, 1)$.

Given (H) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}'_1::(\mathrm{p},[(l',0)]::\bar{l}'_1::[(l'_1,\mu'_1)])}_{\mathcal{D}'_1::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_2,\ \Delta'_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_1)$ and (A), by Lemma 5.2.87 we have (I) $d = wdp3$.

Given (H) and (I), by SMC^2 rule Private Pointer Dereference Write Single Location Private Value we have $(e) \vdash \gamma$, $(bty' = \mathrm{int}) \vee (bty' = \mathrm{float})$, (J) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc},\ n') \parallel C'_1)$, (K) $\gamma(x) = (l',\ \mathrm{private}\ bty'*)$, (L) $\sigma'_1(l') = (\omega',\ \mathrm{private}\ bty'*,\ 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty'*, \mathrm{private}, 1))$, (M) $\mathrm{DecodePtr}(\mathrm{private}\ bty'*,\ 1,\ \omega') = [1,\ [(l'_1, \mu'_1)],\ [1],\ 1]$, (N) $\mathrm{DynamicUpdate}(\Delta'_1, \sigma'_1, [(l'_1, \mu'_1)],\ \mathrm{acc}, \mathrm{private}\ bty') = (\Delta'_2, \bar{l}'_1)$, and (O) $\mathrm{UpdateOffset}(\sigma'_1,\ (l'_1, \mu'_1),\ n', \mathrm{private}\ bty') = (\sigma'_2, 1)$.

Given (B) and (J), by the inductive hypothesis we have (P) $\sigma_1 = \sigma'_1$, (Q) $\Delta_1 = \Delta'_1$, (R) $n = n'$, (S) $\mathcal{D}_1 = \mathcal{D}'_1$, (T) $\mathcal{L}_1 = \mathcal{L}'_1$, and (U) $C_1 = C'_1$.

Given (C) and (K), by Definition 5.3.3 we have (V) $l = l'$ and (W) $bty = bty'$.

Given (D), (L), (P), and (V), by Definition 5.3.4 we have (X) $\omega = \omega'$.

Given (E), (M), (W), and (X), by Lemma 5.3.26 we have (Y) $(l_1, \mu_1) = (l'_1, \mu'_1)$.

Given (F), (N), (Q), (P), (Y), and (W), by Lemma 5.3.25 we have (Z) $\Delta_2 = \Delta_2'$ and (A1) $\bar{l}_1 = \bar{l}_1'$.

Given (G), (O), (P), (Y), (R), and (W), by Lemma 5.3.37 we have (B1) $\sigma_2 = \sigma_2'$.

Given (S) and $(\mathrm{p}, [wdp3])$, by Lemma 5.3.38 we have (C1) $\mathcal{D}_1 :: (\mathrm{p}, [wdp3]) = \mathcal{D}_1' :: (\mathrm{p}, [wdp3])$.

Given (D) and (E), by Lemma 5.3.62 we have accessed location (D1) $(\mathrm{p}, [(l, 0)])$. Given (F), by Lemma 5.3.61 we have accessed location (E1) $(\mathrm{p}, \bar{l}_1)$. Given (G), by Lemma 5.3.69 we have accessed location (F1) $(\mathrm{p}, [(l_1, \mu_1)])$. Given (D1), (E1), and (F1), by Lemmas 5.3.44 and 5.3.45 we have (G1) $(\mathrm{p}, [(l, 0)] :: \bar{l}_1 :: [(l_1, \mu_1)])$.

Given (L) and (M), by Lemma 5.3.62 we have accessed location (H1) $(\mathrm{p}, [(l', 0)])$. Given (N), by Lemma 5.3.61 we have accessed location (I1) $(\mathrm{p}, \bar{l}_1')$. Given (O), by Lemma 5.3.69 we have accessed location (J1) $(\mathrm{p}, [(l_1', \mu_1')])$. Given (H1), (I1), and (J1), by Lemmas 5.3.44 and 5.3.45 we have (K1) $(\mathrm{p}, [(l', 0)] :: \bar{l}_1' :: [(l_1', \mu_1')])$

Given (T), (G1), (K1), (A1), (V), and (Y), by Lemma 5.3.47 we have (L1) $\mathcal{L}_1 :: (\mathrm{p}, [(l, 0)] :: \bar{l}_1 :: [(l_1, \mu_1)) = \mathcal{L}_1' :: (\mathrm{p}, [(l', 0)] :: \bar{l}_1' :: [(l_1', \mu_1')] :: \bar{l}_1')$.

Given (A1), (Z), (U), (B1), and (L1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)] :: \bar{l}_1 :: [(l_1, \mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp4])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)] :: \bar{l}_1 :: [(l_1, \mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp3])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1)$, with the addition of using Axiom 5.3.1 to prove that $\mathrm{encrypt}(n) = \mathrm{encrypt}(n')$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0), (l_1, \mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)] :: \bar{l}_1 :: [(l_1, \mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp3])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1)$, removing the reasoning about $\mathrm{DynamicUpdate}$ and its resulting locations, as it is not present in this rule.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)] :: \bar{l}_1 :: [(l_1, \mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1::(\mathrm{p},[(l,0)]::\bar{l}_1::[(l_1,\mu_1)])}_{\mathcal{D}_1::(\mathrm{p},[wdp2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$ by SMC^2 rule Private Pointer Dereference Write Multiple Locations to Single Location Higher Level Indirection we have $(e) \vdash \gamma$, $(bty = \mathrm{int}) \vee (bty = \mathrm{float})$,

(B) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}_1}_{\mathcal{D}_1} ((\mathrm{p}, \gamma,\ \sigma_1,\ \Delta_1,\ \mathrm{acc}, [\alpha, \bar{l}_e, \bar{j}_e, i-1]) \parallel C_1)$,

(C) $\gamma(x) = (l,\ \mathrm{private}\ bty*)$,

(D) $\sigma_1(l) = (\omega,\ \mathrm{private}\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty*, \mathrm{private}, 1))$,

(E) $\mathrm{DecodePtr}(\mathrm{private}\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1]$,

(F) $\mathrm{DynamicUpdate}(\Delta_1, \sigma_1, [(l_1, \mu_1)],\ \mathrm{acc}, \mathrm{private}\ bty*) = (\Delta_2, \bar{l}_1)$, and

(G) $\mathrm{UpdatePtr}(\sigma_1,\ (l_1, \mu_1),\ [\alpha, \bar{l}_e, \bar{j}_e, i-1], \mathrm{private}\ bty*) = (\sigma_2, 1)$.

Given (H) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}'_1::(\mathrm{p},[(l',0)]::\bar{l}'_1::[(l'_1,\mu'_1)])}_{\mathcal{D}'_1::(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma'_2,\ \Delta'_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C'_1)$ and (A), by Lemma 5.2.87 we have (I) $d = wdp2$.

Given (H) and (I), by SMC^2 rule Private Pointer Dereference Write Multiple Locations to Single Location Higher Level Indirection we have $(e) \vdash \gamma$, $(bty' = \mathrm{int}) \vee (bty' = \mathrm{float})$,

(J) $((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ e) \parallel C) \Downarrow^{\mathcal{L}'_1}_{\mathcal{D}'_1} ((\mathrm{p}, \gamma,\ \sigma'_1,\ \Delta'_1,\ \mathrm{acc}, [\alpha', \bar{l}'_e, \bar{j}'_e, i'-1]) \parallel C'_1)$,

(K) $\gamma(x) = (l',\ \mathrm{private}\ bty'*)$,

(L) $\sigma'_1(l') = (\omega',\ \mathrm{private}\ bty'*,\ 1, \mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty'*, \mathrm{private}, 1))$,

(M) $\mathrm{DecodePtr}(\mathrm{private}\ bty'*,\ 1,\ \omega') = [1,\ [(l'_1, \mu'_1)],\ [1],\ 1]$,

(N) $\mathrm{DynamicUpdate}(\Delta'_1, \sigma'_1, [(l'_1, \mu'_1)],\ \mathrm{acc}, \mathrm{private}\ bty'*) = (\Delta'_2, \bar{l}'_1)$, and

(O) $\mathrm{UpdatePtr}(\sigma'_1,\ (l'_1, \mu'_1),\ [\alpha', \bar{l}'_e, \bar{j}'_e, i'-1], \mathrm{private}\ bty'*) = (\sigma'_2, 1)$.

Given (B) and (J), by the inductive hypothesis we have (P) $\sigma_1 = \sigma'_1$, (Q) $\Delta_1 = \Delta'_1$, (R) $[\alpha, \bar{l}_e, \bar{j}_e, i-1] = [\alpha', \bar{l}'_e, \bar{j}'_e, i'-1]$, (S) $\mathcal{D}_1 = \mathcal{D}'_1$, (T) $\mathcal{L}_1 = \mathcal{L}'_1$, and (U) $C_1 = C'_1$.

Given (C) and (K), by Definition 5.3.3 we have (V) $l = l'$ and (W) $bty = bty'$.

Given (D), (L), (P), and (V), by Definition 5.3.4 we have (X) $\omega = \omega'$.

Given (E), (M), (W), and (X), by Lemma 5.3.26 we have (Y) $(l_1, \mu_1) = (l'_1, \mu'_1)$.

Given (F), (N), (Q), (P), (Y), and (W), by Lemma 5.3.25 we have (Z) $\Delta_2 = \Delta'_2$ and (A1) $\bar{l}_1 = \bar{l}'_1$.

Given (G), (O), (P), (Y), (R), and (W), by Lemma 5.3.37 we have (B1) $\sigma_2 = \sigma_2'$.

Given (S) and $(\mathrm{p}, [wdp2])$, by Lemma 5.3.38 we have (C1) $\mathcal{D}_1 :: (\mathrm{p}, [wdp2]) = \mathcal{D}_1' :: (\mathrm{p}, [wdp2])$.

Given (D) and (E), by Lemma 5.3.62 we have accessed location (D1) $(\mathrm{p}, [(l, 0)])$. Given (F), by Lemma 5.3.61 we have accessed location (E1) $(\mathrm{p}, \bar{l}_1)$. Given (G), by Lemma 5.3.68 we have accessed location (F1) $(\mathrm{p}, [(l_1, \mu_1)])$. Given (D1), (E1), and (F1), by Lemmas 5.3.44 and 5.3.45 we have (G1) $(\mathrm{p}, [(l, 0)] :: \bar{l}_1 :: [(l_1, \mu_1)])$.

Given (L) and (M), by Lemma 5.3.62 we have accessed location (H1) $(\mathrm{p}, [(l', 0)])$. Given (N), by Lemma 5.3.61 we have accessed location (I1) $(\mathrm{p}, \bar{l}_1')$. Given (O), by Lemma 5.3.68 we have accessed location (J1) $(\mathrm{p}, [(l_1', \mu_1')])$. Given (H1), (I1), and (J1), by Lemmas 5.3.44 and 5.3.45 we have (K1) $(\mathrm{p}, [(l', 0)] :: \bar{l}_1' :: [(l_1', \mu_1')])$

Given (T), (G1), (K1), (A1), (V), and (Y), by Lemma 5.3.47 we have (L1) $\mathcal{L}_1 :: (\mathrm{p}, [(l, 0)] :: \bar{l}_1 :: [(l_1, \mu_1)) = \mathcal{L}_1' :: (\mathrm{p}, [(l', 0)] :: \bar{l}_1' :: [(l_1', \mu_1')] :: \bar{l}_1')$.

Given (A1), (Z), (U), (B1), and (L1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0),(l_1,\mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp1])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_1,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)] :: \bar{l}_1 :: [(l_1,\mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1)$, removing the reasoning about $\mathrm{DynamicUpdate}$ and its resulting locations, as it is not present in this rule.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)] :: \bar{l}_1 :: [(l_1,\mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp5])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc},\ \mathrm{skip}) \parallel C_1)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x = e) \parallel C) \Downarrow^{\mathcal{L}_1 :: (\mathrm{p}, [(l,0)] :: \bar{l}_1 :: [(l_1,\mu_1)])}_{\mathcal{D}_1 :: (\mathrm{p}, [wdp2])} ((\mathrm{p}, \gamma,\ \sigma_2,\ \Delta_2,\ \mathrm{acc}, \mathrm{skip}) \parallel C_1)$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p}, [(l,0),(l_1,\mu_1)])}_{(\mathrm{p}, [rdp])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$

Given (A) $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$ by SMC^2 rule Pointer Dereference Single Location, we have (B) $\gamma(x) = (l,\ a\ bty*)$, (C) $\sigma(l) = (\omega,\ a\ bty*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, a\ bty*, a, 1))$, (D) $\mathrm{DecodePtr}(a\ bty*,\ 1,\ \omega) = [1,\ [(l_1, \mu_1)],\ [1],\ 1]$, and (E) $\mathrm{DerefPtr}(\sigma, a\ bty, (l_1, \mu_1)) = (n, 1)$.

Given (F) $\Sigma \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p},[(l',0),(l'_1,\mu'_1)])}_{(\mathrm{p},[d])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n') \parallel C)$ and (A), by Lemma 5.2.87 we have (G) $d = rdp$.

Given (F) and (G), by SMC^2 rule Pointer Dereference Single Location, we have (H) $\gamma(x) = (l',\ a'\ bty'*)$, (I) $\sigma(l') = (\omega',\ a'\ bty'*,\ 1,\ \mathrm{PermL}(\mathrm{Freeable}, a'\ bty'*, a', 1))$, (J) $\mathrm{DecodePtr}(a'\ bty'*,\ 1,\ \omega') = [1,\ [(l'_1, \mu'_1)], [1], 1]$, and (K) $\mathrm{DerefPtr}(\sigma, a'\ bty', (l'_1, \mu'_1)) = (n', 1)$.

Given (B) and (H), by Definition 5.3.3 we have (L) $l = l'$ and (M) $a\ bty = a'\ bty'$.

Given (C), (I), and (L), by Definition 5.3.4 we have (N) $\omega = \omega'$ and (O) $a = a'$.

Given (D), (J), (M), and (N), by Lemma 5.3.26 we have (P) $(l_1, \mu_1) = (l'_1, \mu'_1)$.

Given (E), (K), (M), and (P), by Lemma 5.3.14 we have (Q) $n = n'$.

Given (C) and (D), by Lemma 5.3.62 we have accessed location (R) $(\mathrm{p}, [(l, 0)])$. Given (E), by Lemma 5.3.70 we have accessed location (S) $(\mathrm{p}, [(l_1, \mu_1)])$. Given (R) and (S), by Lemmas 5.3.44 and 5.3.45 we have (T) $(\mathrm{p}, [(l, 0), (l_1, \mu_1)])$.

Given (I) and (J), by Lemma 5.3.62 we have accessed location (U) $(\mathrm{p}, [(l', 0)])$. Given (K), by Lemma 5.3.70 we have accessed location (V) $(\mathrm{p}, [(l'_1, \mu'_1)])$. Given (U) and (V), by Lemmas 5.3.44 and 5.3.45 we have (W) $(\mathrm{p}, [(l, 0), (l_1, \mu_1)])$.

Given (T), (W), (L), and (P), we have (X) $(\mathrm{p}, [(l, 0), (l_1, \mu_1)]) = (\mathrm{p}, [(l', 0), (l'_1, \mu'_1)])$.

Given (Q) and (X), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp1])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$

This case is similar to Case $\Pi \triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ n) \parallel C)$. The

difference is in the use of Lemma 5.3.15 in place of Lemma 5.3.14 to reason about the use of DerefPtrHLI and that the pointer data structure being returned is equivalent. We use Lemma 5.3.71 in place of Lemma 5.3.70 to reason about the locations accessed within DerefPtrHLI.

Case $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp2])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ [\alpha,\ \bar{l},\ \bar{j},\ i-1]) \parallel C)$

This case is similar to Case $\Pi\triangleright ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ *x) \parallel C) \Downarrow^{(\mathrm{p},[(l,0),(l_1,\mu_1)])}_{(\mathrm{p},[rdp1])} ((\mathrm{p}, \gamma,\ \sigma,\ \Delta,\ \mathrm{acc},\ (l_2, \mu_2)) \parallel C)$.

Case $\Pi\triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x) \parallel ... \parallel\ (\mathrm{q}, \gamma^\mathrm{q}, \sigma^\mathrm{q}, \Delta^\mathrm{q}, \mathrm{acc}, *x)) \Downarrow^{(1,(l^1,0)::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^\mathrm{q},0)::\bar{l}^\mathrm{q})}_{(\mathrm{ALL},[mprdp])} ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc},$ $n^1) \parallel ... \parallel\ (\mathrm{q}, \gamma^\mathrm{q}, \sigma^\mathrm{q}, \Delta^\mathrm{q}, \mathrm{acc}, n^\mathrm{q}))$

Given (A) $\Pi\triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x) \parallel ... \parallel\ (\mathrm{q}, \gamma^\mathrm{q}, \sigma^\mathrm{q}, \Delta^\mathrm{q}, \mathrm{acc}, *x)) \Downarrow^{(1,(l^1,0)::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^\mathrm{q},0)::\bar{l}^\mathrm{q})}_{(\mathrm{ALL},[mprdp])} ((1, \gamma^1, \sigma^1, \Delta^1,$ $\mathrm{acc},\ n^1) \parallel ... \parallel\ (\mathrm{q}, \gamma^\mathrm{q}, \sigma^\mathrm{q}, \Delta^\mathrm{q}, \mathrm{acc}, n^\mathrm{q}))$ by SMC^2 rule Multiparty Private Pointer Dereference Single Level Indirection, we have (B) $\{(x) \vdash \gamma^\mathrm{p}\}^\mathrm{q}_{\mathrm{p}=1}$, (C) $\{\gamma^\mathrm{p}(x) = (l^\mathrm{p}, \mathrm{private}\ bty*)\}^\mathrm{q}_{\mathrm{p}=1}$, (D) $\{\sigma^\mathrm{p}(l^\mathrm{p}) = (\omega^\mathrm{p},\ \mathrm{private}\ bty*,\ \alpha,$ $\mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty*, \mathrm{private}, \alpha))\}^\mathrm{q}_{\mathrm{p}=1}$, (E) $\alpha > 1$, (F) $\{\mathrm{DecodePtr}(\mathrm{private}\ bty*,\ \alpha,\ \omega^\mathrm{p}) = [\alpha, \bar{l}^\mathrm{p}, \bar{j}^\mathrm{p},$ $1]\}^\mathrm{q}_{\mathrm{p}=1}$, (G) $\{\mathrm{Retrieve_vals}(\alpha, \bar{l}^\mathrm{p}, \mathrm{private}\ bty, \sigma^\mathrm{p}) = ([n^\mathrm{p}_0, ...n^\mathrm{p}_{\alpha-1}], 1)\}^\mathrm{q}_{\mathrm{p}=1}$, and (H) $\mathrm{MPC}_{dv}([[n^1_0,\ ...,\ n^1_{\alpha-1}],\ ...,$ $[n^\mathrm{q}_0, ..., n^\mathrm{q}_{\alpha-1}]], [\bar{j}^1, ..., \bar{j}^\mathrm{q}]) = (n^1, ..., n^\mathrm{q})$.

Given (I) $\Sigma\triangleright((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x) \parallel ... \parallel\ (\mathrm{q}, \gamma^\mathrm{q}, \sigma^\mathrm{q}, \Delta^\mathrm{q}, \mathrm{acc}, *x)) \Downarrow^{(1,(l'^1,0)::\bar{l}'^1) \parallel ... \parallel (\mathrm{q},(l'^\mathrm{q},0)::\bar{l}'^\mathrm{q})}_{(\mathrm{ALL},[d])} ((1, \gamma^1, \sigma^1, \Delta^1,$ $\mathrm{acc},\ n'^1) \parallel ... \parallel\ (\mathrm{q}, \gamma^\mathrm{q}, \sigma^\mathrm{q}, \Delta^\mathrm{q}, \mathrm{acc}, n'^\mathrm{q}))$ and (A), by Lemma 5.2.87 we have (J) $d = mprdp$.

Given (I) and (J), by SMC^2 rule Multiparty Private Pointer Dereference Single Level Indirection, we have (K) $\{(x) \vdash \gamma^\mathrm{p}\}^\mathrm{q}_{\mathrm{p}=1}$, (L) $\{\gamma^\mathrm{p}(x) = (l'^\mathrm{p}, \mathrm{private}\ bty'*)\}^\mathrm{q}_{\mathrm{p}=1}$, (M) $\{\sigma^\mathrm{p}(l'^\mathrm{p}) = (\omega'^\mathrm{p},\ \mathrm{private}\ bty'*,\ \alpha',$ $\mathrm{PermL}(\mathrm{Freeable}, \mathrm{private}\ bty'*, \mathrm{private}, \alpha'))\}^\mathrm{q}_{\mathrm{p}=1}$, (N) $\alpha' > 1$, (O) $\{\mathrm{DecodePtr}(\mathrm{private}\ bty'*,\ \alpha',\ \omega'^\mathrm{p}) = [\alpha',$ $\bar{l}'^\mathrm{p}, \bar{j}'^\mathrm{p}, 1]\}^\mathrm{q}_{\mathrm{p}=1}$, (P) $\{\mathrm{Retrieve_vals}(\alpha', \bar{l}'^\mathrm{p}, \mathrm{private}\ bty', \sigma^\mathrm{p}) = ([n'^\mathrm{p}_0, ...n'^\mathrm{p}_{\alpha'-1}], 1)\}^\mathrm{q}_{\mathrm{p}=1}$, and (Q) $\mathrm{MPC}_{dv}([[n'^1_0,\ ...,$ $n'^1_{\alpha'-1}],\ ...,\ [n'^\mathrm{q}_0, ..., n'^\mathrm{q}_{\alpha'-1}]], [\bar{j}'^1, ..., \bar{j}'^\mathrm{q}]) = (n'^1, ..., n'^\mathrm{q})$.

Given (C) and (L), by Definition 5.3.3 we have (R) $\{l^\mathrm{p} = l'^\mathrm{p}\}^\mathrm{q}_{\mathrm{p}=1}$, and (S) $bty = bty'$.

Given (D), (M), and (R), by Definition 5.3.4 we have (T) $\{\omega^\mathrm{p} = \omega'^\mathrm{p}\}^\mathrm{q}_{\mathrm{p}=1}$ and (U) $\alpha = \alpha'$.

Given (F), (O), (S), (U), and (T), by Lemma 5.3.26 we have (V) $\{\bar{l}^{\mathrm{p}} = \bar{l}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (W) $\{\bar{j}^{\mathrm{p}} = \bar{j}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (G), (P), (U), (V), and (S), by Lemma 5.3.39 we have (X) $\{[n_0^{\mathrm{p}}, ...n_{\alpha-1}^{\mathrm{p}}] = [n_0'^{\mathrm{p}}, ...n_{\alpha'-1}'^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (H), (Q), (X), and (W), by Axiom 5.3.11 we have (Y) $\{n^{\mathrm{p}} = n'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (D) and (F), by Lemma 5.3.62 we have accessed location (Z) $\{(\mathrm{p}, [(l^{\mathrm{p}}, 0)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (G), by Lemma 5.3.72 we have accessed locations (A1) $\{(\mathrm{p}, \bar{l}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (Z) and (A1), by Lemmas 5.3.44 and 5.3.45 we have (B1) $\{(\mathrm{p}, (l^{\mathrm{p}}, 0) :: \bar{l}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (M) and (O), by Lemma 5.3.62 we have accessed location (C1) $\{(\mathrm{p}, [(l'^{\mathrm{p}}, 0)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (P), by Lemma 5.3.72 we have accessed locations (D1) $\{(\mathrm{p}, \bar{l}'^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (C1) and (D1), by Lemmas 5.3.44 and 5.3.45 we have (E1) $\{(\mathrm{p}, (l'^{\mathrm{p}}, 0) :: \bar{l}'^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (B1), (E1), (R), and (V), we have (F1) $(1, (l^1, 0) :: \bar{l}^1) \parallel ... \parallel (\mathrm{q}, (l^{\mathrm{q}}, 0) :: \bar{l}^{\mathrm{q}}) = (1, (l'^1, 0) :: \bar{l}'^1) \parallel ... \parallel (\mathrm{q}, (l'^{\mathrm{q}}, 0) :: \bar{l}'^{\mathrm{q}})$.

Given (Y) and (F1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, *x)) \Downarrow^{(1,(l^1,0)::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^{\mathrm{q}},0)::\bar{l}^{\mathrm{q}})}_{(\mathrm{ALL},[mprdp1])} ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, [\alpha_\alpha, \bar{l}_\alpha^1, \bar{j}_\alpha^1, i-1]) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, [\alpha_\alpha, \bar{l}_\alpha^{\mathrm{q}}, \bar{j}_\alpha^{\mathrm{q}}, i-1]))$

This case is similar to Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, *x)) \Downarrow^{(1,(l^1,0)::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^{\mathrm{q}},0)::\bar{l}^{\mathrm{q}})}_{(\mathrm{ALL},[mprdp])} ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, n^1) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}}))$. We use Axiom 5.3.12 to reason about the behavior of MPC_{dp}.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x = e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, *x = e)) \Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}_1^1::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^{\mathrm{q}},0)::\bar{l}_1^{\mathrm{q}}::\bar{l}^{\mathrm{q}})}_{\mathcal{D}_1::(\mathrm{ALL},[mpwdp3])} ((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \mathrm{acc}, *x = e) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, *x = e)) \Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}_1^1::\bar{l}^1) \parallel ... \parallel (\mathrm{q},(l^{\mathrm{q}},0)::\bar{l}_1^{\mathrm{q}}::\bar{l}^{\mathrm{q}})}_{\mathcal{D}_1::(\mathrm{ALL},[mpwdp3])} ((1, \gamma^1, \sigma_2^1, \Delta_2^1, \mathrm{acc}, \mathrm{skip}) \parallel ... \parallel (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip}))$ by SMC^2

rule Multiparty Private Pointer Dereference Write Private Value, we have (B) $\{(e) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (C) $((1, \gamma^1,\ \sigma^1,\ \Delta^1,$ acc, $e) \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow_{\mathcal{D}_1}^{\mathcal{L}_1} ((1, \gamma^1, \sigma_1^1, \Delta_1^1, \mathrm{acc}, n^1) \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1^{\mathrm{q}}, \Delta_1^{\mathrm{q}}, \mathrm{acc}, n^{\mathrm{q}}))$, (D) $\{\gamma^{\mathrm{p}}(x) = (l^{\mathrm{p}}, \text{private } bty*)\}_{\mathrm{p}=1}^{\mathrm{q}}$, (E) $\{\sigma_1^{\mathrm{p}}(l^{\mathrm{p}}) = (\omega^{\mathrm{p}}, \text{ private } bty*,\ \alpha, \mathrm{PermL}(\mathrm{Freeable}, \text{private } bty*, \text{private}, \alpha))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (F) $\alpha > 1$, (G) $\{\mathrm{DecodePtr}(\text{private } bty*, \alpha, \omega^{\mathrm{p}}) = [\alpha,\ \bar{l}^{\mathrm{p}},\ \bar{j}^{\mathrm{p}}, 1]\}_{\mathrm{p}=1}^{\mathrm{q}}$, (H) $\{\mathrm{DynamicUpdate}(\Delta_1^{\mathrm{p}}, \sigma_1^{\mathrm{p}}, \bar{l}^{\mathrm{p}},\ \mathrm{acc}, \text{private } bty)$ $= (\Delta_2^{\mathrm{p}}, \bar{l}_1^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$,

(I) $\{\mathrm{Retrieve_vals}(\alpha, \bar{l}^{\mathrm{p}}, \text{private } bty, \sigma_1^{\mathrm{p}}) = ([n_0^{\mathrm{p}}, ...n_{\alpha-1}^{\mathrm{p}}], 1)\}_{\mathrm{p}=1}^{\mathrm{q}}$, (J) $\mathrm{MPC}_{wdv}([[n_0^1, ..., n_{\alpha-1}^1], ..., [n_0^{\mathrm{q}}, ..., n_{\alpha-1}^{\mathrm{q}}]],$ $[n^1, ..., n^{\mathrm{q}}], [\bar{j}^1,\ ...,\ \bar{j}^{\mathrm{q}}]) = ([n_0'^1, ..., n_{\alpha-1}'^1], ..., [n_0'^{\mathrm{q}}, ..., n_{\alpha-1}'^{\mathrm{q}}])$, and (K) $\{\mathrm{UpdateDerefVals}(\alpha, \bar{l}^{\mathrm{p}},\ [n_0'^{\mathrm{p}}, ..., n_{\alpha-1}'^{\mathrm{p}}],$ private bty, $\sigma_1^{\mathrm{p}}) = \sigma_2^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (L) $\Sigma \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \mathrm{acc},\ *x = e)\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}},\ \sigma^{\mathrm{q}},\ \Delta^{\mathrm{q}},\ \mathrm{acc},\ *x = e))$ $\Downarrow_{\mathcal{D}_1'::(\mathrm{ALL},[d])}^{\mathcal{L}_1'::(1,(l'^1,0)::\bar{l}_1'^1::\bar{l}'^1)\ \parallel\ ...\ \parallel\ (\mathrm{q},(l'^{\mathrm{q}},0)::\bar{l}_1'^{\mathrm{q}}::\bar{l}'^{\mathrm{q}})} ((1, \gamma^1,\ \sigma_2'^1,\ \Delta_2'^1,\ \mathrm{acc},\ \mathrm{skip})\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}},\ \sigma_2'^{\mathrm{q}},\ \Delta_2'^{\mathrm{q}},\ \mathrm{acc},\ \mathrm{skip}))$ and (A), by Lemma 5.2.87 we have (M) $d = mpwdp3$.

Given (L) and (M), by SMC^2 rule Multiparty Private Pointer Dereference Write Private Value, we have (N) $\{(e) \vdash \gamma^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (O) $((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \mathrm{acc},\ e)\ \parallel\ ...\ \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, e)) \Downarrow_{\mathcal{D}_1'}^{\mathcal{L}_1'} ((1, \gamma^1, \sigma_1'^1, \Delta_1'^1, \mathrm{acc}, n'^1)\ \parallel\ ...$ - $\parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_1'^{\mathrm{q}}, \Delta_1'^{\mathrm{q}}, \mathrm{acc}, n'^{\mathrm{q}}))$, (P) $\{\gamma^{\mathrm{p}}(x) = (l'^{\mathrm{p}}, \text{private } bty'*)\}_{\mathrm{p}=1}^{\mathrm{q}}$, (Q) $\{\sigma_1'^{\mathrm{p}}(l'^{\mathrm{p}}) = (\omega'^{\mathrm{p}}, \text{ private } bty'*,\ \alpha',$ $\mathrm{PermL}(\mathrm{Freeable}, \text{private } bty'*, \text{private}, \alpha'))\}_{\mathrm{p}=1}^{\mathrm{q}}$, (R) $\alpha' > 1$, (S) $\{\mathrm{DecodePtr}(\text{private } bty'*, \alpha', \omega'^{\mathrm{p}}) = [\alpha',\ \bar{l}'^{\mathrm{p}},\ \bar{j}'^{\mathrm{p}}, 1]\}_{\mathrm{p}=1}^{\mathrm{q}}$,

(T) $\{\mathrm{DynamicUpdate}(\Delta_1'^{\mathrm{p}}, \sigma_1'^{\mathrm{p}}, \bar{l}'^{\mathrm{p}},\ \mathrm{acc}, \text{private } bty') = (\Delta_2'^{\mathrm{p}}, \bar{l}_1'^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$,

(U) $\{\mathrm{Retrieve_vals}(\alpha', \bar{l}'^{\mathrm{p}}, \text{private } bty', \sigma_1'^{\mathrm{p}}) = ([n_0''^{\mathrm{p}}, ...n_{\alpha'-1}''^{\mathrm{p}}], 1)\}_{\mathrm{p}=1}^{\mathrm{q}}$, (V) $\mathrm{MPC}_{wdv}([[n_0''^1, ..., n_{\alpha'-1}''^1], ..., [n_0''^{\mathrm{q}}, ...,$ $n_{\alpha'-1}''^{\mathrm{q}}]], [n'^1, ..., n'^{\mathrm{q}}], [\bar{j}'^1, ..., \bar{j}'^{\mathrm{q}}]) = ([n_0'''^1, ..., n_{\alpha'-1}'''^1], ..., [n_0'''^{\mathrm{q}}, ..., n_{\alpha'-1}'''^{\mathrm{q}}])$, and

(W) $\{\mathrm{UpdateDerefVals}(\alpha', \bar{l}'^{\mathrm{p}}, [n_0'''^{\mathrm{p}}, ..., n_{\alpha'-1}'''^{\mathrm{p}}], \text{private } bty', \sigma_1'^{\mathrm{p}}) = \sigma_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (C) and (O), by the inductive hypothesis we have (X) $\{\sigma_1^{\mathrm{p}} = \sigma_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (Y) $\{\Delta_1^{\mathrm{p}} = \Delta_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (Z) $\{n^{\mathrm{p}} = n'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$, (A1) $\mathcal{D}_1 = \mathcal{D}_1'$, and (B1) $\mathcal{L}_1 = \mathcal{L}_1'$.

Given (D) and (P), by Definition 5.3.3 we have (C1) $\{l^{\mathrm{p}} = l'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (D1) $bty = bty'$.

Given (E), (Q), (X), and (C1), by Definition 5.3.4 we have (E1) $\{\omega^{\mathrm{p}} = \omega'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (F1) $\alpha = \alpha'$.

Given (G), (S), (W), and (X), by Lemma 5.3.26 we have (G1) $\{\bar{l}^{\mathrm{p}} = \bar{l}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (H1) $\{\bar{j}^{\mathrm{p}} = \bar{j}'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (F), (N), (Q), (P), (Y), and (W), by Lemma 5.3.25 we have (I1) $\{\Delta_2^{\mathrm{p}} = \Delta_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$ and (J1) $\{\bar{l}_1^{\mathrm{p}} = \bar{l}_1'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (I), (U), (F1), (G1), (D1), and (X), by Lemma 5.3.39 we have (K1) $\{[n_0^{\mathrm{p}}, ...n_{\alpha-1}^{\mathrm{p}}] = [n_0''^{\mathrm{p}}, ...n_{\alpha'-1}''^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$

Given (J), (V), (K1), (Z), and (H1), by Axiom 5.3.14 we have (L1) $\{[n_0'^{\mathrm{p}}, ..., n_{\alpha-1}'^{\mathrm{p}}] = [n_0'''^{\mathrm{p}}, ..., n_{\alpha'-1}'''^{\mathrm{p}}]\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (K), (W), (F1), (G1), (L1), (D1), and (X), by Lemma 5.3.43 we have (M1) $\{\sigma_2^{\mathrm{p}} = \sigma_2'^{\mathrm{p}}\}_{\mathrm{p}=1}^{\mathrm{q}}$.

Given (A1) and $(\mathrm{ALL}, [mpwdp3])$, by Lemma 5.3.38 we have (N1) $\mathcal{D}_1 :: (\mathrm{ALL}, [mpwdp3]) = \mathcal{D}_1' :: (\mathrm{ALL}, [mpwdp3])$.

Given (E) and (G), by Lemma 5.3.62 we have accessed location (O1) $\{(\mathrm{p}, [(l^{\mathrm{p}}, 0)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (N), by Lemma 5.3.61 we have accessed locations (P1) $\{(\mathrm{p}, \bar{l}_1^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ Given (I) and (K), by Lemma 5.3.72 and 5.3.76 we have accessed locations (Q1) $\{(\mathrm{p}, \bar{l}^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$ Given (C), (O1), (P1), and (Q1), by Lemmas 5.3.44 and 5.3.45 we have (R1) $\mathcal{L}_1 :: (1, (l^1, 0) :: \bar{l}_1^1 :: \bar{l}^1)$ $\parallel$ ... $\parallel$ $(\mathrm{q}, (l^{\mathrm{q}}, 0) :: \bar{l}_1^{\mathrm{q}} :: \bar{l}^{\mathrm{q}})$.

Given (Q) and (S), by Lemma 5.3.62 we have accessed location (S1) $\{(\mathrm{p}, [(l'^{\mathrm{p}}, 0)])\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (T), by Lemma 5.3.61 we have accessed locations (T1) $\{(\mathrm{p}, \bar{l}_1'^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (U) and (W), by Lemma 5.3.72 and 5.3.76 we have accessed locations (U1) $\{(\mathrm{p}, \bar{l}'^{\mathrm{p}})\}_{\mathrm{p}=1}^{\mathrm{q}}$. Given (O), (S1), (T1), and (U1), by Lemmas 5.3.44 and 5.3.45 we have (V1) $\mathcal{L}_1' ::$ $(1, (l'^1, 0) :: \bar{l}_1'^1 :: \bar{l}'^1)$ $\parallel$ ... $\parallel$ $(\mathrm{q}, (l'^{\mathrm{q}}, 0) :: \bar{l}_1'^{\mathrm{q}} :: \bar{l}'^{\mathrm{q}})$.

Given (R1), (V1), (B1), (C1), (J1), and (G1), by Lemma 5.3.47 we have (W1) $\mathcal{L}_1 :: (1, (l^1, 0) :: \bar{l}_1^1 :: \bar{l}^1)$ $\parallel$ ... $\parallel$ $(\mathrm{q}, (l^{\mathrm{q}}, 0) :: \bar{l}_1^{\mathrm{q}} :: \bar{l}^{\mathrm{q}}) = \mathcal{L}_1' :: (1, (l'^1, 0) :: \bar{l}_1'^1 :: \bar{l}'^1)$ $\parallel$ ... $\parallel$ $(\mathrm{q}, (l'^{\mathrm{q}}, 0) :: \bar{l}_1'^{\mathrm{q}} :: \bar{l}'^{\mathrm{q}})$

Given (M1), (I1), (W1), and (N1), by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \mathrm{acc}, *x = e)\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, *x = e)) \Downarrow_{\mathcal{D}_1 :: (\mathrm{ALL}, [mpwdp])}^{\mathcal{L}_1 :: (1, (l^1, 0) :: \bar{l}_1^1 :: \bar{l}^1) \parallel ... \parallel (\mathrm{q}, (l^{\mathrm{q}}, 0) :: \bar{l}_1^{\mathrm{q}} :: \bar{l}^{\mathrm{q}})}$ $((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \mathrm{acc},\ \mathrm{skip})\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip}))$

This case is similar to Case $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \mathrm{acc}, *x = e)\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, *x = e))$ $\Downarrow_{\mathcal{D}_1 :: (\mathrm{ALL}, [mpwdp3])}^{\mathcal{L}_1 :: (1, (l^1, 0) :: \bar{l}_1^1 :: \bar{l}^1) \parallel ... \parallel (\mathrm{q}, (l^{\mathrm{q}}, 0) :: \bar{l}_1^{\mathrm{q}} :: \bar{l}^{\mathrm{q}})}$ $((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \mathrm{acc},\ \mathrm{skip})\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma_2^{\mathrm{q}}, \Delta_2^{\mathrm{q}}, \mathrm{acc}, \mathrm{skip}))$. We use Axiom 5.3.1 to reason about the use of encrypt.

Case $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \mathrm{acc},\ *x = e)\ \parallel ... \parallel\ (\mathrm{q}, \gamma^{\mathrm{q}}, \sigma^{\mathrm{q}}, \Delta^{\mathrm{q}}, \mathrm{acc}, *x = e)) \Downarrow_{\mathcal{D}_1 :: (\mathrm{ALL}, [mpwdp2])}^{\mathcal{L}_1 :: (1, (l^1, 0) :: \bar{l}_1^1 :: \bar{l}^1) \parallel ... \parallel (\mathrm{q}, (l^{\mathrm{q}}, 0) :: \bar{l}_1^{\mathrm{q}} :: \bar{l}^{\mathrm{q}})}$

$((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \text{acc},\ \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_2^{\text{q}}, \Delta_2^{\text{q}}, \text{acc}, \text{skip}))$

This case is similar to Case $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc}, *x = e)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, *x = e))$ $\Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}_1^1::\bar{l}^1)\ \|\ ...\ \|\ (\text{q},(l^{\text{q}},0)::\bar{l}_1^{\text{q}}::\bar{l}^{\text{q}})}_{\mathcal{D}_1::(\text{ALL},[mpwdp3])}$ $((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \text{acc},\ \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_2^{\text{q}}, \Delta_2^{\text{q}}, \text{acc}, \text{skip}))$. We use Axiom 5.3.15 to reason about the use of MPC_{wdp}.

Case $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc},\ *x = e)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, *x = e)) \Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}_1^1::\bar{l}^1)\ \|\ ...\ \|\ (\text{q},(l^{\text{q}},0)::\bar{l}_1^{\text{q}}::\bar{l}^{\text{q}})}_{\mathcal{D}_1::(\text{ALL},[mpwdp1])}$ $((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \text{acc},\ \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_2^{\text{q}}, \Delta_2^{\text{q}}, \text{acc}, \text{skip}))$

This case is similar to Case $\Pi \triangleright ((1, \gamma^1,\ \sigma^1,\ \Delta^1,\ \text{acc}, *x = e)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, *x = e))$ $\Downarrow^{\mathcal{L}_1::(1,(l^1,0)::\bar{l}_1^1::\bar{l}^1)\ \|\ ...\ \|\ (\text{q},(l^{\text{q}},0)::\bar{l}_1^{\text{q}}::\bar{l}^{\text{q}})}_{\mathcal{D}_1::(\text{ALL},[mpwdp3])}$ $((1, \gamma^1,\ \sigma_2^1,\ \Delta_2^1,\ \text{acc},\ \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_2^{\text{q}}, \Delta_2^{\text{q}}, \text{acc}, \text{skip}))$. We use Axiom 5.3.15 to reason about the use of MPC_{wdp}.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, ++\ x)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, ++\ x)) \Downarrow^{(1,[(l^1,0)])\ \|\ ...\ \|\ (\text{q},[(l^{\text{q}},0)])}_{(\text{ALL},[mppin])}$ $((1, \gamma^1, \sigma_1^1,\ \Delta^1,\ \text{acc}, n_2^1)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_1^{\text{q}}, \Delta^{\text{q}}, \text{acc}, n_2^{\text{q}}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, ++\ x)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, ++\ x)) \Downarrow^{(1,[(l^1,0)])\ \|\ ...\ \|\ (\text{q},[(l^{\text{q}},0)])}_{(\text{ALL},[mppin])}$ $((1, \gamma^1, \sigma_1^1,\ \Delta^1,\ \text{acc}, n_2^1)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_1^{\text{q}}, \Delta^{\text{q}}, \text{acc}, n_2^{\text{q}}))$ by SMC^2 rule Multiparty Pre-Increment Private Float Variable, we have (B) $\{\gamma^{\text{p}}(x) = (l^{\text{p}}, \text{private float})\}_{\text{p}=1}^{\text{q}}$, (C) $\{\sigma^{\text{p}}(l^{\text{p}}) = (\omega^{\text{p}}, \text{private float}, 1,\ \text{PermL}(\text{Freeable}, \text{private float}, \text{private}, 1))\}_{\text{p}=1}^{\text{q}}$, (D) $\{(x) \vdash \gamma^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (E) $\{\text{DecodeVal}(\text{private float}, \omega^{\text{p}}) = n_1^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (F) $\text{MPC}_u(++, n_1^1, ..., n_1^{\text{q}}) = (n_2^1, ..., n_2^{\text{q}})$, and (G) $\{\text{UpdateVal}(\sigma^{\text{p}}, l^{\text{p}}, n_2^{\text{p}}, \text{private float}) = \sigma_1^{\text{p}}\}_{\text{p}=1}^{\text{q}}$.

Given (H) $\Sigma \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, ++\ x)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, ++\ x)) \Downarrow^{(1,[(l'^1,0)])\ \|\ ...\ \|\ (\text{q},[(l'^{\text{q}},0)])}_{(\text{ALL},[d])}$ $((1, \gamma^1, \sigma_1'^1,\ \Delta^1,\ \text{acc}, n_2'^1)\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_1'^{\text{q}}, \Delta^{\text{q}}, \text{acc}, n_2'^{\text{q}}))$ and (A), by Lemma 5.2.87 we have (I) $d = mppin$.

Given (H) and (I), by SMC^2 rule Multiparty Pre-Increment Private Float Variable, we have

(J) $\{\gamma^{\text{p}}(x) = (l'^{\text{p}}, \text{private float})\}_{\text{p}=1}^{\text{q}}$,

(K) $\{\sigma^{\text{p}}(l'^{\text{p}}) = (\omega'^{\text{p}}, \text{private float}, 1, \text{PermL}(\text{Freeable}, \text{private float}, \text{private}, 1))\}_{\text{p}=1}^{\text{q}}$,

(L) $\{(x) \vdash \gamma^{\text{p}}\}_{\text{p}=1}^{\text{q}}$,

(M) $\{\text{DecodeVal}(\text{private float}, \omega'^{\text{p}}) = n_1'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$,

(N) $\text{MPC}_u(++, n_1'^1, ..., n_1'^{\text{q}}) = (n_2'^1, ..., n_2'^{\text{q}})$, and

(O) $\{\text{UpdateVal}(\sigma^{\text{p}}, l'^{\text{p}}, n_2'^{\text{p}}, \text{private float}) = \sigma_1'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$.

Given (B) and (J), by Definition 5.3.3 we have (P) $\{l^{\text{p}} = l'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$.

Given (C), (K), and (P), by Definition 5.3.4 we have (Q) $\{\omega^{\text{p}} = \omega'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$.

Given (E), (M), and (Q), by Lemma 5.3.29 we have (R) $\{n_1^{\text{p}} = n_1'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$.

Given (F), (N), and (R), by Axiom 5.3.9 we have (S) $\{n_2^{\text{p}} = n_2'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$.

Given (G), (O), (P), and (S), by Lemma 5.3.34 we have (T) $\{\sigma_1^{\text{p}} = \sigma_1'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$.

Given (A), (H), and (I), we have (U) $(\text{ALL}, [mppin]) = (\text{ALL}, [mppin])$.

Given (C), (E), and (G), by Lemma 5.3.64 and Lemma 5.3.66 we have accessed location (V) $\{(\text{p}, [(l, 0)])\}_{\text{p}=1}^{\text{q}}$. Given (V), by Lemmas 5.3.44 and 5.3.46 we have (W) $(1, [(l^1, 0)]) \parallel ... \parallel (\text{q}, [(l^{\text{q}}, 0)])$.

Given (K), (M), and (O), by Lemma 5.3.64 and Lemma 5.3.66 we have accessed location (X) $\{(\text{p}, [(l', 0)])\}_{\text{p}=1}^{\text{q}}$. Given (X), by Lemmas 5.3.44 and 5.3.46 we have (Y) $(1, [(l'^1, 0)]) \parallel ... \parallel (\text{q}, [(l'^{\text{q}}, 0)])$.

Given (W), (Y), and (P), we have (Z) $(1, [(l^1, 0)]) \parallel ... \parallel (\text{q}, [(l^{\text{q}}, 0)]) = (1, [(l'^1, 0)]) \parallel ... \parallel (\text{q}, [(l'^{\text{q}}, 0)])$.

Given (T), (S), (U), and (Z) by Definition 5.3.2 we have $\Pi \simeq_L \Sigma$.

Case $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, \text{pfree}(x)) \ \parallel ... \parallel \ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{pfree}(x)))$
$\Downarrow^{(1,[(l^1,0)]::\bar{l}^1::\bar{l}_1^1) \ \parallel ... \parallel \ (\text{q},[(l^{\text{q}},0)]::\bar{l}^{\text{q}}::\bar{l}_1^{\text{q}})}_{(\text{ALL},[mpfre])} ((1, \gamma^1, \sigma_2^1, \Delta^1, \text{acc}, \text{skip}) \ \parallel ... \parallel \ (\text{q}, \gamma^{\text{q}}, \sigma_2^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{skip}))$

Given (A) $\Pi \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, \text{pfree}(x)) \ \parallel ... \parallel \ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{pfree}(x)))$
$\Downarrow^{(1,[(l^1,0)]::\bar{l}^1::\bar{l}_1^1) \ \parallel ... \parallel \ (\text{q},[(l^{\text{q}},0)]::\bar{l}^{\text{q}}::\bar{l}_1^{\text{q}})}_{(\text{ALL},[mpfre])} ((1, \gamma^1, \ \sigma_2^1, \ \Delta^1, \text{acc}, \text{skip}) \ \parallel ... \parallel \ (\text{q}, \gamma^{\text{q}}, \sigma_2^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{skip}))$ by SMC^2 rule Private Free Multiple Locations, we have (B) $\{\gamma^{\text{p}}(x) = (l^{\text{p}}, \ \text{private } bty*)\}_{\text{p}=1}^{\text{q}}$, (C) $\text{acc} = 0$, (D) $(bty = \text{int}) \vee (bty = \text{float})$, (E) $\{\sigma^{\text{p}}(l^{\text{p}}) = (\omega^{\text{p}}, \text{private } bty*, \alpha, \text{PermL}(\text{Freeable}, \text{private } bty*, \text{private}, \alpha))\}_{\text{p}=1}^{\text{q}}$, (F) $\{\alpha > 1\}_{\text{p}=1}^{\text{q}}$, (G) $\{[\alpha, \ \bar{l}^{\text{p}}, \ \bar{j}^{\text{p}}, \ i] = \text{DecodePtr}(\text{private } bty*, \ \alpha, \ \omega^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (H) $\text{if}(i > 1)\{ty = \text{private } bty*\}$ else $\{ty = \text{private } bty\}$,

(I) $\{\text{CheckFreeable}(\gamma^{\text{p}}, \bar{l}^{\text{p}}, \bar{j}^{\text{p}}, \sigma^{\text{p}}) = 1\}_{\text{p}=1}^{\text{q}}$, (J) $\{\forall(l_m^{\text{p}}, 0) \in \bar{l}^{\text{p}}.\ \sigma^{\text{p}}(l_m^{\text{p}}) = (\omega_m^{\text{p}}, ty, \alpha_m,$ $\text{PermL}(\text{Freeable},\ ty, \text{private}, \alpha_m))\}_{\text{p}=1}^{\text{q}}$, (K) $\text{MPC}_{free}([[\omega_0^1, ..., \omega_{\alpha-1}^1], ..., [\omega_0^{\text{q}}, ..., \omega_{\alpha-1}^{\text{q}}]], [\bar{j}^1, ...\bar{j}^{\text{q}}]) = ([[\omega_0'^1, ..., \omega_{\alpha-1}'^1], ..., [\omega_0'^{\text{q}}, ..., \omega_{\alpha-1}'^{\text{q}}]], [\bar{j}'^1, ..., \bar{j}'^{\text{q}}])$, (L) $\{\text{UpdateBytesFree}(\sigma^{\text{p}}, \bar{l}^{\text{p}}, [\omega_0'^{\text{p}}, ..., \omega_{\alpha-1}'^{\text{p}}]) = \sigma_1^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, and
(M) $\{(\sigma_2^{\text{p}}, \bar{l}_1^{\text{p}}) = \text{UpdatePointerLocations}(\sigma_1^{\text{p}}, \bar{l}^{\text{p}}[1:\alpha-1], \bar{j}^{\text{p}}[1:\alpha-1], \bar{l}^{\text{p}}[0], \bar{j}^{\text{p}}[0])\}_{\text{p}=1}^{\text{q}}$.

Given (N) $\Sigma \triangleright ((1, \gamma^1, \sigma^1, \Delta^1, \text{acc}, \text{pfree}(x)) \ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{pfree}(x)))$ $\Downarrow_{(\text{ALL},[d])}^{(1,[(l'^1,0)]::\bar{l}'^1::\bar{l}_1'^1)\ \|\ ...\ \|\ (\text{q},[(l'^{\text{q}},0)]::\bar{l}'^{\text{q}}::\bar{l}_1'^{\text{q}})} ((1, \gamma^1, \sigma_2'^1, \Delta^1, \text{acc}, \text{skip})\ \|\ ...\ \|\ (\text{q}, \gamma^{\text{q}}, \sigma_2'^{\text{q}}, \Delta^{\text{q}}, \text{acc}, \text{skip}))$ and (A), by Lemma 5.2.87 we have (O) $d = mpfre$.

Given (N) and (O), by SMC^2 rule Private Free Multiple Locations, we have (P) $\{\gamma^{\text{p}}(x) = (l'^{\text{p}},\ \text{private}\ bty'*)\}_{\text{p}=1}^{\text{q}}$, (Q) $\text{acc} = 0$, (R) $(bty' = \text{int}) \vee (bty' = \text{float})$, (S) $\{\sigma^{\text{p}}(l'^{\text{p}}) = (\omega'^{\text{p}}, \text{private}\ bty'*, \alpha',\ \text{PermL}(\text{Freeable}, \text{private}\ bty'*, \text{private}, \alpha'))\}_{\text{p}=1}^{\text{q}}$, (T) $\{\alpha' > 1\}_{\text{p}=1}^{\text{q}}$, (U) $\{[\alpha',\ \bar{l}'^{\text{p}},\ \bar{j}''^{\text{p}},\ i'] = \text{DecodePtr}(\text{private}\ bty'*,\ \alpha',\ \omega'^{\text{p}})\}_{\text{p}=1}^{\text{q}}$, (V) $\text{if}(i' > 1)\{ty' = \text{private}\ bty'*\}$ else $\{ty' = \text{private}\ bty'\}$, (W) $\{\text{CheckFreeable}(\gamma^{\text{p}}, \bar{l}'^{\text{p}}, \bar{j}''^{\text{p}}, \sigma^{\text{p}}) = 1\}_{\text{p}=1}^{\text{q}}$, (X) $\{\forall(l_{m'}'^{\text{p}}, 0) \in \bar{l}'^{\text{p}}.\ \sigma^{\text{p}}(l_{m'}'^{\text{p}}) = (\omega_{m'}''^{\text{p}}, ty', \alpha_{m'}',\ \text{PermL}(\text{Freeable},\ ty', \text{private}, \alpha_{m'}'))\}_{\text{p}=1}^{\text{q}}$, (Y) $\text{MPC}_{free}([[\omega_0''^1, ..., \omega_{\alpha'-1}''^1], ..., [\omega_0''^{\text{q}}, ..., \omega_{\alpha'-1}''^{\text{q}}]], [\bar{j}''^1, ...\bar{j}''^{\text{q}}]) = ([[\omega_0'''^1, ..., \omega_{\alpha'-1}'''^1], ..., [\omega_0'''^{\text{q}}, ..., \omega_{\alpha'-1}'''^{\text{q}}]], [\bar{j}'''^1, ..., \bar{j}'''^{\text{q}}])$,
(Z) $\{\text{UpdateBytesFree}(\sigma^{\text{p}}, \bar{l}'^{\text{p}}, [\omega_0'''^{\text{p}}, ..., \omega_{\alpha'-1}'''^{\text{p}}]) = \sigma_1'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, and
(A1) $\{(\sigma_2'^{\text{p}}, \bar{l}_1'^{\text{p}}) = \text{UpdatePointerLocations}(\sigma_1'^{\text{p}}, \bar{l}'^{\text{p}}[1:\alpha'-1], \bar{j}''^{\text{p}}[1:\alpha'-1], \bar{l}'^{\text{p}}[0], \bar{j}''^{\text{p}}[0])\}_{\text{p}=1}^{\text{q}}$.

Given (B) and (P), by Definition 5.3.3 we have (B1) $\{l^{\text{p}} = l'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$ and (C1) $bty = bty'$.

Given (E), (S), and (B1), by Definition 5.3.4 we have (D1) $\{\omega^{\text{p}} = \omega'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$ and (E1) $\alpha = \alpha'$.

Given (G), (U), (C1), (E1), and (D1), by Lemma 5.3.26 we have (F1) $\{\bar{l}^{\text{p}} = \bar{l}'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, (G1) $\{\bar{j}^{\text{p}} = \bar{j}''^{\text{p}}\}_{\text{p}=1}^{\text{q}}$, and (H1) $i = i'$.

Given (H), (V), (H1), and (C1), we have (I1) $ty = ty'$.

Given (I), (W), (F1), and (G1), by Lemma 5.3.40 we have (J1) $1 = 1$.

Given (J), (X), and (F1), we have (K1) $\{l_m^{\text{p}} = l_{m'}'^{\text{p}}\}_{\text{p}=1}^{\text{q}}$ such that (L1) $m = m'$. Given (J), (X), (F1), (K1), (E1), and (L1), by Definition 5.3.4 we have (M1) $\{\forall m = m' \in \{0...\alpha-1\},\ \omega_m^{\text{p}} = \omega_{m'}''^{\text{p}}\}_{\text{p}=1}^{\text{q}}$ and (N1) $\forall m = m' \in \{0...\alpha-1\},\ \alpha_m = \alpha_{m'}'$.

Given (K), (Y), (E1), (M1), and (G1), by Axiom 5.2.15 we have (O1) $\{\forall m = m' \in \{0...\alpha - 1\},\ \omega'^{\mathrm{p}}_{m} = \omega'''^{\mathrm{p}}_{m'}\}^{\mathrm{q}}_{\mathrm{p}=1}$ and (P1) $\{\bar{j}'^{\mathrm{p}} = \bar{j}'''^{\mathrm{p}}\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Given (L), (Z), (F1), (O1), and (E1), by Lemma 5.3.41 we have (Q1) $\{\sigma^{\mathrm{p}}_{1} = \sigma'^{\mathrm{p}}_{1}\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Given (M), (A1), (Q1), (F1), (E1), and (G1), by Lemma 5.3.42 we have (R1) $\{\sigma^{\mathrm{p}}_{2} = \sigma'^{\mathrm{p}}_{2}\}^{\mathrm{q}}_{\mathrm{p}=1}$ and (S1) $\{\bar{l}^{\mathrm{p}}_{1} = \bar{l}'^{\mathrm{p}}_{1}\}^{\mathrm{q}}_{\mathrm{p}=1}$.

Given (A), (N), and (O), we have (T1) $(\mathrm{ALL}, [mpfre]) = (\mathrm{ALL}, [mpfre])$.

Given (E) and (G), by Lemma 5.3.62 we have accessed location (U1) $\{(\mathrm{p}, [(l^{\mathrm{p}}, 0)])\}^{\mathrm{q}}_{\mathrm{p}=1}$. Given (I), (J), (L), by Lemma 5.3.73 and Lemma 5.3.74 we have accessed locations (V1) $\{(\mathrm{p}, \bar{l}^{\mathrm{p}})\}^{\mathrm{q}}_{\mathrm{p}=1}$. Given (M), by Lemma 5.3.75 we have accessed locations (W1) $\{(\mathrm{p}, \bar{l}^{\mathrm{p}}_{1})\}^{\mathrm{q}}_{\mathrm{p}=1}$. Given (U1), (V1), and (W1), by Lemmas 5.3.44 and 5.3.46 we have (X1) $(1, [(l^{1}, 0)] :: \bar{l}^{1} :: \bar{l}^{1}_{1})\ \|\ ...\ \|\ (\mathrm{q}, [(l^{\mathrm{q}}, 0)] :: \bar{l}^{\mathrm{q}} :: \bar{l}^{\mathrm{q}}_{1})$.

Given (S) and (U), by Lemma 5.3.62 we have accessed location (Y1) $\{(\mathrm{p}, [(l'^{\mathrm{p}}, 0)])\}^{\mathrm{q}}_{\mathrm{p}=1}$. Given (W), (X), (Z), by Lemma 5.3.73 and Lemma 5.3.74 we have accessed locations (Z1) $\{(\mathrm{p}, \bar{l}'^{\mathrm{p}})\}^{\mathrm{q}}_{\mathrm{p}=1}$. Given (A1), by Lemma 5.3.75 we have accessed locations (A2) $\{(\mathrm{p}, \bar{l}'^{\mathrm{p}}_{1})\}^{\mathrm{q}}_{\mathrm{p}=1}$. Given (Y1), (Z1), and (A2), by Lemmas 5.3.44 and 5.3.46 we have (B2) $(1, [(l'^{1}, 0)] :: \bar{l}'^{1} :: \bar{l}'^{1}_{1})\ \|\ ...\ \|\ (\mathrm{q}, [(l'^{\mathrm{q}}, 0)] :: \bar{l}'^{\mathrm{q}} :: \bar{l}'^{\mathrm{q}}_{1})$.

Given (X1), (B2), (B1), (F1), and (S1), we have (C2) $(1, [(l^{1}, 0)] :: \bar{l}^{1} :: \bar{l}^{1}_{1})\ \|\ ...\ \|\ (\mathrm{q}, [(l^{\mathrm{q}}, 0)] :: \bar{l}^{\mathrm{q}} :: \bar{l}^{\mathrm{q}}_{1}) = (1, [(l'^{1}, 0)] :: \bar{l}'^{1} :: \bar{l}'^{1}_{1})\ \|\ ...\ \|\ (\mathrm{q}, [(l'^{\mathrm{q}}, 0)] :: \bar{l}'^{\mathrm{q}} :: \bar{l}'^{\mathrm{q}}_{1})$.

Given (R1), (T1), and (C2) by Definition 5.3.2 we have $\Pi \simeq_{L} \Sigma$.

□

6 Realization

6.1 Implementing Multiparty SMC^2 in PICCO

We implement our semantics in the PICCO compiler. Our implementation is available at `https://github.com/amypritc/SMC2` and is currently being merged into the main PICCO branch as a submitted patch. The main modifications to PICCO revolve around a *conditional code block tracking* scheme and resolution mechanism for private variables, support for pointer operations and computation within private-conditioned branches, and the addition of tracking structures for pointers based on our semantics in Chapter 5. PICCO uses single-statement resolution to manage modifications to private variables made within a private-conditioned branch, which can prove to be more costly when a single variable is modified multiple times within a private-conditioned branch, as shown in the example in Figure 6.1. Here, we show how PICCO conditionally updates the true value of a variable after each statement, resulting in a more costly operation for each statement, as obtaining the true value requires communication between the various computational parties (i.e. this is a distributed secure computation). In SMC^2, we provide a conditional code block tracking scheme, where we store the original value for each modified variable before the execution of the **`then`** branch, execute the **`then`** branch as normal, store the updated values, restore the original values, execute the **`else`** branch as normal, and perform resolution of all modified variables once at the end of both branches. The cost of executing each statement remains the same, however, in SMC^2, the communication overhead is greatly reduced due to only needing to communicate between computational parties to resolve a single variable once at the end of both branches. In Figures 6.1b and 6.1c, this amounts to six fewer times the program needs to communicate between computational parties to resolve.

6.1.1 Conditional Code Block Tracking Implementation

To implement data-oblivious execution of branches on private data, SMC implementations typically execute both branches, but privately apply the effects of only the relevant branch. Figure 6.2a shows the standard handling of private-conditioned branches, as we have formalized in Basic SMC^2. The top left shows the original C-code block, with annotations for private data; the top right shows how compilers would flatten the

```
1 private int c,
2     a=1,b=2;
3 if(a < b) {
4   c = a;
5   a = a + b;
6   c = c * b;
7   a = c + a;
8 } else {
9   c = b;
10  a = a - b;
11  c = c * a;
12  a = c - a;
13 }
```

(a) Example code.

```
1 private int c,a=1,b=2;
2 private int res=a<b;
3 {
4 c=(res·a)+((1-res)·c);
5 a=(res·(a+b))+((1-res)·a);
6 c=(res·(c*b))+((1-res)·c);
7 a=(res·(c+a))+((1-res)·a);
8 }
9 {c=((1-res)·b)+(res·c);
10 a=((1-res)·(a-b))+(res·a);
11 c=((1-res)·(c*a))+(res·c);
12 a=((1-res)·(c-a))+(res·a);
13 }
```

(b) How PICCO executes 6.1a.

```
1 private int c,a=1,b=2,c_t,
2   res=a<b,c_e=c,a_t,a_e=a;
3 {c = a;
4  a = a + b;
5  c = c * b;
6  a = c + a;}
7 c_t=c;c=c_e;a_t=a;a=a_e;
8 {c = b;
9  a = a - b;
10 c = c * a;
11 a = c - a;}
12 c=(res·c_t)+((1-res)·c);
13 a=(res·a_t)+((1-res)·a);
```

(c) How SMC^2 executes 6.1a.

Figure 6.1: Example illustrating why single-statement resolution is more costly when modifying variables multiple times in both branches.

branch to hide the private data used in the condition. Here, the only variable modified within either branch is `c`, so we insert temporary variables to assist in tracking the results of both branches. Notation $l_{\mathtt{x}}$ in the table indicates the (private) value stored at the location of variable `x`. For every variable we list the value in the `initial` state, before executing the conditional; the one after executing the `then` branch; the one after the initial state has been restored; the one after the execution of the `else` branch; and the one after the true value of `c` has been resolved.

To guarantee data-oblivious executions we also need to guarantee that when we branch on a private condition, the two branches do not have different public side effects. A way to address this is to reject programs that contain public side effects in the body of private-conditioned branching statements. This has an impact on other language constructs such as functions. That is, to be able to call a function from the body of such a branching statement, it must have no public side effects. To address this challenge, in our formalism we evaluate each declared function for public side effects and mark it with a flag that indicates whether it is allowed to be called from within a private-conditioned branch.

PICCO is the only compiler we are aware of that treats pointers to private values, or private pointers. For private pointers, the location being pointed to might be private as well. That is, if a pointer is assigned a new value inside a private-conditioned branch, we cannot reveal whether the original or new location is to be used. For that reason, a private pointer is associated with multiple locations and a private tag that indicates which location is the true location. In particular, a private pointer is represented as a data structure which tracks the number of locations α being pointed to; a list of these α (known) locations; a list of α (private) tags, one of

```
1 private int a=3,b=7,c=0;
2 if(a<b) {c=a;}
3 else {c=b;}
```

```
1 private int a=3,b=7,c=0,res=a<b,c_e=c,c_t;
2 {c=a;} c_t=c; c=c_e;
3 {c=b;} c=(res·c_t)+((1-res)·c);
```

location	init	**then**	restore	**else**	resolve
$l_{\mathtt{c}}$	0	3	0	7	3
$l_{\mathtt{c_t}}$			3	3	3

location	value
$l_{\mathtt{res}}$	1
$l_{\mathtt{c_e}}$	0

(a) Regular variable handling within private-conditioned branches.

```
1 private int a=3,b=7,*p;
2 if(a<b) {p=&a;}
3 else {p=&b;}
```

```
1 private int a=3,b=7,*p,res=a<b,*p_e=p,*p_t;
2 {p=&a;} p_t=p; p=p_e;
3 {p=&b;} p=resolve(res,p_t,p);
```

location	init	**then**	restore	**else**	resolve
$l_{\mathtt{p}}$	( ), ( )	($l_{\mathtt{a}}$), (1)	( ), ( )	($l_{\mathtt{b}}$), (1)	($l_{\mathtt{a}}, l_{\mathtt{b}}$), (1, 0)
$l_{\mathtt{p_t}}$			($l_{\mathtt{a}}$), (1)	($l_{\mathtt{a}}$), (1)	($l_{\mathtt{a}}$), (1)

location	value
$l_{\mathtt{res}}$	1
$l_{\mathtt{p_e}}$	( ), ( )

(b) Pointer handling within private-conditioned branches.

Figure 6.2: Simple private-conditioned branching examples. An example for private integer variables is shown in 6.2a, and for private pointers 6.2b. We show values in memory that change in the table to the left, and values for temporary variables that do not change in the table to the right. We indicate updates in memory in green.

which is set to 1 and all others set to 0; and the level of pointer indirection. Because locations associated with pointers can now be private, there might be additional limitations on what programs can contain within private-conditioned branches to guarantee data-oblivious execution. To understand why multiple locations may potentially be stored for a pointer consider Figure 6.2b, whose code will result in the pointer `p` storing two locations, $l_{\mathtt{a}}$ and $l_{\mathtt{b}}$, with $l_{\mathtt{a}}$ being the true location.

6.1.2 Multiparty SMC^2 Evaluation

To highlight the feasibility of our approach we provide preliminary performances numbers over both microbenchmarks and real-world SMC programs. All experiments were run in a local and distributed manner. We leverage local runs, where all participants in the SMC program execute on the same machine, to analyze overheads and benefits of our approach. We also provide distributed deployment of the same benchmarks to illustrate real-world performance. In the distributed configuration, each participant in the SMC program is executed on a separate machine. We ran our experiments using single-threaded execution on three 2.10GHz machines running CentOS-7. The machines were connected via 1Gbps Ethernet.

Benchmark pay-gap is the program shown in Figure 2.1, where the average female salary and average male salary is computed. Benchmark LR-parser is a program that will parse and execute a mathematical

function over private data. This program is given as input the function as a list of tokens and the private data to execute the function over. Benchmark h_analysis is a program that takes two sets of input data and calculates the percent difference between each element of the two data sets. All of the "pb" benchmarks are programs designed to emphasize the difference in executing using single-statement resolution, as PICCO does, and when using the conditional code block tracking, as in SMC^2. These programs are run through a large number of iterations of branches where variables are modified several times in each branch, as in the example in Figure 6.1.

When executing locally, we ran all three computational parties on a single machine. When executing distributed, one computational party is run on each machine. All programs were run for 50 times both distributed and locally. LR-parser has the shortest execution time of our benchmarks (seconds), and its timing is more influenced by small differences in communication overhead during the computation, resulting in a greater standard deviation. Program pb-reuse has the longest runtime, executing in about 5 and 3.5 minutes locally and 15 and 8 minutes distributed for PICCO and SMC^2, respectively. In our computations, we first took the average time of each of the three computational parties, then the total average and standard deviation of all runs. We calculated the percent speedup in Figure 6.3 using PICCO's timings as a baseline. We can see from our micro-benchmarks which stress private-conditioned branches that our approach provides significant runtime improvement. This is not surprising as our resolution mechanism requires less communication between parties. However, this benefit is reduced in real-world SMC programs and is proportional to the number of private-conditioned branches as well as their complexity (i.e. the number of private variables they use and modify). Similarly we observe that performance improvement or reduction is dilated when communication overheads increase (e.g. we execute in a distributed setting). This is also not surprising as the communication cost as a percentage of total runtime increases in a distributed execution.

Runtime Statistics

To calculate the averages and standard deviation, we first average the runtimes of each of the 3 parties in a single run (i.e., (Party3 + Party2 + Party1)/3). We then use the average timing for each run to obtain the total average and standard deviation for the runtime of each program. To calculate percent speedup with PICCO as the baseline, we used the formula: (PICCO avg - SMC^2 avg)/PICCO avg * 100. To calculate the standard deviation error bars, we used the formula: ((PICCO avg - (SMC^2 avg - SMC^2 st dev))/PICCO avg*100) - percent speedup. We illustrate these statistics in Figure 6.3 and show the values in Tables 6.1 and 6.2

(a) Local

(b) Distributed

Figure 6.3: Percentage Runtime Differences

Program Name	PICCO Average	PICCO Standard Deviation	SMC^2 Average	SMC^2 Standard Deviation
LR-parser	0.00226	0.00047	0.00222	0.00044
pay-gap	4.05632	0.08251	4.08879	0.13961
h_analysis	12.60051	0.14929	12.81269	0.22293
private-branching	1.67252	0.16551	1.57602	0.08988
private-branching-mult	1.89925	0.17203	1.61712	0.14109
private-branching-add	2.30731	0.09335	1.77433	0.1394
private-branching-reuse	307.72411	2.17444	207.99393	1.29311

Table 6.1: Average runtimes and standard deviation for local computation.

Program Name	PICCO Average	PICCO Standard Deviation	SMC^2 Average	SMC^2 Standard Deviation
LR-parser	0.00242	0.00029	0.00242	0.00019
pay-gap	13.86972	0.38221	14.30434	0.25417
h_analysis	33.17922	0.26027	33.16173	0.31844
private-branching	3.44979	0.06592	3.222	0.06049
private-branching-mult	4.56412	0.06466	3.23905	0.06333
private-branching-add	6.45718	0.02443	3.99943	0.14629
private-branching-reuse	923.30999	18.20752	470.81101	9.9084

Table 6.2: Average runtimes and standard deviation for distributed computation.

Programs

Benchmarking program pay-gap is shown in Figure 6.4. Here, we use the PICCO syntax for executing `smcinput` and `smcoutput` this program, as opposed to the simplified syntax used in the Figure 1 in the main paper. For simplicity, we aggregated the input data into a single file, rather than reading from 100 different files.

```
1 public int main() {
2    public int numParticipants = 100, i, j, maxInputSize = 100;
3    public int inputSize[numParticipants], inputNum;
4    private int salary[numParticipants][maxInputSize];
5    private int<1> gender[numParticipants][maxInputSize];
6    private int avgMaleSalary = 0, avgFemaleSalary = 0;
7    private int maleCount = 0, femaleCount = 0;
8    public int historicFemaleSalaryAvg, historicMaleSalaryAvg;
9
10   smcinput(inputSize, 1, numParticipants);
11   smcinput(gender, 1, numParticipants, maxInputSize);
12   smcinput(salary, 1, numParticipants, maxInputSize);
13   smcinput(historicFemaleSalaryAvg, 1); smcinput(historicMaleSalaryAvg, 1);
14   for (i = 0; i < numParticipants; i++){
15      for (j = 0; j < inputSize[i]; j++){
16         if (gender[i][j] == 0) {
17            avgFemaleSalary += salary[i][j];
18            femaleCount++;
19         }
20         else {
21            avgMaleSalary += salary[i][j];
22            maleCount++;
23         }
24      }
25   }
26   avgFemaleSalary=(avgFemaleSalary/femaleCount)/2 + historicFemaleSalaryAvg/2;
27   avgMaleSalary=(avgMaleSalary/maleCount)/2 + historicFemaleSalaryAvg/2;
28   smcoutput(avgFemaleSalary, 1);
29   smcoutput(avgMaleSalary, 1);
30   return 1;
31 }
```

Figure 6.4: Benchmarking program: pay-gap.c

Benchmarking program LR-parser is split into two parts due to the length of the program, and shown in Figures 6.5 and 6.6. When reading the program, be aware that several lines contain multiple statements to be

able to show this program within two figures, and the program contains comments (enclosed in `/*...*/`) to help understand the program.

```
1 public int K = 100; /* max number of variables in the expression */
2   /* this defines integer representation of symbols: */
3   /* + = K; * = K+1; ( = K+2; ) = K+3; EOF = K+4  */
4 public int M = 10; /* the number of variables in the expression */
5 public int S = 29; /* the length of the expression */
6 struct token{ private int val; public int type; struct token* next; };
7   /* type == 0 —— id; type == 1 —— F; type == 2 —— T; type == 3 —— S  */
8   /* type == 4 —— +; type == 5 —— *; type == 6 —— (; type == 7 —— ) */
9 struct token *pop(struct token** header) {
10   struct token* t = *header; struct token* tmp = *header;
11   *header = tmp->next; return t; }
12 public void push(struct token** header, struct token* t) {
13   t->next = *header;  *header = t;  }
14 public void id_routine(struct token** header, int val) {
15   struct token* t; t = pmalloc(1, struct token);
16   t->type = 0; t->val = val; push(header, t); }
17 public void check_for_removable_lbra(struct token** header) {
18   struct token* t; t = pop(header);
19   if(t->type != 6) push(header, t); }
20 public void prod_sub_routine(struct token** header,struct token* x1, public int
21  flag){
22   struct token* x3;  x3 = pop(header);
23   x1->type = 2; /* T */ x1->val =  x3->val * x1->val;
24   if(*header != 0) {
25     struct token* x4; x4 = pop(header);
26     if(x4->type == 4) /* + */ {
27       struct token* x5; x5 = pop(header);
28       x1->val = x1->val + x5->val; x1->type = 3; }
29     else push(header, x4); }
30   if(flag == 1) check_for_removable_lbra(header);
31   push(header, x1); }
32 public void plus_sub_routine(struct token** header,struct token* x1, public int flag){
33   struct token* x3; x3 = pop(header);
34   x1->type = 3; x1->val = x1->val + x3->val;
35   if(flag == 1) check_for_removable_lbra(header);
36   push(header, x1); }
37 public void plus_routine(struct token** header) {
38   struct token* plus; plus = pmalloc(1, struct token); plus->type = 4;
39   struct token* x1; x1 = pop(header);
40   if(*header != 0) {
41     if(x1->type == 0) /* id */ x1->type == 1; /* F */
42     struct token* x2; x2 = pop(header);
43     if(x2->type == 5) /* * */ prod_sub_routine(header, x1, 0);
44     else if(x2->type == 4) /* +  */ plus_sub_routine(header, x1, 0);
45     else if(x2->type == 6) {  /* (  */
46       x1->type = 3; /* S */ push(header, x2); push(header, x1);  } }
47   else { x1->type = 3; push(header, x1); }
48   push(header, plus); }
```

Figure 6.5: Benchmarking program: LR-parser.c (Part 1/2)

```
50 public void prod_routine(struct token** header) {
51   struct token* prod; prod = pmalloc(1, struct token);
52   prod->type = 5; struct token* x1; x1 = pop(header);
53   if(*header != 0){
54     if(x1->type == 0) /* id */ x1->type == 1; /* F */
55     struct token* x2; x2 = pop(header);
56     if(x2->type == 5){
57       struct token* x3; x3 = pop(header);
58       x1->type = 2; x1->val = x1->val * x3->val; push(header, x1); } /* * */
59     else if(x2->type == 4 || x2->type == 6) {  /* + or ( */
60       x1->type = 2; /* T  */ push(header, x2); push(header, x1); } }
61   else { x1->type = 2; push(header, x1); }
62   push(header, prod); }
63 public void lbra_routine(struct token** header) {
64   struct token* t;  t = pmalloc(1, struct token);
65   t->type = 6;  push(header, t); }
66 public void rbra_routine(struct token** header){
67   struct token* x1; x1 = pop(header);
68   if(*header != 0) {
69   if(x1->type == 0) /* id  */ x1->type == 1; /* F */
70   struct token* x2; x2 = pop(header);
71     if(x2->type == 5) /* * */ prod_sub_routine(header, x1, 1);
72     else if(x2->type == 4) /* + */ plus_sub_routine(header, x1, 1);
73     else if(x2->type == 6) { x1->type = 1; push(header, x1); } } }
74 public void eof_routine(struct token** header){
75   struct token* x1; x1 = pop(header); int result = 0;
76   if(*header != 0) {
77     if(x1->type == 0) /* id */ x1->type == 1; /* F */
78     struct token* x2; x2 = pop(header);
79     if(x2->type == 5) /* * */ prod_sub_routine(header, x1, 0);
80     else if(x2->type == 4) /* + */ plus_sub_routine(header, x1, 0);
81     x1 = pop(header);  result = x1->val; smcoutput(result, 1); }
82   else{ result = x1->val; smcoutput(result, 1); /*output the result */ } }
83 public int main() {
84   private int ids[M]; public int expr[S];
85   struct token *header = 0; //header of the stack
86   public int index = 0; public int symbol = 0;
87   smcinput(expr, 1, S); smcinput(ids, 1, M);
88   while(index < S) {
89     symbol = expr[index];
90     if(symbol < K) /* id */ id_routine(&header, ids[symbol]);
91     else if(symbol == K) /* + */ plus_routine(&header);
92     else if(symbol == K+1) /* * */ prod_routine(&header);
93     else if(symbol == K+2) /* ( */ lbra_routine(&header);
94     else if(symbol == K+3) /* ) */ rbra_routine(&header);
95     else if(symbol == K+4) /* EOF */ eof_routine(&header);
96     index = index+1; }
97   return 1;
98 }
```

Figure 6.6: Benchmarking program: LR-parser.c (Part 2/2)

```
1 public int K=1000; /* length of input set */
2 public int main() {
3   public int i; private int year1[K], year2[K]; private int final[K];
4   smcinput(year1, 1, K); smcinput(year2, 1, K);
5   for(i = 0; i < K; i++) { year2[i] = (year2[i] - year1[i]) * 1000; }
6   for(i = 0; i < K; i++) { final[i] = year2[i] / year1[i]; }
7   smcoutput(final, 1, K);
8   return 0;
9 }
```

Figure 6.7: Benchmarking program: h_analysis.c

```
1 public int main() {
2    public int S = 100; private int A[S]; private int B[S];
3    private int c; public int i, j;
4    smcinput(A, 1, S); smcinput(B, 1, S);
5    for (i = 0; i < S; i++) {
6       if (A[i] < B[i]){ c = A[i]; } else{ c = B[i]; } }
7    smcoutput(c, 1);
8    for (i = 0; i < S; i++) {
9       if (A[i] > B[i]){ c = A[i]; } else{ c = B[i]; } }
10   smcoutput(c, 1);
11   for (i = 0; i < S; i++) {
12      if (A[i] < B[i]){ c = B[i] - A[i]; } else{ c = A[i] - B[i]; } }
13   smcoutput(c, 1);
14   for (i = 0; i < S; i++) {
15      if (A[i] > B[i]){ c = A[i] - B[i]; } else{ c = B[i] - A[i]; } }
16   smcoutput(c, 1);
17   for (i = 0; i < 1000; i++) {
18      j = i%100;
19      if (A[j] < B[j]){ c = A[j]; } else{ c = B[j]; } }
20   smcoutput(c, 1);
21   return 0;
22 }
```

Figure 6.8: Benchmarking program: private-branching.c

```
1 public int main() {
2    public int S=100; private int A[S];  private int B[S];
3    private int c; public int i, j;
4    smcinput(A, 1, S); smcinput(B, 1, S);
5    for (i = 0; i < S; i++) {
6       if (A[i] < B[i]){ c = A[i]; c = c + B[i]; c = c * 2; }
7       else{ c = B[i]; c = c + B[i]; c = c + 2; } }
8    smcoutput(c, 1);
9    for (i = 0; i < S; i++) {
10      if (A[i] < B[i]){ c = A[i]; c = c + B[i]; c = c * 2; }
11      else{ c = B[i]; c = c + B[i]; c = c + 2; } }
12   smcoutput(c, 1);
13   for (i = 0; i < S; i++) {
14      if (A[i] < B[i]){ c = A[i]; c = c + B[i]; c = c * 2; }
15      else{ c = B[i]; c = c + B[i]; c = c + 2; } }
16   smcoutput(c, 1);
17   for (i = 0; i < S; i++) {
18      if (A[i] < B[i]){ c = A[i]; c = c + B[i]; c = c * 2; }
19      else{ c = B[i]; c = c + B[i]; c = c + 2; } }
20   smcoutput(c, 1);
21   for (i = 0; i < 1000; i++) {
22      j = i%100;
23      if (A[j] < B[j]){ c = A[j]; c = c + B[j]; c = c * 2; }
24      else{ c = B[j]; c = c + B[j]; c = c + 2; } }
25   smcoutput(c, 1);
26   return 0;
27 }
```

Figure 6.9: Benchmarking program: private-branching-mult.c

```
1  public int main() {
2     public int S=100, i, j;
3     private int A[S], B[S], c, d;
4     smcinput(A, 1, S); smcinput(B, 1, S);
5     for (i = 0; i < S; i++) {
6        if (A[i] < B[i]){
7           c = A[i]; d = c; c = c + B[i]; d = d * c; c = c * 2; d = d + c; }
8        else{
9           c = B[i]; d = c; c = c + B[i]; d = d * c; c = c + 2; d = d + c; } }
10    smcoutput(c, 1); smcoutput(d, 1);
11    for (i = 0; i < S; i++) {
12       if (A[i] < B[i]){
13          c = A[i]; d = c; c = c + B[i]; d = d * c; c = c * 2; d = d + c; }
14       else{
15          c = B[i]; d = c; c = c + B[i]; d = d * c; c = c + 2; d = d + c; } }
16    smcoutput(c, 1); smcoutput(d, 1);
17    for (i = 0; i < S; i++) {
18       if (A[i] < B[i]){
19          c = A[i]; d = c; c = c + B[i]; d = d * c; c = c * 2; d = d + c; }
20       else{
21          c = B[i]; d = c; c = c + B[i]; d = d * c; c = c + 2; d = d + c; } }
22    smcoutput(c, 1); smcoutput(d, 1);
23    for (i = 0; i < S; i++) {
24       if (A[i] < B[i]){
25          c = A[i]; d = c; c = c + B[i]; d = d * c; c = c * 2; d = d + c; }
26       else{
27          c = B[i]; d = c; c = c + B[i]; d = d * c; c = c + 2; d = d + c; } }
28    smcoutput(c, 1); smcoutput(d, 1);
29    for (i = 0; i < 1000; i++) {
30       j = i%100;
31       if (A[j] < B[j]){
32          c = A[j]; d = c; c = c + B[j]; d = d * c; c = c * 2; d = d + c; }
33       else{
34          c = B[j]; d = c; c = c + B[j]; d = d * c; c = c + 2; d = d + c; } }
35    smcoutput(c, 1); smcoutput(d, 1);
36    return 0;
37 }
```

Figure 6.10: Benchmarking program: private-branching-add.c

```
1 public int main() {
2    public int S=100; private int A[S]; private int B[S];
3    private int c=0, d=0, e=0; public int i, j;
4    smcinput(A, 1, S); smcinput(B, 1, S);
5    for (i = 0; i < 100000; i++) {
6       j = i%100;
7       if (A[j] < B[j]){
8          c = c + A[j]; e = e + c; d = d + c; e = e - 2;
9          c = c + B[j]; e = e + d; d = d + c; e = e - c;
10         c = c + 2; e = e - 2; d = d + c; e = e + 10;
11         e = e - 100; e = e + d - c; }
12      else{
13         c = c + B[j]; e = e + c; d = d + c; e = e + d;
14         c = c + B[j]; e = e - 50; d = d + c; e = e + e;
15         c = c + 2; e = e - c - d; d = d + c; e = e + 10;
16         e = e - 100; e = e + d - c; }
17      if(e > 100000){ e = e - 100000; }
18      if(i%50 == 0){ c = 0; d = 0; e = 0; } }
19   smcoutput(c, 1); smcoutput(d, 1); smcoutput(e, 1);
20   return 0;
21 }
```

Figure 6.11: Benchmarking program: private-branching-reuse.c

Local Runtimes

Table 6.3: h_analysis - local PICCO

Run	Party 3	Party 2	Party 1
1	12.6165	12.6182	12.6205
2	12.6407	12.6411	12.6430
3	12.6690	12.6697	12.6712
4	12.6986	12.6992	12.7020
5	12.9585	12.9592	12.9607
6	12.5118	12.5127	12.5150
7	12.5107	12.5113	12.5136
8	12.5615	12.5621	12.5644
9	12.6041	12.6049	12.6066
10	12.5351	12.5357	12.5371
11	12.7150	12.7158	12.7175
12	12.5866	12.5868	12.5885
13	12.6791	12.6796	12.6811
14	12.5861	12.5865	12.5885
15	12.5791	12.5807	12.5815
16	12.5046	12.5051	12.5069
17	12.5390	12.5396	12.5409
18	12.5444	12.5448	12.5462
19	12.5525	12.5532	12.5555
20	12.5229	12.5235	12.5250
21	12.5750	12.5758	12.5778
22	12.5120	12.5126	12.5144
23	12.5044	12.5053	12.5067
24	12.5996	12.6005	12.6030
25	12.5209	12.5218	12.5241
26	12.5379	12.5390	12.5413
27	12.5548	12.5553	12.5582
28	12.5402	12.5409	12.5431
29	12.5705	12.5712	12.5729
30	12.4891	12.4898	12.4921
31	12.5110	12.5116	12.5142
32	12.4431	12.4438	12.4454
33	12.5129	12.5132	12.5156
34	12.5426	12.5432	12.5446
35	12.5670	12.5674	12.5690
36	12.5775	12.5781	12.5805
37	12.5252	12.5259	12.5283
38	12.6392	12.6400	12.6414
39	13.2066	13.2092	13.2113
40	12.5044	12.5050	12.5066
41	12.4948	12.4952	12.4977
42	12.5765	12.5777	12.5789
43	12.5128	12.5135	12.5156
44	12.5350	12.5358	12.5380
45	13.1942	13.1948	13.1970
46	12.5688	12.5692	12.5705
47	12.7549	12.7573	12.7592
48	12.6145	12.6152	12.6166
49	12.5559	12.5566	12.5583
50	12.6109	12.6116	12.6138

Table 6.4: h_analysis - local SMC^2

Run	Party 3	Party 2	Party 1
1	12.8186	12.8203	12.8245
2	12.8695	12.8724	12.8727
3	12.9447	12.9467	12.9501
4	12.7745	12.7764	12.7794
5	12.9614	12.9634	12.9664
6	12.6944	12.6975	12.6977
7	12.6011	12.6032	12.6032
8	12.6476	12.6498	12.6527
9	12.8900	12.8924	12.8958
10	12.9264	12.9273	12.9319
11	13.1416	13.1442	13.1444
12	12.7686	12.7721	12.7708
13	12.8705	12.8730	12.8766
14	12.8357	12.8382	12.8384
15	12.8516	12.8545	12.8553
16	12.7758	12.7791	12.7777
17	12.7853	12.7878	12.7905
18	13.0108	13.0138	13.0141
19	13.4011	13.4038	13.4070
20	13.2590	13.2618	13.2648
21	12.9835	12.9867	12.9871
22	12.5944	12.5972	12.5979
23	12.7999	12.8027	12.8056
24	12.7980	12.8004	12.8031
25	12.6918	12.6953	12.6942
26	12.4757	12.4765	12.4777
27	12.7187	12.7210	12.7243
28	12.5477	12.5487	12.5506
29	13.0753	13.0778	13.0812
30	12.9022	12.9048	12.9051
31	12.5510	12.5516	12.5535
32	13.1789	13.1817	13.1821
33	12.8397	12.8421	12.8449
34	13.2833	13.2850	13.2879
35	12.9896	12.9930	12.9935
36	13.0452	13.0475	13.0502
37	12.7573	12.7599	12.7628
38	12.7752	12.7778	12.7781
39	12.7908	12.7921	12.7955
40	12.5684	12.5689	12.5700
41	13.0630	13.0651	13.0677
42	12.8728	12.8752	12.8781
43	12.6476	12.6506	12.6513
44	12.5682	12.5687	12.5713
45	12.5392	12.5403	12.5426
46	12.4598	12.4604	12.4627
47	12.5042	12.5045	12.5066
48	12.5861	12.5869	12.5893
49	12.5304	12.5316	12.5340
50	12.5667	12.5671	12.5685

Table 6.5: LR-parser - local PICCO

Run	Party 3	Party 2	Party 1
1	$8.540 \cdot 10^{-4}$	$2.316 \cdot 10^{-3}$	$5.012 \cdot 10^{-3}$
2	$7.990 \cdot 10^{-4}$	$1.695 \cdot 10^{-3}$	$2.984 \cdot 10^{-3}$
3	$8.340 \cdot 10^{-4}$	$1.633 \cdot 10^{-3}$	$3.939 \cdot 10^{-3}$
4	$7.750 \cdot 10^{-4}$	$1.176 \cdot 10^{-3}$	$2.742 \cdot 10^{-3}$
5	$8.270 \cdot 10^{-4}$	$1.489 \cdot 10^{-3}$	$3.861 \cdot 10^{-3}$
6	$8.160 \cdot 10^{-4}$	$1.664 \cdot 10^{-3}$	$3.401 \cdot 10^{-3}$
7	$7.920 \cdot 10^{-4}$	$1.086 \cdot 10^{-3}$	$3.703 \cdot 10^{-3}$
8	$7.270 \cdot 10^{-4}$	$1.064 \cdot 10^{-3}$	$3.501 \cdot 10^{-3}$
9	$7.700 \cdot 10^{-4}$	$1.632 \cdot 10^{-3}$	$2.592 \cdot 10^{-3}$
10	$7.880 \cdot 10^{-4}$	$1.344 \cdot 10^{-3}$	$3.129 \cdot 10^{-3}$
11	$8.180 \cdot 10^{-4}$	$1.392 \cdot 10^{-3}$	$3.188 \cdot 10^{-3}$
12	$8.310 \cdot 10^{-4}$	$1.641 \cdot 10^{-3}$	$3.369 \cdot 10^{-3}$
13	$8.900 \cdot 10^{-4}$	$2.542 \cdot 10^{-3}$	$5.069 \cdot 10^{-3}$
14	$8.300 \cdot 10^{-4}$	$1.722 \cdot 10^{-3}$	$3.196 \cdot 10^{-3}$
15	$8.310 \cdot 10^{-4}$	$1.463 \cdot 10^{-3}$	$3.075 \cdot 10^{-3}$
16	$9.180 \cdot 10^{-4}$	$2.500 \cdot 10^{-3}$	$2.920 \cdot 10^{-3}$
17	$8.710 \cdot 10^{-4}$	$2.389 \cdot 10^{-3}$	$5.403 \cdot 10^{-3}$
18	$8.190 \cdot 10^{-4}$	$1.290 \cdot 10^{-3}$	$3.023 \cdot 10^{-3}$
19	$8.210 \cdot 10^{-4}$	$1.494 \cdot 10^{-3}$	$3.746 \cdot 10^{-3}$
20	$9.120 \cdot 10^{-4}$	$2.459 \cdot 10^{-3}$	$3.413 \cdot 10^{-3}$
21	$8.180 \cdot 10^{-4}$	$1.555 \cdot 10^{-3}$	$3.850 \cdot 10^{-3}$
22	$7.640 \cdot 10^{-4}$	$1.487 \cdot 10^{-3}$	$3.134 \cdot 10^{-3}$
23	$8.050 \cdot 10^{-4}$	$1.256 \cdot 10^{-3}$	$2.739 \cdot 10^{-3}$
24	$7.890 \cdot 10^{-4}$	$1.390 \cdot 10^{-3}$	$2.839 \cdot 10^{-3}$
25	$8.060 \cdot 10^{-4}$	$1.434 \cdot 10^{-3}$	$2.744 \cdot 10^{-3}$
26	$1.572 \cdot 10^{-3}$	$2.111 \cdot 10^{-3}$	$4.370 \cdot 10^{-3}$
27	$9.490 \cdot 10^{-4}$	$1.740 \cdot 10^{-3}$	$6.200 \cdot 10^{-3}$
28	$9.020 \cdot 10^{-4}$	$2.551 \cdot 10^{-3}$	$3.305 \cdot 10^{-3}$
29	$9.010 \cdot 10^{-4}$	$2.073 \cdot 10^{-3}$	$4.935 \cdot 10^{-3}$
30	$8.860 \cdot 10^{-4}$	$2.610 \cdot 10^{-3}$	$5.219 \cdot 10^{-3}$
31	$8.990 \cdot 10^{-4}$	$1.662 \cdot 10^{-3}$	$5.993 \cdot 10^{-3}$
32	$9.480 \cdot 10^{-4}$	$2.482 \cdot 10^{-3}$	$3.348 \cdot 10^{-3}$
33	$9.610 \cdot 10^{-4}$	$2.432 \cdot 10^{-3}$	$3.020 \cdot 10^{-3}$
34	$7.910 \cdot 10^{-4}$	$1.508 \cdot 10^{-3}$	$3.285 \cdot 10^{-3}$
35	$9.160 \cdot 10^{-4}$	$2.268 \cdot 10^{-3}$	$5.130 \cdot 10^{-3}$
36	$1.021 \cdot 10^{-3}$	$3.897 \cdot 10^{-3}$	$2.418 \cdot 10^{-3}$
37	$9.020 \cdot 10^{-4}$	$2.550 \cdot 10^{-3}$	$4.612 \cdot 10^{-3}$
38	$8.870 \cdot 10^{-4}$	$2.491 \cdot 10^{-3}$	$5.528 \cdot 10^{-3}$
39	$8.870 \cdot 10^{-4}$	$2.556 \cdot 10^{-3}$	$5.101 \cdot 10^{-3}$
40	$9.790 \cdot 10^{-4}$	$2.485 \cdot 10^{-3}$	$3.595 \cdot 10^{-3}$
41	$1.032 \cdot 10^{-3}$	$2.776 \cdot 10^{-3}$	$5.811 \cdot 10^{-3}$
42	$9.460 \cdot 10^{-4}$	$3.891 \cdot 10^{-3}$	$2.379 \cdot 10^{-3}$
43	$8.000 \cdot 10^{-4}$	$1.312 \cdot 10^{-3}$	$2.932 \cdot 10^{-3}$
44	$9.200 \cdot 10^{-4}$	$2.252 \cdot 10^{-3}$	$4.529 \cdot 10^{-3}$
45	$9.120 \cdot 10^{-4}$	$2.482 \cdot 10^{-3}$	$5.177 \cdot 10^{-3}$
46	$8.940 \cdot 10^{-4}$	$2.409 \cdot 10^{-3}$	$3.015 \cdot 10^{-3}$
47	$8.930 \cdot 10^{-4}$	$2.597 \cdot 10^{-3}$	$3.109 \cdot 10^{-3}$
48	$9.630 \cdot 10^{-4}$	$2.347 \cdot 10^{-3}$	$2.817 \cdot 10^{-3}$
49	$1.014 \cdot 10^{-3}$	$2.666 \cdot 10^{-3}$	$6.002 \cdot 10^{-3}$
50	$9.220 \cdot 10^{-4}$	$3.894 \cdot 10^{-3}$	$2.536 \cdot 10^{-3}$

Table 6.6: LR-parser - local SMC^2

Run	Party 3	Party 2	Party 1
1	$8.710 \cdot 10^{-4}$	$1.145 \cdot 10^{-3}$	$3.156 \cdot 10^{-3}$
2	$9.390 \cdot 10^{-4}$	$2.686 \cdot 10^{-3}$	$3.339 \cdot 10^{-3}$
3	$8.770 \cdot 10^{-4}$	$2.452 \cdot 10^{-3}$	$5.342 \cdot 10^{-3}$
4	$8.840 \cdot 10^{-4}$	$2.273 \cdot 10^{-3}$	$2.630 \cdot 10^{-3}$
5	$9.010 \cdot 10^{-4}$	$2.452 \cdot 10^{-3}$	$5.315 \cdot 10^{-3}$
6	$1.003 \cdot 10^{-3}$	$3.679 \cdot 10^{-3}$	$2.311 \cdot 10^{-3}$
7	$9.090 \cdot 10^{-4}$	$2.376 \cdot 10^{-3}$	$5.328 \cdot 10^{-3}$
8	$9.480 \cdot 10^{-4}$	$2.487 \cdot 10^{-3}$	$2.941 \cdot 10^{-3}$
9	$8.990 \cdot 10^{-4}$	$2.349 \cdot 10^{-3}$	$3.401 \cdot 10^{-3}$
10	$1.225 \cdot 10^{-3}$	$4.193 \cdot 10^{-3}$	$2.377 \cdot 10^{-3}$
11	$8.880 \cdot 10^{-4}$	$2.421 \cdot 10^{-3}$	$4.982 \cdot 10^{-3}$
12	$9.070 \cdot 10^{-4}$	$1.575 \cdot 10^{-3}$	$5.926 \cdot 10^{-3}$
13	$9.020 \cdot 10^{-4}$	$2.340 \cdot 10^{-3}$	$5.113 \cdot 10^{-3}$
14	$9.490 \cdot 10^{-4}$	$2.139 \cdot 10^{-3}$	$5.352 \cdot 10^{-3}$
15	$9.290 \cdot 10^{-4}$	$2.621 \cdot 10^{-3}$	$3.371 \cdot 10^{-3}$
16	$8.900 \cdot 10^{-4}$	$2.363 \cdot 10^{-3}$	$6.000 \cdot 10^{-3}$
17	$8.820 \cdot 10^{-4}$	$2.386 \cdot 10^{-3}$	$5.253 \cdot 10^{-3}$
18	$9.130 \cdot 10^{-4}$	$3.613 \cdot 10^{-3}$	$2.008 \cdot 10^{-3}$
19	$9.780 \cdot 10^{-4}$	$2.622 \cdot 10^{-3}$	$5.543 \cdot 10^{-3}$
20	$8.430 \cdot 10^{-4}$	$2.472 \cdot 10^{-3}$	$3.559 \cdot 10^{-3}$
21	$8.530 \cdot 10^{-4}$	$1.528 \cdot 10^{-3}$	$5.406 \cdot 10^{-3}$
22	$9.080 \cdot 10^{-4}$	$2.544 \cdot 10^{-3}$	$5.216 \cdot 10^{-3}$
23	$9.140 \cdot 10^{-4}$	$3.811 \cdot 10^{-3}$	$2.401 \cdot 10^{-3}$
24	$8.450 \cdot 10^{-4}$	$1.225 \cdot 10^{-3}$	$3.310 \cdot 10^{-3}$
25	$9.780 \cdot 10^{-4}$	$2.569 \cdot 10^{-3}$	$2.828 \cdot 10^{-3}$
26	$8.310 \cdot 10^{-4}$	$2.000 \cdot 10^{-3}$	$3.705 \cdot 10^{-3}$
27	$7.830 \cdot 10^{-4}$	$1.690 \cdot 10^{-3}$	$2.561 \cdot 10^{-3}$
28	$7.530 \cdot 10^{-4}$	$1.155 \cdot 10^{-3}$	$3.258 \cdot 10^{-3}$
29	$8.120 \cdot 10^{-4}$	$1.665 \cdot 10^{-3}$	$3.065 \cdot 10^{-3}$
30	$7.800 \cdot 10^{-4}$	$1.332 \cdot 10^{-3}$	$3.652 \cdot 10^{-3}$
31	$7.860 \cdot 10^{-4}$	$1.475 \cdot 10^{-3}$	$3.993 \cdot 10^{-3}$
32	$7.710 \cdot 10^{-4}$	$1.214 \cdot 10^{-3}$	$2.527 \cdot 10^{-3}$
33	$8.320 \cdot 10^{-4}$	$1.611 \cdot 10^{-3}$	$3.107 \cdot 10^{-3}$
34	$8.190 \cdot 10^{-4}$	$1.365 \cdot 10^{-3}$	$3.329 \cdot 10^{-3}$
35	$8.440 \cdot 10^{-4}$	$1.510 \cdot 10^{-3}$	$4.705 \cdot 10^{-3}$
36	$8.780 \cdot 10^{-4}$	$1.842 \cdot 10^{-3}$	$3.954 \cdot 10^{-3}$
37	$8.510 \cdot 10^{-4}$	$1.525 \cdot 10^{-3}$	$3.250 \cdot 10^{-3}$
38	$8.990 \cdot 10^{-4}$	$2.663 \cdot 10^{-3}$	$4.528 \cdot 10^{-3}$
39	$7.820 \cdot 10^{-4}$	$8.510 \cdot 10^{-4}$	$3.009 \cdot 10^{-3}$
40	$7.990 \cdot 10^{-4}$	$1.503 \cdot 10^{-3}$	$3.539 \cdot 10^{-3}$
41	$8.420 \cdot 10^{-4}$	$1.464 \cdot 10^{-3}$	$3.614 \cdot 10^{-3}$
42	$8.210 \cdot 10^{-4}$	$1.842 \cdot 10^{-3}$	$3.921 \cdot 10^{-3}$
43	$7.430 \cdot 10^{-4}$	$1.336 \cdot 10^{-3}$	$3.164 \cdot 10^{-3}$
44	$8.520 \cdot 10^{-4}$	$1.627 \cdot 10^{-3}$	$2.923 \cdot 10^{-3}$
45	$8.290 \cdot 10^{-4}$	$1.585 \cdot 10^{-3}$	$3.940 \cdot 10^{-3}$
46	$7.950 \cdot 10^{-4}$	$1.371 \cdot 10^{-3}$	$2.803 \cdot 10^{-3}$
47	$8.230 \cdot 10^{-4}$	$1.489 \cdot 10^{-3}$	$3.662 \cdot 10^{-3}$
48	$8.200 \cdot 10^{-4}$	$1.478 \cdot 10^{-3}$	$2.972 \cdot 10^{-3}$
49	$7.570 \cdot 10^{-4}$	$1.175 \cdot 10^{-3}$	$3.160 \cdot 10^{-3}$
50	$7.500 \cdot 10^{-4}$	$1.280 \cdot 10^{-3}$	$3.948 \cdot 10^{-3}$

Table 6.7: pay-gap - local PICCO

Run	Party 3	Party 2	Party 1
1	4.16726	4.16811	4.16981
2	4.03592	4.03685	4.03826
3	4.00866	4.00958	4.01208
4	4.10085	4.10164	4.10376
5	4.04085	4.04344	4.04604
6	3.98020	3.98091	3.98327
7	4.01444	4.01522	4.01747
8	4.29332	4.29734	4.29578
9	4.05769	4.05859	4.06192
10	4.02625	4.02711	4.02931
11	3.99417	3.99530	4.00207
12	4.04705	4.04803	4.04939
13	4.12716	4.12822	4.13046
14	3.98831	3.98895	3.99048
15	4.00110	4.00219	4.00437
16	4.03571	4.03719	4.03813
17	4.25825	4.25555	4.26014
18	4.01612	4.01703	4.01820
19	4.21264	4.21373	4.21821
20	4.02595	4.02700	4.02937
21	4.17542	4.17704	4.18179
22	4.01560	4.01614	4.01937
23	3.98058	3.98066	3.98402
24	4.00213	4.00046	4.00343
25	4.00142	4.00205	4.00435
26	4.20610	4.20698	4.20958
27	4.12068	4.12291	4.12347
28	4.10280	4.10058	4.10424
29	4.01360	4.01449	4.01568
30	3.96217	3.96325	3.96492
31	4.03994	4.04055	4.04318
32	4.00828	4.00980	4.01079
33	3.97393	3.97476	3.97659
34	4.02941	4.03030	4.03258
35	3.98304	3.98376	3.98556
36	4.00000	4.00049	4.00202
37	3.98944	3.99020	3.99271
38	4.17173	4.17255	4.17467
39	4.13375	4.13451	4.13671
40	3.98662	3.98736	3.98963
41	4.02506	4.02599	4.02843
42	3.98928	3.99069	3.99465
43	4.19137	4.19228	4.19389
44	4.04239	4.04339	4.04532
45	3.98701	3.98745	3.98949
46	3.98489	3.98588	3.99115
47	3.99005	3.99224	3.99289
48	4.04459	4.04541	4.04779
49	4.16325	4.16382	4.16617
50	4.00166	4.00349	4.00453

Table 6.8: pay-gap - local SMC^2

Run	Party 3	Party 2	Party 1
1	4.02504	4.02626	4.02898
2	4.00674	4.00838	4.01023
3	4.00143	4.00328	4.00424
4	4.20375	4.20519	4.20611
5	4.01353	4.01505	4.01574
6	4.17787	4.17887	4.18324
7	3.98043	3.98128	3.98279
8	4.18946	4.19008	4.19118
9	4.11967	4.12053	4.12294
10	4.02879	4.02999	4.03219
11	4.47848	4.48203	4.48278
12	4.01288	4.01387	4.01608
13	3.97971	3.98042	3.98265
14	4.19594	4.19803	4.19890
15	4.03561	4.03655	4.03919
16	3.98955	3.98980	3.99255
17	4.00117	4.00187	4.00328
18	3.98528	3.98599	3.98754
19	4.25372	4.25549	4.25623
20	3.97173	3.97263	3.97480
21	3.99446	3.99529	3.99759
22	4.02002	4.02076	4.02276
23	3.98496	3.98674	3.98779
24	4.00550	4.00646	4.00781
25	4.55908	4.56078	4.56227
26	3.97448	3.97510	3.97744
27	4.49137	4.49360	4.49689
28	4.12475	4.12648	4.12932
29	3.98664	3.98840	3.98931
30	3.98306	3.98355	3.98665
31	3.99172	3.99308	3.99364
32	4.27796	4.27945	4.27989
33	4.14147	4.14232	4.14440
34	4.20099	4.20122	4.20419
35	4.02764	4.02862	4.03097
36	4.01179	4.01307	4.01597
37	3.98353	3.98402	3.98563
38	4.02764	4.02832	4.03072
39	3.98398	3.98509	3.98716
40	3.99998	4.00196	4.00284
41	4.10665	4.10772	4.11041
42	4.18827	4.18885	4.19138
43	3.97756	3.97819	3.98022
44	3.98376	3.98494	3.98699
45	4.11079	4.11221	4.11315
46	4.03477	4.03561	4.03757
47	4.22173	4.22256	4.22479
48	4.15755	4.15995	4.16039
49	4.16673	4.16786	4.16921
50	3.99967	4.00050	4.00172

Table 6.9: private-branching - local PICCO

Run	Party 3	Party 2	Party 1
1	1.60458	1.60673	1.60834
2	1.65330	1.65421	1.65680
3	1.60857	1.60919	1.61080
4	1.59710	1.59767	1.59945
5	1.61263	1.61330	1.61579
6	1.62257	1.62348	1.62550
7	1.61156	1.61220	1.61395
8	1.59742	1.59801	1.60061
9	1.60492	1.60566	1.60717
10	1.60842	1.60911	1.61165
11	1.59461	1.59533	1.59769
12	2.50973	2.51158	2.51490
13	1.58400	1.58452	1.58694
14	1.60348	1.60428	1.60652
15	1.76747	1.76820	1.76988
16	1.96118	1.96397	1.96693
17	1.59610	1.59674	1.59823
18	1.59450	1.59513	1.59753
19	1.64438	1.64514	1.65158
20	1.66879	1.66941	1.67101
21	1.61062	1.61118	1.61233
22	2.04844	2.05102	2.05405
23	1.61313	1.61384	1.61556
24	1.59406	1.59463	1.59695
25	1.59437	1.59497	1.59651
26	1.61402	1.61473	1.61719
27	1.60158	1.60215	1.60342
28	1.60954	1.61027	1.61187
29	1.60095	1.60162	1.60396
30	1.83951	1.84268	1.84503
31	1.56627	1.56726	1.56862
32	1.60700	1.60719	1.61004
33	1.61720	1.61777	1.61957
34	1.84574	1.84823	1.85089
35	1.58155	1.58230	1.58442
36	1.59460	1.59521	1.59740
37	1.59218	1.59294	1.59525
38	1.59741	1.59851	1.59918
39	1.60366	1.60427	1.60685
40	1.92327	1.92631	1.92846
41	1.59204	1.59228	1.59454
42	1.94828	1.95088	1.95365
43	1.66567	1.66617	1.66852
44	1.61146	1.61214	1.61398
45	1.63584	1.63656	1.63896
46	1.59177	1.59239	1.59357
47	1.60450	1.60493	1.60650
48	1.79401	1.79672	1.79906
49	1.60696	1.60738	1.60991
50	1.60545	1.60631	1.60769

Table 6.10: private-branching - local SMC^2

Run	Party 3	Party 2	Party 1
1	1.54662	1.54731	1.54858
2	1.54512	1.54577	1.54682
3	1.55057	1.55120	1.55271
4	1.52629	1.52701	1.52838
5	1.53156	1.53222	1.53410
6	1.55063	1.55140	1.55305
7	1.55285	1.55350	1.55526
8	1.55436	1.55503	1.55730
9	1.87710	1.87850	1.88182
10	1.51997	1.52075	1.52320
11	1.59211	1.59217	1.59526
12	1.53990	1.54064	1.54209
13	1.53930	1.53956	1.54156
14	1.60499	1.60590	1.60759
15	1.55078	1.55152	1.55306
16	1.53605	1.53667	1.53897
17	1.54284	1.54336	1.54498
18	1.84004	1.84267	1.84534
19	1.53181	1.53255	1.53451
20	1.53991	1.54024	1.54194
21	1.54858	1.54927	1.55104
22	1.72932	1.73221	1.73412
23	1.53651	1.53700	1.53847
24	1.53265	1.53333	1.53584
25	1.55363	1.55451	1.55675
26	1.55277	1.55355	1.55581
27	1.55109	1.55155	1.55298
28	1.53933	1.53991	1.54191
29	1.54686	1.54776	1.55025
30	1.76529	1.76749	1.77031
31	1.52424	1.52501	1.52687
32	1.53305	1.53401	1.53590
33	1.53941	1.53998	1.54147
34	1.54670	1.54761	1.54911
35	1.53822	1.53879	1.54017
36	1.53811	1.53882	1.54105
37	1.53320	1.53388	1.53615
38	1.54169	1.54203	1.54460
39	1.53514	1.53569	1.53741
40	1.52526	1.52616	1.52859
41	1.55529	1.55616	1.55830
42	1.52545	1.52637	1.52794
43	1.54622	1.54679	1.54858
44	1.55675	1.55736	1.55886
45	1.54968	1.55074	1.55217
46	1.88024	1.88338	1.88447
47	1.53542	1.53611	1.53888
48	1.72987	1.73224	1.73470
49	1.53644	1.53710	1.53974
50	1.53938	1.53983	1.54230

Table 6.11: private-branching-mult - local PICCO

Run	Party 3	Party 2	Party 1
1	1.84452	1.84614	1.84859
2	1.83161	1.83239	1.83416
3	1.82864	1.82927	1.83088
4	1.83207	1.83290	1.83461
5	1.82922	1.83012	1.83177
6	2.17623	2.17756	2.18199
7	1.81076	1.81217	1.81256
8	2.18860	2.19121	2.19377
9	1.82277	1.82319	1.82446
10	1.82890	1.82947	1.83144
11	1.84167	1.84242	1.84360
12	1.82250	1.82295	1.82549
13	1.83336	1.83418	1.83658
14	1.83057	1.83099	1.83256
15	1.83686	1.83733	1.83899
16	1.85853	1.85929	1.86167
17	1.81368	1.81429	1.81695
18	2.07964	2.08242	2.08509
19	1.81677	1.81699	1.81945
20	2.48197	2.48377	2.48751
21	1.82953	1.83019	1.83176
22	1.83899	1.83977	1.84206
23	1.81361	1.81446	1.81593
24	1.81658	1.81801	1.81881
25	1.81737	1.81791	1.82040
26	1.99169	1.99413	1.99673
27	1.78829	1.78899	1.79113
28	1.82606	1.82748	1.82819
29	1.81855	1.81883	1.82111
30	1.82258	1.82325	1.82468
31	1.82914	1.82949	1.83157
32	2.52867	2.53119	2.53375
33	1.81997	1.82122	1.82293
34	1.82738	1.82803	1.83074
35	1.81170	1.81202	1.81471
36	1.92061	1.92316	1.92566
37	1.80583	1.80639	1.80898
38	2.13714	2.14014	2.14237
39	1.81543	1.81637	1.81789
40	1.82036	1.82112	1.82356
41	1.82171	1.82248	1.82504
42	2.16868	2.16983	2.17338
43	1.80742	1.80765	1.81031
44	2.27346	2.27592	2.27898
45	1.81042	1.81120	1.81293
46	1.81737	1.81804	1.82065
47	1.83220	1.83295	1.83462
48	1.80715	1.80719	1.80937
49	1.82041	1.82082	1.82340
50	1.84488	1.84581	1.84804

Table 6.12: private-branching-mult - local SMC^2

Run	Party 3	Party 2	Party 1
1	1.54888	1.55031	1.55182
2	1.53811	1.53893	1.54046
3	1.53257	1.53305	1.53573
4	1.54548	1.54625	1.54842
5	1.54048	1.54082	1.54309
6	1.53612	1.53677	1.53841
7	1.52219	1.52293	1.52515
8	1.90041	1.90295	1.90517
9	1.54033	1.54103	1.54261
10	1.59069	1.59132	1.59357
11	1.77185	1.77268	1.77516
12	2.08521	2.08763	2.09016
13	1.53254	1.53304	1.53565
14	1.64102	1.64349	1.64630
15	1.52872	1.52959	1.53131
16	1.54642	1.54720	1.54897
17	1.54477	1.54589	1.54836
18	1.80777	1.81025	1.81294
19	1.52321	1.52405	1.52658
20	1.52073	1.52143	1.52325
21	1.53302	1.53371	1.53533
22	1.53418	1.53491	1.53636
23	1.53334	1.53404	1.53558
24	1.53236	1.53317	1.53466
25	1.53389	1.53415	1.53668
26	1.52296	1.52361	1.52610
27	1.54760	1.54816	1.54963
28	1.55751	1.55837	1.56068
29	1.54656	1.54736	1.54994
30	1.69967	1.70265	1.70360
31	1.53887	1.53953	1.54117
32	1.67955	1.68211	1.68450
33	1.55447	1.55541	1.55708
34	1.53531	1.53588	1.53827
35	1.71574	1.71853	1.71930
36	1.55839	1.56080	1.56344
37	1.52407	1.52591	1.52682
38	1.52493	1.52541	1.52702
39	1.61137	1.61196	1.61380
40	1.72207	1.72284	1.72529
41	1.56052	1.56134	1.56352
42	1.81293	1.81597	1.81702
43	1.97124	1.97382	1.97671
44	1.78799	1.79075	1.79322
45	1.52358	1.52413	1.52567
46	1.55960	1.56031	1.56216
47	1.54804	1.54877	1.55009
48	2.02455	2.02757	2.02838
49	1.53037	1.53106	1.53258
50	1.65964	1.66256	1.66333

Table 6.13: private-branching-add - local PICCO

Run	Party 3	Party 2	Party 1
1	2.26137	2.26344	2.26575
2	2.25126	2.25196	2.25415
3	2.26869	2.26954	2.27113
4	2.37157	2.37416	2.37640
5	2.25388	2.25427	2.25700
6	2.26388	2.26453	2.26589
7	2.24790	2.24844	2.25111
8	2.44936	2.45201	2.45472
9	2.26814	2.26872	2.27139
10	2.28338	2.28404	2.28663
11	2.29533	2.29539	2.29759
12	2.25524	2.25630	2.25900
13	2.27186	2.27259	2.27411
14	2.25764	2.25817	2.26003
15	2.24657	2.24718	2.24961
16	2.52695	2.52984	2.53208
17	2.28396	2.28441	2.28616
18	2.26472	2.26531	2.26676
19	2.28065	2.28115	2.28294
20	2.41586	2.41889	2.41946
21	2.23359	2.23398	2.23586
22	2.46324	2.46554	2.46761
23	2.25360	2.25417	2.25589
24	2.25771	2.25838	2.26094
25	2.28734	2.28779	2.28942
26	2.23570	2.23646	2.23913
27	2.34608	2.34681	2.34867
28	2.24490	2.24554	2.24794
29	2.25011	2.25080	2.25272
30	2.25984	2.26068	2.26289
31	2.34647	2.34694	2.34923
32	2.32354	2.32579	2.32836
33	2.26515	2.26598	2.26776
34	2.25612	2.25678	2.25834
35	2.24158	2.24220	2.24450
36	2.61099	2.61364	2.61595
37	2.28469	2.28557	2.28706
38	2.38805	2.39068	2.39333
39	2.22541	2.22597	2.22799
40	2.51652	2.51919	2.52146
41	2.25918	2.26020	2.26283
42	2.25602	2.25717	2.25856
43	2.25852	2.25908	2.26142
44	2.26348	2.26391	2.26521
45	2.25589	2.25649	2.25918
46	2.34781	2.35000	2.35257
47	2.25329	2.25419	2.25654
48	2.25370	2.25424	2.25660
49	2.25657	2.25756	2.25943
50	2.57972	2.58237	2.58505

Table 6.14: private-branching-add - local SMC^2

Run	Party 3	Party 2	Party 1
1	1.71235	1.71382	1.71662
2	1.72306	1.72404	1.72604
3	1.71693	1.71758	1.72048
4	1.84594	1.84914	1.85015
5	1.69609	1.69681	1.69823
6	1.71023	1.71102	1.71333
7	1.69963	1.70073	1.70256
8	1.81254	1.81504	1.81743
9	1.69577	1.69650	1.69884
10	1.99570	1.99863	1.99932
11	1.71234	1.71335	1.71481
12	1.70743	1.70784	1.71001
13	1.70228	1.70313	1.70442
14	1.70913	1.70986	1.71147
15	1.70737	1.70801	1.70997
16	2.02438	2.02725	2.02912
17	1.69039	1.69109	1.69275
18	1.69969	1.70007	1.70276
19	1.71531	1.71623	1.71845
20	2.27585	2.27866	2.28155
21	1.71745	1.71798	1.71927
22	1.93455	1.93817	1.93695
23	1.70552	1.70621	1.70856
24	1.70485	1.70570	1.70729
25	1.70303	1.70357	1.70530
26	1.87633	1.87756	1.88020
27	1.71121	1.71202	1.71425
28	2.28842	2.29075	2.29354
29	1.98787	1.99013	1.99183
30	1.71027	1.71093	1.71340
31	1.69438	1.69494	1.69622
32	1.99784	2.00022	2.00298
33	1.69397	1.69459	1.69580
34	1.70427	1.70499	1.70739
35	1.70912	1.70961	1.71124
36	1.73849	1.73924	1.74060
37	1.71039	1.71123	1.71276
38	1.88031	1.88292	1.88553
39	1.69504	1.69572	1.69726
40	1.69340	1.69383	1.69548
41	1.71090	1.71165	1.71303
42	1.69498	1.69654	1.69744
43	1.71914	1.71990	1.72149
44	1.73577	1.73649	1.73809
45	1.79852	1.79906	1.80076
46	1.71324	1.71384	1.71557
47	1.70883	1.70936	1.71105
48	1.71332	1.71389	1.71615
49	1.71390	1.71435	1.71633
50	1.73054	1.73090	1.73261

Table 6.15: private-branching-reuse - local PICCO

Run	Party 3	Party 2	Party 1
1	306.417	306.417	306.420
2	306.770	306.770	306.773
3	307.605	307.605	307.608
4	304.854	304.854	304.856
5	305.650	305.651	305.653
6	313.356	313.356	313.359
7	307.517	307.517	307.519
8	306.909	306.910	306.912
9	308.475	308.477	308.478
10	307.584	307.585	307.588
11	308.927	308.928	308.930
12	305.307	305.308	305.310
13	307.220	307.221	307.224
14	307.052	307.053	307.055
15	306.521	306.522	306.524
16	313.535	313.536	313.536
17	309.687	309.687	309.690
18	309.937	309.937	309.939
19	312.982	312.984	312.985
20	309.028	309.027	309.029
21	307.059	307.059	307.061
22	306.067	306.068	306.069
23	309.403	309.404	309.406
24	308.003	308.004	308.006
25	311.320	311.320	311.323
26	308.300	308.301	308.303
27	307.684	307.685	307.688
28	311.649	311.652	311.655
29	304.103	304.104	304.106
30	307.001	307.001	307.003
31	307.276	307.277	307.279
32	306.671	306.671	306.674
33	306.545	306.546	306.548
34	305.210	305.211	305.213
35	306.935	306.936	306.939
36	306.058	306.058	306.060
37	305.447	305.447	305.449
38	309.230	309.230	309.233
39	305.702	305.703	305.705
40	309.179	309.180	309.182
41	308.917	308.917	308.919
42	306.335	306.335	306.337
43	306.904	306.906	306.906
44	305.180	305.181	305.184
45	308.632	308.633	308.634
46	305.768	305.769	305.771
47	310.487	310.488	310.490
48	306.675	306.675	306.678
49	307.250	307.251	307.254
50	305.823	305.824	305.826

Table 6.16: private-branching-reuse - local SMC^2

Run	Party 3	Party 2	Party 1
1	207.435	207.436	207.438
2	208.437	208.438	208.442
3	207.562	207.562	207.564
4	208.017	208.018	208.020
5	207.633	207.635	207.636
6	208.643	208.645	208.646
7	206.995	206.995	206.997
8	208.240	208.241	208.243
9	207.078	207.078	207.081
10	208.841	208.842	208.843
11	206.712	206.713	206.714
12	209.619	209.619	209.621
13	207.887	207.887	207.890
14	208.898	208.898	208.901
15	208.852	208.853	208.855
16	209.493	209.494	209.496
17	207.297	207.298	207.299
18	206.614	206.615	206.617
19	209.137	209.138	209.140
20	206.691	206.691	206.693
21	207.515	207.516	207.517
22	210.254	210.255	210.257
23	208.206	208.207	208.209
24	206.548	206.548	206.551
25	207.497	207.498	207.500
26	208.610	208.611	208.614
27	207.952	207.953	207.955
28	206.238	206.239	206.240
29	207.875	207.876	207.878
30	208.124	208.125	208.127
31	206.764	206.765	206.767
32	213.974	213.976	213.977
33	207.685	207.686	207.688
34	207.504	207.505	207.506
35	207.304	207.305	207.306
36	206.047	206.047	206.050
37	206.517	206.518	206.519
38	208.409	208.410	208.411
39	209.372	209.372	209.375
40	207.550	207.550	207.553
41	209.026	209.027	209.029
42	207.617	207.617	207.619
43	208.531	208.532	208.533
44	206.267	206.267	206.270
45	207.775	207.775	207.778
46	206.919	206.922	206.923
47	208.300	208.301	208.303
48	209.102	209.103	209.105
49	208.143	208.145	208.148
50	207.930	207.931	207.932

Table 6.17: h_analysis - distributed PICCO

Run	Party 3	Party 2	Party 1
1	32.9724	32.9741	32.9740
2	33.0030	33.0038	33.0038
3	33.1318	33.1323	33.1327
4	33.0414	33.0425	33.0429
5	33.1214	33.1224	33.1228
6	33.0819	33.0831	33.0846
7	33.2351	33.2362	33.2367
8	32.6875	32.6882	32.6885
9	33.0159	33.0171	33.0174
10	33.3818	33.3826	33.3836
11	33.4762	33.4771	33.4776
12	33.0908	33.0904	33.0908
13	32.9965	32.9971	32.9977
14	33.4771	33.4784	33.4783
15	33.2027	33.2043	33.2043
16	33.1458	33.1465	33.1469
17	33.0859	33.0870	33.0874
18	33.2989	33.2999	33.2999
19	33.3585	33.3594	33.3601
20	33.4771	33.4781	33.4782
21	32.5989	32.5988	32.5992
22	33.2016	33.2024	33.2031
23	33.4653	33.4665	33.4667
24	33.4956	33.4970	33.4975
25	33.5036	33.5048	33.5059
26	33.5028	33.5037	33.5039
27	33.3172	33.3185	33.3193
28	32.8595	32.8605	32.8608
29	33.2181	33.2191	33.2192
30	32.8554	32.8566	32.8564
31	33.5673	33.5682	33.5691
32	32.6680	32.6691	32.6698
33	33.3845	33.3856	33.3862
34	33.4602	33.4609	33.4610
35	33.3772	33.3784	33.3791
36	32.8090	32.8104	32.8110
37	33.5681	33.5695	33.5702
38	32.9796	32.9805	32.9809
39	32.9324	32.9333	32.9336
40	33.0995	33.1009	33.1015
41	33.4864	33.4872	33.4876
42	33.0941	33.0950	33.0954
43	33.4355	33.4369	33.4379
44	32.7640	32.7646	32.7649
45	32.9461	32.9474	32.9473
46	33.1884	33.1895	33.1900
47	33.3549	33.3554	33.3559
48	33.1047	33.1055	33.1057
49	33.4986	33.4993	33.5002
50	32.9027	32.9039	32.9047

Table 6.18: h_analysis - distributed SMC^2

Run	Party 3	Party 2	Party 1
1	33.3860	33.3859	33.3858
2	33.4402	33.4409	33.4410
3	33.0142	33.0152	33.0157
4	33.3040	33.3052	33.3056
5	33.4974	33.4984	33.4991
6	32.7267	32.7272	32.7281
7	33.2915	33.2938	33.2930
8	33.2545	33.2560	33.2564
9	33.3282	33.3296	33.3298
10	32.4965	32.4973	32.5000
11	33.2348	33.2357	33.2357
12	32.4525	32.4532	32.4534
13	32.5411	32.5419	32.5428
14	33.2799	33.2807	33.2825
15	33.2421	33.2439	33.2444
16	33.2257	33.2272	33.2276
17	33.6459	33.6471	33.6472
18	32.4917	32.4932	32.4936
19	32.8390	32.8403	32.8407
20	33.2840	33.2849	33.2864
21	33.3813	33.3825	33.3828
22	33.2197	33.2208	33.2220
23	32.8390	32.8383	32.8382
24	33.3357	33.3365	33.3374
25	33.2647	33.2659	33.2659
26	33.4063	33.4057	33.4060
27	32.8752	32.8760	32.8760
28	33.3966	33.3981	33.3980
29	33.5832	33.5847	33.5851
30	33.3178	33.3189	33.3192
31	33.3770	33.3777	33.3783
32	33.2906	33.2918	33.2921
33	33.0039	33.0044	33.0059
34	33.4027	33.4044	33.4042
35	33.5612	33.5625	33.5626
36	33.3462	33.3479	33.3485
37	33.2213	33.2239	33.2245
38	32.7314	32.7322	32.7333
39	33.5250	33.5262	33.5270
40	33.0647	33.0660	33.0659
41	33.4969	33.4981	33.4987
42	33.6321	33.6330	33.6344
43	32.9909	32.9921	32.9929
44	33.2335	33.2346	33.2348
45	33.0860	33.0868	33.0877
46	32.9811	32.9821	32.9822
47	33.0257	33.0263	33.0269
48	33.1959	33.1973	33.1965
49	32.7920	32.7926	32.7939
50	32.4894	32.4902	32.4912

Table 6.19: LR-parser - distributed PICCO

Run	Party 3	Party 2	Party 1
1	$1.549 \cdot 10^{-3}$	$3.097 \cdot 10^{-3}$	$3.742 \cdot 10^{-3}$
2	$1.574 \cdot 10^{-3}$	$2.668 \cdot 10^{-3}$	$2.459 \cdot 10^{-3}$
3	$1.594 \cdot 10^{-3}$	$2.662 \cdot 10^{-3}$	$2.632 \cdot 10^{-3}$
4	$1.609 \cdot 10^{-3}$	$2.899 \cdot 10^{-3}$	$2.222 \cdot 10^{-3}$
5	$1.584 \cdot 10^{-3}$	$3.201 \cdot 10^{-3}$	$3.635 \cdot 10^{-3}$
6	$1.497 \cdot 10^{-3}$	$2.546 \cdot 10^{-3}$	$2.317 \cdot 10^{-3}$
7	$1.539 \cdot 10^{-3}$	$2.773 \cdot 10^{-3}$	$3.946 \cdot 10^{-3}$
8	$1.739 \cdot 10^{-3}$	$2.730 \cdot 10^{-3}$	$3.145 \cdot 10^{-3}$
9	$1.508 \cdot 10^{-3}$	$2.747 \cdot 10^{-3}$	$2.376 \cdot 10^{-3}$
10	$1.529 \cdot 10^{-3}$	$2.863 \cdot 10^{-3}$	$2.891 \cdot 10^{-3}$
11	$1.523 \cdot 10^{-3}$	$2.528 \cdot 10^{-3}$	$3.607 \cdot 10^{-3}$
12	$1.507 \cdot 10^{-3}$	$2.286 \cdot 10^{-3}$	$2.583 \cdot 10^{-3}$
13	$1.482 \cdot 10^{-3}$	$2.596 \cdot 10^{-3}$	$2.993 \cdot 10^{-3}$
14	$1.619 \cdot 10^{-3}$	$2.295 \cdot 10^{-3}$	$3.279 \cdot 10^{-3}$
15	$1.460 \cdot 10^{-3}$	$2.817 \cdot 10^{-3}$	$3.066 \cdot 10^{-3}$
16	$1.591 \cdot 10^{-3}$	$2.880 \cdot 10^{-3}$	$2.949 \cdot 10^{-3}$
17	$1.586 \cdot 10^{-3}$	$2.810 \cdot 10^{-3}$	$2.955 \cdot 10^{-3}$
18	$1.583 \cdot 10^{-3}$	$2.686 \cdot 10^{-3}$	$3.654 \cdot 10^{-3}$
19	$1.584 \cdot 10^{-3}$	$2.731 \cdot 10^{-3}$	$3.259 \cdot 10^{-3}$
20	$1.541 \cdot 10^{-3}$	$2.752 \cdot 10^{-3}$	$3.583 \cdot 10^{-3}$
21	$1.719 \cdot 10^{-3}$	$2.697 \cdot 10^{-3}$	$3.675 \cdot 10^{-3}$
22	$1.607 \cdot 10^{-3}$	$2.661 \cdot 10^{-3}$	$2.966 \cdot 10^{-3}$
23	$1.597 \cdot 10^{-3}$	$2.556 \cdot 10^{-3}$	$2.753 \cdot 10^{-3}$
24	$1.561 \cdot 10^{-3}$	$2.754 \cdot 10^{-3}$	$3.218 \cdot 10^{-3}$
25	$1.510 \cdot 10^{-3}$	$2.876 \cdot 10^{-3}$	$2.876 \cdot 10^{-3}$
26	$1.228 \cdot 10^{-3}$	$2.165 \cdot 10^{-3}$	$3.021 \cdot 10^{-3}$
27	$1.608 \cdot 10^{-3}$	$2.521 \cdot 10^{-3}$	$3.243 \cdot 10^{-3}$
28	$1.488 \cdot 10^{-3}$	$2.998 \cdot 10^{-3}$	$3.607 \cdot 10^{-3}$
29	$1.666 \cdot 10^{-3}$	$2.615 \cdot 10^{-3}$	$3.443 \cdot 10^{-3}$
30	$1.196 \cdot 10^{-3}$	$2.203 \cdot 10^{-3}$	$3.062 \cdot 10^{-3}$
31	$1.588 \cdot 10^{-3}$	$2.649 \cdot 10^{-3}$	$2.558 \cdot 10^{-3}$
32	$1.245 \cdot 10^{-3}$	$2.054 \cdot 10^{-3}$	$3.131 \cdot 10^{-3}$
33	$1.723 \cdot 10^{-3}$	$2.394 \cdot 10^{-3}$	$2.611 \cdot 10^{-3}$
34	$1.257 \cdot 10^{-3}$	$2.056 \cdot 10^{-3}$	$2.666 \cdot 10^{-3}$
35	$1.655 \cdot 10^{-3}$	$2.450 \cdot 10^{-3}$	$2.621 \cdot 10^{-3}$
36	$1.627 \cdot 10^{-3}$	$2.730 \cdot 10^{-3}$	$3.777 \cdot 10^{-3}$
37	$1.598 \cdot 10^{-3}$	$2.868 \cdot 10^{-3}$	$3.194 \cdot 10^{-3}$
38	$1.164 \cdot 10^{-3}$	$1.121 \cdot 10^{-3}$	$1.113 \cdot 10^{-3}$
39	$1.556 \cdot 10^{-3}$	$2.399 \cdot 10^{-3}$	$2.723 \cdot 10^{-3}$
40	$1.538 \cdot 10^{-3}$	$2.593 \cdot 10^{-3}$	$2.758 \cdot 10^{-3}$
41	$1.624 \cdot 10^{-3}$	$2.681 \cdot 10^{-3}$	$2.761 \cdot 10^{-3}$
42	$1.536 \cdot 10^{-3}$	$2.343 \cdot 10^{-3}$	$3.304 \cdot 10^{-3}$
43	$1.618 \cdot 10^{-3}$	$2.886 \cdot 10^{-3}$	$2.842 \cdot 10^{-3}$
44	$1.587 \cdot 10^{-3}$	$2.269 \cdot 10^{-3}$	$2.522 \cdot 10^{-3}$
45	$1.687 \cdot 10^{-3}$	$3.383 \cdot 10^{-3}$	$3.675 \cdot 10^{-3}$
46	$1.660 \cdot 10^{-3}$	$3.083 \cdot 10^{-3}$	$3.629 \cdot 10^{-3}$
47	$1.664 \cdot 10^{-3}$	$3.295 \cdot 10^{-3}$	$3.371 \cdot 10^{-3}$
48	$1.537 \cdot 10^{-3}$	$2.805 \cdot 10^{-3}$	$3.419 \cdot 10^{-3}$
49	$1.697 \cdot 10^{-3}$	$2.794 \cdot 10^{-3}$	$3.796 \cdot 10^{-3}$
50	$1.559 \cdot 10^{-3}$	$2.700 \cdot 10^{-3}$	$3.388 \cdot 10^{-3}$

Table 6.20: LR-parser - distributed SMC^2

Run	Party 3	Party 2	Party 1
1	$1.637 \cdot 10^{-3}$	$2.625 \cdot 10^{-3}$	$3.257 \cdot 10^{-3}$
2	$1.567 \cdot 10^{-3}$	$2.633 \cdot 10^{-3}$	$3.690 \cdot 10^{-3}$
3	$1.491 \cdot 10^{-3}$	$2.160 \cdot 10^{-3}$	$2.374 \cdot 10^{-3}$
4	$1.565 \cdot 10^{-3}$	$2.612 \cdot 10^{-3}$	$2.544 \cdot 10^{-3}$
5	$1.618 \cdot 10^{-3}$	$2.542 \cdot 10^{-3}$	$3.210 \cdot 10^{-3}$
6	$1.477 \cdot 10^{-3}$	$2.852 \cdot 10^{-3}$	$2.890 \cdot 10^{-3}$
7	$1.620 \cdot 10^{-3}$	$2.481 \cdot 10^{-3}$	$2.512 \cdot 10^{-3}$
8	$1.577 \cdot 10^{-3}$	$2.879 \cdot 10^{-3}$	$3.007 \cdot 10^{-3}$
9	$1.462 \cdot 10^{-3}$	$2.566 \cdot 10^{-3}$	$2.138 \cdot 10^{-3}$
10	$1.533 \cdot 10^{-3}$	$2.772 \cdot 10^{-3}$	$3.091 \cdot 10^{-3}$
11	$1.524 \cdot 10^{-3}$	$2.360 \cdot 10^{-3}$	$2.444 \cdot 10^{-3}$
12	$1.553 \cdot 10^{-3}$	$2.868 \cdot 10^{-3}$	$2.984 \cdot 10^{-3}$
13	$1.717 \cdot 10^{-3}$	$2.877 \cdot 10^{-3}$	$2.968 \cdot 10^{-3}$
14	$1.549 \cdot 10^{-3}$	$2.622 \cdot 10^{-3}$	$2.733 \cdot 10^{-3}$
15	$1.620 \cdot 10^{-3}$	$2.809 \cdot 10^{-3}$	$3.237 \cdot 10^{-3}$
16	$1.602 \cdot 10^{-3}$	$2.230 \cdot 10^{-3}$	$3.178 \cdot 10^{-3}$
17	$1.554 \cdot 10^{-3}$	$2.687 \cdot 10^{-3}$	$2.697 \cdot 10^{-3}$
18	$1.538 \cdot 10^{-3}$	$2.568 \cdot 10^{-3}$	$3.231 \cdot 10^{-3}$
19	$1.559 \cdot 10^{-3}$	$2.369 \cdot 10^{-3}$	$2.405 \cdot 10^{-3}$
20	$1.590 \cdot 10^{-3}$	$2.871 \cdot 10^{-3}$	$3.280 \cdot 10^{-3}$
21	$1.581 \cdot 10^{-3}$	$2.597 \cdot 10^{-3}$	$2.555 \cdot 10^{-3}$
22	$1.525 \cdot 10^{-3}$	$2.470 \cdot 10^{-3}$	$2.604 \cdot 10^{-3}$
23	$1.622 \cdot 10^{-3}$	$2.545 \cdot 10^{-3}$	$2.641 \cdot 10^{-3}$
24	$1.464 \cdot 10^{-3}$	$2.367 \cdot 10^{-3}$	$2.570 \cdot 10^{-3}$
25	$1.584 \cdot 10^{-3}$	$3.247 \cdot 10^{-3}$	$3.428 \cdot 10^{-3}$
26	$1.691 \cdot 10^{-3}$	$2.589 \cdot 10^{-3}$	$2.947 \cdot 10^{-3}$
27	$1.616 \cdot 10^{-3}$	$2.265 \cdot 10^{-3}$	$3.365 \cdot 10^{-3}$
28	$1.436 \cdot 10^{-3}$	$2.379 \cdot 10^{-3}$	$2.555 \cdot 10^{-3}$
29	$1.662 \cdot 10^{-3}$	$3.476 \cdot 10^{-3}$	$3.694 \cdot 10^{-3}$
30	$1.488 \cdot 10^{-3}$	$2.437 \cdot 10^{-3}$	$3.049 \cdot 10^{-3}$
31	$1.515 \cdot 10^{-3}$	$2.998 \cdot 10^{-3}$	$3.454 \cdot 10^{-3}$
32	$1.537 \cdot 10^{-3}$	$2.580 \cdot 10^{-3}$	$3.648 \cdot 10^{-3}$
33	$1.526 \cdot 10^{-3}$	$2.600 \cdot 10^{-3}$	$3.531 \cdot 10^{-3}$
34	$1.560 \cdot 10^{-3}$	$2.723 \cdot 10^{-3}$	$3.289 \cdot 10^{-3}$
35	$1.546 \cdot 10^{-3}$	$2.668 \cdot 10^{-3}$	$2.603 \cdot 10^{-3}$
36	$1.528 \cdot 10^{-3}$	$2.727 \cdot 10^{-3}$	$3.226 \cdot 10^{-3}$
37	$1.608 \cdot 10^{-3}$	$2.723 \cdot 10^{-3}$	$2.855 \cdot 10^{-3}$
38	$1.564 \cdot 10^{-3}$	$2.645 \cdot 10^{-3}$	$2.646 \cdot 10^{-3}$
39	$1.642 \cdot 10^{-3}$	$2.640 \cdot 10^{-3}$	$3.251 \cdot 10^{-3}$
40	$1.502 \cdot 10^{-3}$	$2.829 \cdot 10^{-3}$	$2.936 \cdot 10^{-3}$
41	$1.564 \cdot 10^{-3}$	$2.362 \cdot 10^{-3}$	$3.332 \cdot 10^{-3}$
42	$1.551 \cdot 10^{-3}$	$2.969 \cdot 10^{-3}$	$2.876 \cdot 10^{-3}$
43	$1.516 \cdot 10^{-3}$	$2.174 \cdot 10^{-3}$	$3.008 \cdot 10^{-3}$
44	$1.577 \cdot 10^{-3}$	$2.618 \cdot 10^{-3}$	$3.239 \cdot 10^{-3}$
45	$1.656 \cdot 10^{-3}$	$2.653 \cdot 10^{-3}$	$3.531 \cdot 10^{-3}$
46	$1.573 \cdot 10^{-3}$	$2.582 \cdot 10^{-3}$	$3.645 \cdot 10^{-3}$
47	$1.593 \cdot 10^{-3}$	$2.875 \cdot 10^{-3}$	$3.258 \cdot 10^{-3}$
48	$1.470 \cdot 10^{-3}$	$2.418 \cdot 10^{-3}$	$3.786 \cdot 10^{-3}$
49	$1.541 \cdot 10^{-3}$	$2.741 \cdot 10^{-3}$	$3.348 \cdot 10^{-3}$
50	$1.543 \cdot 10^{-3}$	$2.614 \cdot 10^{-3}$	$3.615 \cdot 10^{-3}$

Table 6.21: private-branching - distributed PICCO

Run	Party 3	Party 2	Party 1
1	3.23942	3.24051	3.24039
2	3.32584	3.32801	3.32975
3	3.41976	3.42104	3.42105
4	3.50588	3.57090	3.50791
5	3.50504	3.50583	3.50598
6	3.52268	3.52391	3.52445
7	3.49831	3.49998	3.50061
8	3.48104	3.48227	3.48231
9	3.27109	3.27205	3.27310
10	3.47149	3.47318	3.47330
11	3.46770	3.46963	3.46930
12	3.41150	3.41241	3.41247
13	3.50748	3.50928	3.50926
14	3.46141	3.46275	3.46261
15	3.48426	3.48535	3.48518
16	3.41861	3.41867	3.41885
17	3.52237	3.52339	3.52409
18	3.47470	3.47588	3.47688
19	3.41949	3.42051	3.42104
20	3.42541	3.42669	3.42736
21	3.41429	3.41580	3.41606
22	3.51975	3.52094	3.52186
23	3.40489	3.40664	3.40703
24	3.48310	3.48456	3.48458
25	3.43877	3.44023	3.44059
26	3.49143	3.49223	3.49284
27	3.40712	3.40810	3.40841
28	3.45702	3.45831	3.45812
29	3.47067	3.47160	3.47144
30	3.45716	3.45846	3.45848
31	3.47034	3.47139	3.47160
32	3.32684	3.32811	3.32818
33	3.44954	3.45095	3.45088
34	3.51742	3.51830	3.51939
35	3.45803	3.45918	3.45967
36	3.45175	3.45274	3.45274
37	3.43281	3.43406	3.43469
38	3.46394	3.46513	3.46535
39	3.53129	3.53300	3.53338
40	3.52613	3.52715	3.52741
41	3.48972	3.49049	3.49174
42	3.33954	3.34034	3.33998
43	3.47977	3.48063	3.48168
44	3.44981	3.45097	3.45144
45	3.42624	3.42774	3.42787
46	3.32197	3.32286	3.32288
47	3.51539	3.51679	3.51722
48	3.51441	3.51560	3.51640
49	3.45600	3.45720	3.45723
50	3.42339	3.42473	3.42506

Table 6.22: private-branching - distributed SMC^2

Run	Party 3	Party 2	Party 1
1	3.22575	3.22693	3.22709
2	3.31135	3.31285	3.31392
3	3.32549	3.32621	3.32626
4	3.14090	3.14259	3.14264
5	3.22530	3.22638	3.22636
6	3.19258	3.19365	3.19423
7	3.19228	3.19378	3.19397
8	3.25889	3.25975	3.26027
9	3.21114	3.21212	3.21235
10	3.20388	3.20535	3.20590
11	3.27287	3.27393	3.27411
12	3.27767	3.27873	3.27860
13	3.30338	3.30437	3.30485
14	3.31036	3.31107	3.31126
15	3.16673	3.16760	3.16863
16	3.16910	3.16895	3.16904
17	3.11065	3.11214	3.11259
18	3.18197	3.18278	3.18289
19	3.24375	3.24505	3.24465
20	3.22024	3.22139	3.22159
21	3.24179	3.24286	3.24307
22	3.33262	3.33370	3.33387
23	3.20144	3.20242	3.20316
24	3.18528	3.18648	3.18624
25	3.23825	3.23958	3.24017
26	3.23015	3.23186	3.23219
27	3.18560	3.18664	3.18778
28	3.22072	3.22191	3.22287
29	3.12446	3.12533	3.12534
30	3.28129	3.28270	3.28344
31	3.21136	3.21253	3.21259
32	3.30145	3.30228	3.30240
33	3.24651	3.24750	3.24837
34	3.25778	3.25880	3.25932
35	3.19060	3.19143	3.19146
36	3.25630	3.25724	3.25729
37	3.23832	3.23925	3.23916
38	3.20474	3.20554	3.20572
39	3.18841	3.18770	3.18864
40	3.33595	3.33666	3.33635
41	3.16652	3.16739	3.16768
42	3.17713	3.17823	3.17821
43	3.31800	3.31935	3.31969
44	3.18529	3.18608	3.18621
45	3.20002	3.20121	3.20190
46	3.18166	3.18285	3.18282
47	3.13168	3.13296	3.13328
48	3.16449	3.16536	3.16550
49	3.06241	3.06366	3.06418
50	3.25584	3.25689	3.25775

Table 6.23: private-branching-mult - distributed PICCO

Run	Party 3	Party 2	Party 1
1	4.59540	4.59636	4.59609
2	4.53936	4.54049	4.54061
3	4.49363	4.49520	4.49554
4	4.63247	4.63376	4.63421
5	4.52342	4.52411	4.52453
6	4.56561	4.56762	4.56772
7	4.43305	4.43410	4.43538
8	4.59511	4.59691	4.59689
9	4.57623	4.57736	4.57834
10	4.63623	4.63711	4.63736
11	4.60049	4.60142	4.60220
12	4.61652	4.61733	4.61768
13	4.55563	4.55672	4.55650
14	4.60324	4.60420	4.60450
15	4.26586	4.26708	4.26720
16	4.53426	4.53562	4.53601
17	4.55882	4.55996	4.56028
18	4.58667	4.58789	4.58838
19	4.59861	4.59996	4.60039
20	4.61689	4.61770	4.61797
21	4.53529	4.53596	4.53716
22	4.62299	4.62452	4.62489
23	4.56215	4.56327	4.56317
24	4.58665	4.58748	4.58826
25	4.50510	4.50637	4.50617
26	4.55368	4.55424	4.55463
27	4.57286	4.57407	4.57444
28	4.60919	4.61025	4.61068
29	4.57066	4.57152	4.57266
30	4.64601	4.64708	4.64726
31	4.54558	4.54722	4.54768
32	4.59367	4.59467	4.59476
33	4.53832	4.53918	4.53935
34	4.46108	4.46304	4.46307
35	4.51477	4.51649	4.51689
36	4.59608	4.59698	4.59723
37	4.48141	4.48268	4.48337
38	4.55729	4.55807	4.55816
39	4.57702	4.57796	4.57899
40	4.53622	4.53781	4.53826
41	4.65083	4.65278	4.65319
42	4.50843	4.51003	4.51003
43	4.61518	4.61689	4.61719
44	4.58877	4.58969	4.59020
45	4.61282	4.61464	4.61505
46	4.54268	4.54413	4.54483
47	4.51636	4.51743	4.51820
48	4.61769	4.61860	4.61878
49	4.63943	4.64074	4.64127
50	4.57378	4.57488	4.57506

Table 6.24: private-branching-mult - distributed SMC^2

Run	Party 3	Party 2	Party 1
1	3.24871	3.24850	3.24850
2	3.24513	3.24615	3.24697
3	3.28705	3.28819	3.28874
4	3.19546	3.19640	3.19654
5	3.30502	3.30589	3.30737
6	3.20291	3.20383	3.20390
7	3.26045	3.26134	3.26235
8	3.23805	3.23976	3.24205
9	3.22116	3.22245	3.22218
10	3.30338	3.30437	3.30457
11	3.23186	3.23358	3.23359
12	3.24383	3.24506	3.24505
13	3.26867	3.26974	3.27076
14	3.22740	3.22850	3.22928
15	3.14338	3.14479	3.14475
16	3.26569	3.26706	3.26746
17	3.19437	3.19544	3.19594
18	3.20916	3.21031	3.21105
19	3.24551	3.24647	3.24649
20	3.37681	3.37665	3.37682
21	3.07650	3.07712	3.07828
22	3.37420	3.37544	3.37491
23	3.22330	3.22408	3.22510
24	3.29998	3.30118	3.30176
25	3.18802	3.18935	3.18907
26	3.25063	3.25161	3.25164
27	3.24884	3.25005	3.25041
28	3.26558	3.26676	3.26762
29	3.14865	3.14964	3.14947
30	3.26928	3.27028	3.27031
31	3.18859	3.18982	3.18980
32	3.32418	3.32516	3.32620
33	3.16827	3.16913	3.16982
34	3.14683	3.14786	3.14780
35	3.14419	3.14557	3.14431
36	3.21007	3.21171	3.21177
37	3.28256	3.28375	3.28489
38	3.16202	3.16304	3.16409
39	3.26626	3.26717	3.26719
40	3.25889	3.26021	3.26068
41	3.20518	3.20634	3.20620
42	3.31639	3.31774	3.31825
43	3.20719	3.20798	3.20822
44	3.28656	3.28770	3.28863
45	3.23714	3.23846	3.23915
46	3.39525	3.39600	3.39599
47	3.27075	3.27159	3.27197
48	3.21416	3.21503	3.21609
49	3.14558	3.14653	3.14661
50	3.22141	3.22269	3.22296

Table 6.25: private-branching-add - distributed PICCO

Run	Party 3	Party 2	Party 1
1	6.50164	6.50246	6.50214
2	6.46684	6.46768	6.46832
3	6.48319	6.48395	6.48499
4	6.42252	6.42351	6.42375
5	6.46481	6.46628	6.46677
6	6.51813	6.51912	6.51919
7	6.41184	6.41273	6.41274
8	6.46118	6.46196	6.46297
9	6.48374	6.48493	6.48537
10	6.46173	6.46315	6.46359
11	6.44803	6.44916	6.44910
12	6.46533	6.46643	6.46651
13	6.44199	6.44286	6.44395
14	6.46640	6.46746	6.46753
15	6.45817	6.45885	6.45899
16	6.45497	6.45630	6.45621
17	6.47309	6.47411	6.47428
18	6.46976	6.47101	6.47160
19	6.47669	6.47790	6.47804
20	6.44402	6.44509	6.44497
21	6.46258	6.46305	6.46305
22	6.45184	6.45304	6.45379
23	6.46708	6.46874	6.46888
24	6.50002	6.50121	6.50155
25	6.43453	6.43648	6.43651
26	6.42595	6.42746	6.42788
27	6.44219	6.44334	6.44332
28	6.45200	6.45313	6.45325
29	6.45469	6.45614	6.45663
30	6.42916	6.43043	6.43107
31	6.43679	6.43779	6.43815
32	6.48524	6.48660	6.48672
33	6.46485	6.46631	6.46651
34	6.47884	6.48058	6.48092
35	6.37378	6.37457	6.37482
36	6.45143	6.45277	6.45247
37	6.43697	6.43808	6.43825
38	6.46466	6.46566	6.46550
39	6.45191	6.45330	6.45383
40	6.45636	6.45735	6.45731
41	6.44296	6.44427	6.44446
42	6.42655	6.42765	6.42843
43	6.46664	6.46785	6.46804
44	6.46296	6.46393	6.46425
45	6.44887	6.44998	6.45031
46	6.47456	6.47553	6.47668
47	6.46136	6.46230	6.46228
48	6.43963	6.44074	6.44077
49	6.47579	6.47713	6.47777
50	6.42132	6.42280	6.42314

Table 6.26: private-branching-add - distributed SMC^2

Run	Party 3	Party 2	Party 1
1	4.17418	4.17424	4.17450
2	3.74590	3.74765	3.74947
3	3.75282	3.75458	3.75771
4	3.82356	3.82493	3.82896
5	4.07070	4.07152	4.07156
6	4.10426	4.10537	4.10543
7	4.07842	4.07957	4.07937
8	4.11001	4.11096	4.11143
9	4.01142	4.01319	4.01349
10	4.11668	4.11746	4.11801
11	4.06778	4.06862	4.06961
12	4.14263	4.14362	4.14347
13	4.04563	4.04723	4.04756
14	4.10021	4.10129	4.10145
15	3.91254	3.91383	3.91392
16	4.21228	4.21303	4.21319
17	3.96840	3.96974	3.97054
18	4.01983	4.02107	4.02158
19	4.07483	4.07623	4.07665
20	4.12336	4.12429	4.12477
21	4.03838	4.03942	4.03948
22	3.95919	3.96049	3.96082
23	4.00141	4.00303	4.00309
24	3.99121	3.99227	3.99317
25	4.11988	4.12121	4.12112
26	4.01589	4.01801	4.01814
27	4.02534	4.02631	4.02747
28	4.13796	4.13845	4.13946
29	3.93888	3.93997	3.94090
30	4.00427	4.00550	4.00619
31	4.16683	4.16785	4.16876
32	4.05642	4.05820	4.05857
33	4.15460	4.15534	4.15575
34	4.15907	4.15994	4.16018
35	4.01624	4.01715	4.01773
36	4.00861	4.00983	4.01022
37	3.97886	3.97984	3.98024
38	4.11109	4.11238	4.11240
39	3.73384	3.73468	3.73571
40	3.77550	3.77636	3.77728
41	3.99583	3.99726	3.99769
42	4.02460	4.02564	4.02683
43	4.03986	4.04157	4.04184
44	4.14853	4.14932	4.14955
45	4.09817	4.09904	4.09903
46	3.75002	3.75129	3.75145
47	3.72462	3.72539	3.72574
48	3.66927	3.67020	3.67034
49	3.69157	3.69257	3.69338
50	3.73260	3.73357	3.73455

Table 6.27: private-branching-reuse - distributed PICCO

Run	Party 3	Party 2	Party 1
1	939.677	939.678	939.678
2	934.200	934.201	934.201
3	933.701	933.702	933.702
4	933.659	933.659	933.657
5	933.478	933.478	933.477
6	933.456	933.457	933.457
7	933.273	933.273	933.274
8	930.833	930.834	930.833
9	930.503	930.504	930.504
10	929.933	929.934	929.934
11	893.004	893.005	893.006
12	883.523	883.524	883.523
13	883.316	883.317	883.318
14	882.261	882.262	882.262
15	879.770	879.771	879.771
16	878.815	878.816	878.817
17	874.693	874.694	874.694
18	934.262	934.262	934.262
19	896.204	896.206	896.206
20	933.213	933.214	933.215
21	934.800	934.801	934.801
22	930.751	930.752	930.753
23	939.329	939.329	939.336
24	931.335	931.336	931.336
25	934.438	934.439	934.440
26	930.345	930.347	930.347
27	929.639	929.641	929.641
28	946.800	946.802	946.802
29	930.576	930.578	930.578
30	932.871	932.872	932.873
31	931.209	931.210	921.211
32	933.872	933.873	933.874
33	936.157	936.159	936.159
34	918.741	918.742	918.742
35	917.047	917.048	917.049
36	927.901	927.902	927.902
37	929.557	929.559	929.559
38	922.454	922.455	922.455
39	928.380	928.381	928.382
40	929.934	929.935	929.936
41	928.381	928.383	928.383
42	929.074	929.075	929.075
43	926.697	926.698	926.699
44	924.680	924.681	924.682
45	929.556	929.557	929.557
46	925.540	925.542	925.542
47	928.330	928.331	928.332
48	924.925	924.917	924.917
49	934.215	934.217	934.218
50	929.488	929.489	929.489

Table 6.28: private-branching-reuse - distributed SMC^2

Run	Party 3	Party 2	Party 1
1	475.252	475.252	475.252
2	474.749	474.750	474.751
3	475.626	475.627	475.628
4	456.563	456.564	456.564
5	450.563	450.651	450.661
6	471.822	471.824	471.835
7	471.068	471.069	471.080
8	448.641	448.643	448.654
9	452.754	452.756	452.767
10	452.021	452.023	452.042
11	458.783	458.785	458.796
12	465.188	465.190	465.211
13	474.854	474.855	474.855
14	478.646	478.647	478.648
15	475.822	475.824	475.825
16	474.708	474.709	474.709
17	472.542	472.543	472.544
18	475.325	475.326	475.326
19	474.326	474.327	474.328
20	474.090	474.091	474.092
21	474.348	474.349	474.349
22	473.485	473.486	473.486
23	471.181	471.182	471.183
24	477.147	477.148	477.148
25	476.871	476.872	476.872
26	480.441	480.442	480.443
27	477.851	477.852	477.853
28	477.094	477.095	477.095
29	485.520	485.521	485.521
30	450.194	450.196	450.195
31	450.494	450.496	450.496
32	449.176	449.176	449.177
33	452.379	452.380	452.381
34	472.001	472.001	472.002
35	477.727	477.728	477.730
36	478.685	478.685	478.687
37	476.482	476.483	476.483
38	473.843	473.843	473.844
39	476.136	476.137	476.138
40	476.402	476.404	476.404
41	475.359	475.360	475.360
42	478.351	478.352	478.352
43	477.333	477.334	477.334
44	475.617	475.618	475.619
45	475.808	475.810	475.810
46	474.918	474.919	474.920
47	476.794	476.795	476.796
48	474.985	474.986	474.986
49	475.471	475.472	475.473
50	474.977	474.978	474.978

6.2 Domain Specific Language (DSL)

In this section, we will discuss extending our formal model and implementation with a DSL. Writing SMC programs can be complex, and attempting to write them in such a way that they will be efficient at runtime requires in depth knowledge of the system, the underlying SMC protocols, and also the runtime environment. There are also limitations on what a programmer can do to write more efficient programs based on the system itself. For example, if a system does not provide the syntax to allow the programmer to specify certain elements (e.g., variable bit lengths) that could help improve the runtime of a program, then there is a limit to how much the programmer can do to improve the efficiency of their program on their own. A programmer could, of course, write out their own functions and libraries to attempt to improve efficiency, but at that point the SMC system is of questionable use to them. There is a great need for a system that simplifies writing efficient SMC programs and facilitates implementing optimizations on SMC programs.

In order to satisfy this need, we must extend our model to include a DSL. Our goal with this DSL extension is to have a fully formalized model for SMC that is proven correct and secure while also helping programmers write more efficient SMC programs that can easily be optimized. In order to accomplish this, we must begin by designing the DSL, keeping in mind what syntactic elements may help us in enabling optimizations. Then we must formalize the DSL and it's translation to our Multiparty SMC^2 semantics. Finally, we must implement the DSL, enabling us to show that by providing the DSL we can simplify writing SMC programs that will be more efficient at runtime.

6.2.1 Design

When trying to design our DSL, we first need to take into consideration what we want to accomplish with this DSL. As discussed above, we need a way to simplify writing and optimizing SMC programs. One fairly straight-forward way to do this is by designing the DSL with at least one optimization in mind that would help improve the efficiency of SMC programs. To start, we have chosen to focus on the optimization of finding optimal variable sizes, as the performance of SMC operations are often dependent on the bit length of data.

Optimal Variable Sizes

Because performance of SMC operations is often dependent on the bit length of the data provided as input, we plan on exploring automatic mechanisms for inferring optimal variable sizes. Specifically, we want to automatically reduce the number of bits necessary to encode various data types, as there are many operations on private variables whose cost is at least linear in the bit length of their representation. If we can determine the bounds on the range of values a variable will store (e.g., based on the input bit length), we can determine the optimal number of bits needed to represent the values without overflow. As an example, consider the code in Figure 6.12 that uses two private bit arrays A and B of size S.

```
1 private int<1> A[S], B[S];
2 public int d; private int hd, c;
3 hd = A @ B;
4 if (hd > d) c = 1;
5 else c = 0;
```

Figure 6.12: Computation of Hamming distance: notation $<n>$ denotes the size of integers in bits and @ the dot product.

If we compute the Hamming distance hd and consequently use this result in a comparison, the size of the variable in which the Hamming distance is stored will determine the cost of the comparison. If hd is declared to a default value (such as 32 or 64 bits), performance can be several times higher than necessary if $\log(\mathrm{S})$ is substantially lower than the default bit length. Inference of the optimal number of bits needed for the computation requires effective constant propagation [42], loop bound inference [43, 44], as well as precise data-flow information [45, 46]. We are not aware of previous study of this topic in the SMC context.

DSL Grammar

When designing the grammar for the DSL, we chose to start with a fairly basic set of statements, expressions, and operations. We maintained the primitive functions from SMC^2 to facilitate data input and output, as well as annotating types with privacy labels. After starting with this basic set, we began to fully consider the first optimization we will be implementing – finding the optimal bit length of variables. To enable us to do this, we first need the syntax to annotate variable declarations with a size n, indicating that the given variable should not contain data that is larger than what can be stored in n bits. We chose to use the syntax $ty < n > \ var$, as it provides a simple way to declare the variable's size, and it is already present within PICCO, the system

$$
\begin{array}{llll}
ty & \in \mathit{Type} & ::= & a\ bty \mid \overline{ty} \rightarrow ty \\
bty & \in \mathit{BasicType} & ::= & \text{int} \\
a & \in \mathit{PrivacyLabel} & ::= & \text{private} \mid \text{public} \\
 & & & \\
s & \in \mathit{Statement} & ::= & var = e \mid s;\ s \mid decl \mid \text{if}\ (e)\ s\ \text{else}\ s \mid \{s\} \mid prim \mid e \\
 & & & \mid \text{bound}\ n\ \text{for}(s; e; s)\{s\} \mid \text{return}\ e \\
e & \in \mathit{Expression} & ::= & e\ bop\ e \mid var \mid x(\overline{e}) \mid (e) \mid v \mid ++\ x \\
decl & \in \mathit{Declaration} & ::= & ty\ var \mid ty < n >\ var \mid ty\ x(\overline{p})\{s\} \\
var & \in \mathit{Variable} & ::= & x \mid x[e] \\
v & \in \mathit{Value} & ::= & n \mid \overline{v} \mid \text{skip} \\
 & & & \\
prim & \in \mathit{PrimitiveFunction} & ::= & \text{smcinput}(var, n) \mid \text{smcoutput}(var, n) \\
bop & \in \mathit{BinaryOperation} & ::= & - \mid + \mid \cdot \mid \div \mid == \mid != \mid < \mid << \mid >> \\
\overline{e} & \in \mathit{ExpressionList} & ::= & \overline{e},\ e \mid e \mid \text{void} \\
\overline{p} & \in \mathit{ParameterList} & ::= & \overline{p},\ ty\ var \mid ty\ var \mid \text{void} \\
 & & & \\
\overline{cl} & \in \mathit{ComputeSizeList} & ::= & f : x \rightarrow e \\
\theta & \in \mathit{OptimalSizeMap} & ::= & f : x \rightarrow (ty, n, n, n, \overline{cl}, \overline{x}) \\
 & & & \\
\overline{ty} & \in \mathit{TypeList} & & \\
\overline{v} & \in \mathit{ValueList} & n & \in \mathbb{N} \\
\overline{x} & \in \mathit{VariableList} & x & \in \mathit{Variable}
\end{array}
$$

Figure 6.13: DSL Grammar. The color red denotes terms specific to programs written in the DSL extension of SMC^2, and the color blue denotes terms synthesized by the semantics.

upon which our system is built. We then carefully considered each of the elements in our grammar and how they would impact our ability to evaluate what size a variable should be during the execution of a program.

The element we found to prove the most difficult without having further information from the programmer was loops. Given that the number of times a loop will execute can be based entirely upon variables (e.g., using the termination condition $i < j$), it is possible for a programmer to write a program that we cannot obtain a loop bound just by simply analyzing the content of the program. We could simply increase the size of the variables modified within the loop to be their maximum possible size, but this would unnecessarily limit us in our ability to improve the efficiency of many programs when using this optimization. Instead, we chose to extend the syntax of SMC^2 to include a bound specification for loops (i.e., $\text{bound}\ n\ \text{for}(s; e; s)\{s\}$). This allows us to reason about how large each variable could possibly get in the worst case execution of the loop. With this, we can now formalize our optimal variable sizing pass.

6.2.2 Formalization

Algorithm 147 (GetBitsize) takes a number and returns the bit size necessary to store that number in memory. This algorithm is used when we have variables with constant values or other hard-coded numbers in a program. We add in an additional check at the beginning to check if the value is 0, and if it is, set the return

Algorithm 147 $n_2 \leftarrow \mathrm{GetBitsize}(n_1)$

```
n_2 = 0
if n_1 = 0 then
    n_2 = 1
end if
while n_1 ≥ 2 do
    n_1 = n_1 ÷ 2
    n_2 = n_2 + 1
end while
if n_1 ≥ 1 then
    n_2 = n_2 + 1
end if
return n_2
```

value to 1, as the size should not be 0.

Algorithm 148 $n_3 \leftarrow \text{GetSizeBinary}(n_1, n_2, e, bop)$

```
1: n_3 = 0
2: if (bop = ·) then
3:     n_3 = n_1 + n_2
4: else if (bop = +) ∧ (bop = −) then
5:     n_3 = max(n_1, n_2) + 1
6: else if (bop = ÷) then
7:     n_3 = n_1
8: else if (bop =<<) then
9:     if (e = n) then
10:        n_3 = n_1 + n
11:    else
12:        n_3 = MAX_VAL
13:    end if
14: else if (bop =>>) then
15:    if (e = n) then
16:        n_3 = n_1 − n
17:    else
18:        n_3 = n_1
19:    end if
20: else
21:    n_3 = 1
22: end if
23: if n_3 > MAX_VAL then
24:    n_3 = MAX_VAL
25: else if n_3 < 1 then
26:    n_3 = 1
27: end if
28: return n_3
```

Algorithm 148 (GetSizeBinary) computes the size that is necessary for a variable storing the result of the given binary operation bop on values of size n_1 and n_2. It takes as input the size of the sizes of the first and second expression, the second expression, and the binary operation that is being performed. For multiplication, we add the two sizes together. For addition, we take the maximum of the two sizes and add one to obtain the final size; in order to account for negative values, we do the same for subtraction. For division, we keep the size of the dividend, n_1.

For left shifts, if the expression is a constant number, then we increment the size by this number; otherwise we increase the size to the maximum size. For right shifts, if the expression is a constant, we can decrement the size by this number. If not, we keep the size of the first expression (in the case of a shift by 0). The final case accounts for all comparison operations, which will return either 0 or 1, and therefore needs a bit length of 1. We then check the size we are returning to ensure that the size is not greater than the maximum size or less than 1.

Algorithm 149 (EvaluateVariableSizeExpr) computes the size requirement of the given expression, using recursion to evaluate more complex expressions. To evaluate binary operations, it leverages Algo-

Algorithm 149 $(\theta_f, n_f) \leftarrow \text{EvaluateVariableSizeExpr}(e, \theta)$

1: $\theta_f = \theta$
2: $n_f = 1$
3: **if** $(e = e_1 \ bop \ e_2)$ **then**
4: $\quad (\theta_1, n_1) = \text{EvaluateVariableSizeExpr}(e_1, \theta)$
5: $\quad (\theta_2, n_2) = \text{EvaluateVariableSizeExpr}(e_2, \theta_1)$
6: $\quad n_f = \text{GetSizeBinary}(n_1, n_2, e_2, bop)$
7: $\quad \theta_f = \theta_2$
8: **else if** $(e = ++x)$ **then**
9: $\quad \theta_1[x \rightarrow (ty, n_{max}, n_{curr_max}, n_{curr}, \text{NULL}, \text{NULL})] = \theta$
10: $\quad n_f = n_{curr} + 1$
11: $\quad \theta_f = \theta_1[x \rightarrow (ty, n_{max}, n_{curr_max}, n_f, \text{NULL}, \text{NULL})]$
12: **else if** $(e = x[e_1]) \vee (e = x)$ **then**
13: $\quad (\theta_f, n_1) = \text{EvaluateVariableSizeExpr}(e_1, \theta)$
14: $\quad (ty, n_{max}, n_{curr_max}, n_{curr}, \text{NULL}, \text{NULL}) = \theta_f(x)$
15: $\quad n_f = n_{curr}$
16: **else if** $(e = (e_1))$ **then**
17: $\quad (\theta_f, n_f) = \text{EvaluateVariableSizeExpr}(e_1, \theta)$
18: **else if** $(e = x(\overline{e}))$ **then**
19: $\quad (\theta_f, n_f) = \text{EvaluateFunctionSize}(x(\overline{e}), \theta)$
20: **else if** $(e = n)$ **then**
21: $\quad n_f = \text{GetBitsize}(n)$
22: **end if**
23: **return** (θ_f, n_f)

rithm 148. For pre-increment operations, it increases the size requirement for the given variable by 1, and returns this size. For memory allocation and finding the address of a variable, we evaluate e_1, and return the size of a Void pointer, as these will return a pointer to the given memory address. When evaluating a parenthesized expression, we evaluate the inner expression and pass along the results. For function calls, we are currently returning the size of the return type. This can be refined further by analyzing the content of the function with relation to the given arguments. To evaluate a hard-coded number n, we find the size of the number in bits and return this size. Similarly, with the primitive function for finding the size of a type, we evaluate the size of the type to get n and return the number of bits required to store the number n.

Algorithm 150 $\theta_3 \leftarrow \text{MaxSizeMap}(\theta_1, \theta_2)$

1: $\theta_3 = \theta_2$
2: **for all** $(x \rightarrow (ty, n_{max}, n_{curr_max}, n_{curr}, \overline{cl}, \overline{x}) \in \theta_1)$ **do**
3: $\quad$ **if** $(x \notin \theta_2)$ **then**
4: $\quad\quad \theta_3 = \theta_3[x \rightarrow (ty, n_{max}, n_{curr_max}, n_{curr}, \overline{cl}, \overline{x})]$
5: $\quad$ **else**
6: $\quad\quad \theta_4[x \rightarrow (ty, n_{max}, n'_{curr_max}, n'_{curr}, \overline{cl}, \overline{x})] = \theta_3$
7: $\quad\quad \theta_3 = \theta_4[x \rightarrow (ty, n_{max}, max(n_{curr_max}, n'_{curr_max}), max(n_{curr}, n'_{curr}), \overline{cl}, \overline{x})]$
8: $\quad$ **end if**
9: **end for**
10: **return** θ_3

Algorithm 150 (MaxSizeMap) is designed to get the maximum values possible that can occur from the

evaluation of either branch. It is used after both branches of an **`if else`** statement have been evaluated, and must keep the larger size that is determined for either branch as the new value for the variable in order to accommodate having either branch being the true branch that will be evaluated when the program is run. It takes the resulting maps from the **`then`** and **`else`** branches as input, and chooses one to serve as the base for the resulting map, then iterates through the other. If a variable mapping is not present in the resulting map, it simply adds in the variable mapping; otherwise, it keeps the maximum value of the current max and current size for each variable that is present in both maps. It then returns the final, combined map.

Algorithm 151 $\theta_f \leftarrow \mathrm{SetCurrentSize}(\theta, x, n)$

1: $\theta_1[x \rightarrow (ty, n_{max}, n_{curr_max}, n_{curr}, \overline{cl}, \overline{x})] = \theta$
2: $(n_1, n_2) = (n_{curr_max}, n)$
3: **if** $(n \geq n_{max})$ **then**
4: $\quad (n_1, n_2) = (n_{max}, n_{max})$
5: **else if** $(n > n_{curr_max})$ **then**
6: $\quad n_1 = n$
7: **end if**
8: $\theta_f = \theta_1[x \rightarrow (ty, n_{max}, n_1, n_2, \overline{cl}, \overline{x})]$
9: **return** θ_f

Algorithm 151 ($\mathrm{SetCurrentSize}$) is designed to update the size for a variable in our variable size map θ. It takes as input the current map, the variable to be updated, and the new size to update the variable with, and returns the updated map. In this algorithm, our defaults in line 2 are set to update the current maximum as itself and the current size as the new size. We then ensure that if the new size is larger than the maximum size for this variable, we instead updated these sizes as the maximum size. Otherwise, if the new size is larger than the current max, we then ensure that the current max is updated to this new maximum size.

Algorithm 152 ($\mathrm{EvaluateVariableSizeStmt}$) takes a program s and variable size map θ as input. It iterates over and evaluates all statements in s in order to determine the smallest possible size for each variable based on the operations performed within the program and the sizes provided by the programmer for the variables storing the input data. For each declaration with a size annotation, a mapping is added to θ specifying the variable name, type, and maximum size as the given size; the current size and current maximum size are initialized to 1, and to be updated during later statements. For arrays, the sizes refer to that of each element within the array. For declarations without a size annotation, we initialize the mapping with the size 1 for the current maximum and current size, and a maximum size of the maximum size for the declared type. For function declarations and definitions, we initialize the mapping for the function variable with the expected size of the return type.

For regular assignments (i.e., $x = e$), we evaluate the expression, then update the mapping for the variable with the size returned from evaluating the expression using Algorithm 151. This prevents the variable's size from growing beyond that of its maximum size. For array assignments at an index (i.e., $x[e_1] = e_2$), we evaluate the first expression to ensure we catch and increment operations, then we evaluate the second expression and update the mapping for the variable with the size returned from the evaluation of the second expression using Algorithm 151. For bounded loops, we first evaluate the initialization statement s_1, then use the loop bound n to evaluate the loop contents n times. We must evaluate the loop contents in the order the loop would be executed: the expression first, then the loop body s_3, and finally the increment statement s_2. For branches, we first evaluate the expression, then both of the branches, keeping all changes. With evaluating both branches separately after the expression, we are ensuring that we will update the sizes for variables accurately for each branch. Then we perform a comparison between θ_2 and θ_3, keeping the maximum sizes for each changed variable in order to account for either branch being taken using Algorithm 150. Whenever we see data being input to a variable, we assume that it will be the maximum size defined for that variable, and we update the variable's current size as such. This functionality is why all variables storing input data should have the sizes pre-defined by the programmer.

Algorithm 152 $(\theta_f) \leftarrow \text{EvaluateVariableSizeStmt}(s, \theta)$

$\theta_f = \theta$
for all $s_1 \in s$ **do**
  if $(s_1 = ty < n > \ x[e])$ **then**
    $(\theta_1, n_1) = \text{EvaluateVariableSizeExpr}(e, \theta_f)$
    $\theta_f = \theta_1[x \rightarrow (ty, n, 1, 1, \text{NULL}, \text{NULL})]$
  else if $(s_1 = ty < n > \ x)$ **then**
    $\theta_f = \theta_f[x \rightarrow (ty, n, 1, 1, \text{NULL}, \text{NULL})]$
  else if $(s_1 = ty\ x[e])$ **then**
    $(\theta_1, n) = \text{EvaluateVariableSizeExpr}(e, \theta_f)$
    $\theta_f = \theta_1[x \rightarrow (ty, \tau(ty) \cdot 8, 1, 1, \text{NULL}, \text{NULL})]$
  else if $(s_1 = ty\ x)$ **then**
    $\theta_f = \theta_f[x \rightarrow (ty, \tau(ty) \cdot 8, 1, 1, \text{NULL}, \text{NULL})]$
  else if $(s_1 = ty\ x(\overline{p})\{s_2\})$ **then**
    $(\theta_f) \leftarrow \text{EvaluateFunctionDefinition}(ty, 0, x, \overline{p}, s_2, \theta_f)$
  else if $(s_1 = ty < n > x(\overline{p})\{s_2\})$ **then**
    $(\theta_f) \leftarrow \text{EvaluateFunctionDefinition}(ty, n, x, \overline{p}, s_2, \theta_f)$
  else if $(s_1 = x = e)$ **then**
    $(\theta_1, n) = \text{EvaluateVariableSizeExpr}(e, \theta_f)$
    $\theta_f = \text{SetCurrentSize}(\theta_1, x, n)$
  else if $(s_1 = x[e_1] = e_2)$ **then**
    $(\theta_1, n_1) = \text{EvaluateVariableSizeExpr}(e_1, \theta_f)$
    $(\theta_2, n_2) = \text{EvaluateVariableSizeExpr}(e_2, \theta_1)$
    $\theta_f = \text{SetCurrentSize}(\theta_1, x, n_2)$
  else if $(s_1 = \text{bound}\ n\ \text{for}(s_2; e; s_3)\ \{s_4\})$ **then**
    $(\theta_1) = \text{EvaluateVariableSizeStmt}(s_2, \theta_f)$
    for all $i \in \{0...n\}$ **do**
      $(\theta_2, n_1) = \text{EvaluateVariableSizeExpr}(e, \theta_1)$
      $(\theta_3) = \text{EvaluateVariableSizeStmt}(s_4, \theta_2)$
      $(\theta_1) = \text{EvaluateVariableSizeStmt}(s_3, \theta_3)$
    end for
    $\theta_f = \theta_1$
  else if $(s_1 = \text{if}\ (e)\ s_2\ \text{else}\ s_3)$ **then**
    $(\theta_1, n) = \text{EvaluateVariableSizeExpr}(e, \theta_f)$
    $\theta_2 = \text{EvaluateVariableSizeStmt}(s_2, \theta_1)$
    $\theta_3 = \text{EvaluateVariableSizeStmt}(s_3, \theta_1)$
    $\theta_f = \text{MaxSizeMap}(\theta_2, \theta_3)$
  else if $(s_1 = \{s_2\})$ **then**
    $\theta_f = \text{EvaluateVariableSizeStmt}(s_2, \theta_f)$
  else if $(s_1 = e)$ **then**
    $(\theta_f, n) = \text{EvaluateVariableSizeExpr}(e, \theta_f)$
  else if $(s_1 = \text{smcinput}(x, n))$ **then**
    $(ty, n_{max}, n_{curr_max}, n_{curr}, \text{NULL}, \text{NULL}) = \theta_f(x)$
    $\theta_f = \text{SetCurrentSize}(\theta_f, x, n_{max})$
  end if
end for
return (θ_f)

Algorithm 153 $(\overline{ty}, \overline{x}, \theta_f) \leftarrow \text{EvaluateFunctionParameters}(\overline{p}, \theta)$

1: $(\overline{ty}, \overline{x}, \theta_f) = ([\,], [\,], \theta)$
2: **while** $(\overline{p} \neq \text{void})$ **do**
3: **if** $(\overline{p} = ty\ x) \vee (\overline{p} = ty\ x[e])$ **then**
4: $\overline{ty} = ty :: \overline{ty}$
5: $\overline{x} = x :: \overline{x}$
6: $\overline{p} = \text{void}$
7: $\theta_f = \theta_f[x \rightarrow (ty, \tau(ty) \cdot 8, 1, 1, \text{NULL}, \text{NULL})]$
8: **else if** $(\overline{p} = ty < n > x) \vee (\overline{p} = ty < n > x[e])$ **then**
9: $\overline{ty} = ty :: \overline{ty}$
10: $\overline{x} = x :: \overline{x}$
11: $\overline{p} = \text{void}$
12: $\theta_f = \theta_f[x \rightarrow (ty, n, 1, 1, \text{NULL}, \text{NULL})]$
13: **else if** $(\overline{p} = \overline{p}_1, ty\ x) \vee (\overline{p} = \overline{p}_1, ty\ x[e])$ **then**
14: $\overline{ty} = ty :: \overline{ty}$
15: $\overline{x} = x :: \overline{x}$
16: $\overline{p} = \overline{p}_1$
17: $\theta_f = \theta_f[x \rightarrow (ty, \tau(ty) \cdot 8, 1, 1, \text{NULL}, \text{NULL})]$
18: **else if** $(\overline{p} = \overline{p}_1, ty < n > x) \vee (\overline{p} = \overline{p}_1, ty < n > x[e])$ **then**
19: $\overline{ty} = ty :: \overline{ty}$
20: $\overline{x} = x :: \overline{x}$
21: $\overline{p} = \overline{p}_1$
22: $\theta_f = \theta_f[x \rightarrow (ty, n, 1, 1, \text{NULL}, \text{NULL})]$
23: **end if**
24: **end while**
25: **return** $\overline{ty}$

Algorithm 153 (EvaluateFunctionParameters) is designed to obtain a list of types of the parameters $\overline{ty}$, a list of parameter variable names $\overline{x}$, and add mappings for all of the parameter variables into memory. If the parameter is not given a bit size in its declaration, then the maximum size is defined to be the maximum size for that type; otherwise, it is given the bit size from its declaration. All parameter entries are given an initial current size and current maximum size of 1, which will be updated as we evaluate function calls for this function.

Algorithm 154 $(\theta_f) \leftarrow \text{EvaluateFunctionDefinition}(ty, n, x, \overline{p}, s, \theta)$

1: $(\overline{ty}, \overline{x}, \theta_1) \leftarrow \text{EvaluateFunctionParameters}(\overline{p}, \theta)$
2: $(\overline{cl}, n_{max}) = ([\,], n)$
3: **if** $(n_{max} = 0)$ **then**
4: $n_{max} = \tau(ty) \cdot 8$
5: **end if**
6: $(\overline{cl}, \theta_2) = \text{GetComputeListStmt}(s, \theta_1)$
7: $\theta_f = \theta_2[x \rightarrow (\overline{ty} \rightarrow ty, n_{max}, 1, 1, \overline{cl}, \overline{x})]$
8: **return** (θ_f)

Algorithm 154 (EvaluateFunctionDefinition) is designed to handle the evaluation of a function definition. It first calls Algorithm 153 to evaluate the function parameters, obtaining the type list for the function, the list of parameter variable names, and the updated optimal size map with the function parameters added in.

It is worthwhile to note again here that we assume all variable names are unique, including those used as function parameters. We then initialize the compute size list as an empty list, and the max size as the number n given as a parameter. We check to ensure that if n was given as 0, we set the max size to the maximum size for the type, as this means that the maximum size was not pre-defined for the function. We then call Algorithm 155 to obtain the compute size list for the function body. It is important to note here that we do not currently support recursive functions; the model can be extended to support them in the future, but this extension is beyond the scope of the current formalization.

Algorithm 155 $(\overline{cl}, \theta_f) \leftarrow \text{GetComputeListStmt}(s, \theta)$

1: $\theta_f = \theta$
2: $\overline{cl} = [\,]$
3: **for all** $s_1 \in s$ **do**
4: **if** $(s_1 = ty < n > \ x[e])$ **then**
5: $\theta_f = \theta_f[x \rightarrow (ty, n, 1, 1, \text{NULL}, \text{NULL})]$
6: $\overline{cl} = \overline{cl}[\text{setSize}(x, 1)][\text{GetComputeListExpr}(e)]$
7: **else if** $(s_1 = ty < n > \ x)$ **then**
8: $\theta_f = \theta_f[x \rightarrow (ty, n, 1, 1, \text{NULL}, \text{NULL})]$
9: $\overline{cl} = \overline{cl}[\text{setSize}(x, 1)]$
10: **else if** $(s_1 = ty\ x[e])$ **then**
11: $\theta_f = \theta_f[x \rightarrow (ty, \tau(ty) \cdot 8, 1, 1, \text{NULL}, \text{NULL})]$
12: $\overline{cl} = \overline{cl}[\text{setSize}(x, 1)][\text{GetComputeListExpr}(e)]$
13: **else if** $(s_1 = ty\ x)$ **then**
14: $\theta_f = \theta_f[x \rightarrow (ty, \tau(ty) \cdot 8, 1, 1, \text{NULL}, \text{NULL})]$
15: $\overline{cl} = \overline{cl}[\text{setSize}(x, 1)]$
16: **else if** $(s_1 = x = e)$ **then**
17: $\overline{cl} = \overline{cl}[\text{setSize}(x, \text{GetComputeListExpr}(e))]$
18: **else if** $(s_1 = x[e_1] = e_2)$ **then**
19: $\overline{cl} = \overline{cl}[\text{GetComputeListExpr}(e_1)][\text{setSize}(x, \text{GetComputeListExpr}(e_2))]$
20: **else if** $(s_1 = \text{bound}\ n\ \text{for}(s_2; e; s_3)\ \{s_4\})$ **then**
21: $\overline{cl}_1 = \overline{cl}[\text{GetComputeListStmt}(s_2)]$
22: $\overline{cl} = \overline{cl}_1[\text{loop}(n, \text{GetComputeListExpr}(e), \text{GetComputeListStmt}(s_4), \text{GetComputeListStmt}(s_3))]$
23: **else if** $s_1 = \text{if}\ (e)\ s_2\ \text{else}\ s_3$ **then**
24: $\overline{cl}_1 = \overline{cl}[\text{GetComputeListExpr}(e)]$
25: $\overline{cl} = \overline{cl}_1[\text{branch}(\text{GetComputeListStmt}(s_2), \text{GetComputeListStmt}(s_3))]$
26: **else if** $(s_1 = \{s_2\})$ **then**
27: $\overline{cl} = \overline{cl}[\text{GetComputeListStmt}(s_2)]$
28: **else if** $(s_1 = e)$ **then**
29: $\overline{cl} = \overline{cl}[\text{GetComputeListExpr}(e)]$
30: **else if** $(s_1 = \text{smcinput}(x, n))$ **then**
31: $\overline{cl} = \overline{cl}[\text{setSize}(x, \text{getMaxSize}(x))]$
32: **else if** $(s_1 = \text{return}\ e)$ **then**
33: $\overline{cl} = \overline{cl}[\text{return}(\text{GetComputeListExpr}(e))]$
34: **end if**
35: **end for**
36: **return** $(\overline{cl}, \theta_f)$

Algorithm 155 (GetComputeListStmt) analyzes the body of a function and creates the compute size list for the function. This list will then be used to evaluate the return size when this function is called. It takes the function body s and the optimal size map θ as input, and returns the compute size list for the function and

the optimal size map updated with any variables local to the function. It is important to note here that we assume all variable names will be unique. We use blue text in this algorithm to identify 'functions' that will be used in Algorithm 157 when we are evaluating the size of the function call – they are part of the compute list equation, and thus not evaluated here. All other functions are evaluated here.

Algorithm 156 $(cl) \leftarrow \text{GetComputeListExpr}(e)$

```
1: if (e = e1 + e2) ∨ (e = e1 − e2) then
2:     cl = add(max(GetComputeListExpr(e1), GetComputeListExpr(e2)), 1)
3: else if e = e1 · e2 then
4:     cl = add(GetComputeListExpr(e1), GetComputeListExpr(e2))
5: else if e = e1 ÷ e2 then
6:     cl = first(GetComputeListExpr(e1), GetComputeListExpr(e2))
7: else if e = e1 << e2 then
8:     if e2 = n then
9:         cl = add(GetComputeListExpr(e1), n)
10:    else
11:        cl = first(getMaxSize(x), first(GetComputeListExpr(e1), GetComputeListExpr(e2)))
12:    end if
13: else if e = e1 >> e2 then
14:    if e2 = n then
15:        cl = add(GetComputeListExpr(e1), −n)
16:    else
17:        cl = first(GetComputeListExpr(e1), GetComputeListExpr(e2))
18:    end if
19: else if e = (e1) then
20:    cl = (GetComputeListExpr(e1))
21: else if e = x[e1] then
22:    cl = first(getCurrentSize(x), GetComputeListExpr(e1))
23: else if e = x then
24:    cl = getCurrentSize(x)
25: else if e = ++x then
26:    cl = setSize(x, add(getCurrentSize(x), 1))
27: else if e = n then
28:    cl = GetBitsize(n)
29: end if
30: return cl
```

Algorithm 156 (GetComputeListExpr) analyzes an expression to form an equation to add to the compute list for finding the size needed for the result of that expression. This algorithm is used by to develop the compute list during the evaluation of a function definition. We use blue text in this algorithm to identify compute list commands that will be used in Algorithm 157 when we are evaluating the size of the function call – they are part of the compute list equation, and thus not evaluated here. All other functions are evaluated here.

Algorithm 157 (EvaluateFunctionSize) is designed to manage the evaluation of the function arguments and function body to obtain the size of the value returned by the function call. It first looks up the function in the optimal size map, then assigns the arguments to their corresponding parameters. Then, it proceeds to

Algorithm 157 $(\theta_f, n_{ret}) = \text{EvaluateFunctionSize}(x(\overline{e}), \theta)$

1: $\theta_1[x \rightarrow (\overline{ty} \rightarrow ty, n_{max}, n_{curr_max}, n_{curr}, \overline{cl}, \overline{x})] = \theta$
2: $\theta_f = \theta_1$
3: $n_{ret} = 0$
4: $n_x = |\overline{x}|$
5: **for all** $i \in \{0...n_x - 1\}$ **do**
6: $\quad x = \overline{x}[i]$
7: $\quad e = \overline{e}[i]$
8: $\quad \theta_f = \text{EvaluateVariableSizeStmt}(x = e, \theta_f)$
9: **end for**
10: **for** $\overline{cl}_1 \in \overline{cl}$ **do**
11: $\quad (n_1, \theta_f, n_2) = \text{EvaluateComputeList}(\overline{cl}, \theta, 0)$
12: $\quad$ **if** $(n_2 \geq n_{max})$ **then**
13: $\quad\quad n_{ret} = n_{max}$
14: $\quad$ **else if** $(n_2 > n_{ret})$ **then**
15: $\quad\quad n_{ret} = n_2$
16: $\quad$ **end if**
17: **end for**
18: $n_m = max(n_{curr_max}, n_{ret})$
19: $\theta_f = \theta_f[x \rightarrow (ty, n_{max}, n_m, n_{ret}, \overline{cl}, \overline{x})]$
20: **return** (θ_f, n_{ret})

evaluate the compute size list that is stored for this function, finding the maximum return size for the function. In line 18, it will find the new current maximum return size for the function, then in line 19 update the current maximum and the current return size for the function. Finally, it returns the updated optimal size map and the return size.

Algorithm 158 (EvaluateComputeList) to evaluate the final size of a compute list command. We have nine compute list commands, which can be nested within each other. We use blue text in this algorithm to highlight the compute list commands, differentiating them from other algorithm and function names used in the formalism. It takes the compute size list that we need to evaluate, an optimal size map to use and update, and a return size m. Once complete, it will return the size of the current list that was computed, the updated optimal size map, and an updated return size. Command add evaluates its two arguments, then adds the resulting values together. Command max evaluates its two arguments, then returns the maximum of the two resulting values. Command first evaluates its two arguments in order, and returns the first resulting value. Command setSize evaluates the second argument, then assigns the returned value to the variable specified as the first argument. Command getCurrentSize returns the current size of the variable specified as its argument. Command getMaxSize returns the maximum size of the variable specified as its argument. Command branch first evaluates the first argument from the original optimal size map, then the second argument from the original optimal size map. It then uses Algorithm 150 to keep the maximum values within the optimal size map. This behavior mirrors that of regular branch evaluation that occurs outside of a function

call. We then evaluate our return sizes m_1 and m_2 to see if they are larger than the current return size, as we must return the maximum possible return size for the function. Command loop will evaluate its arguments the number of times specified by n_1. It then evaluates its other three arguments sequentially, and once complete, returns. Command return evaluates its argument, then checks if the value returned is larger than the current return size, and if so, updates the return size. The second to last case handles when we have reached a number n and simply need to return that number. The final case handles when we call this function and the compute size list is a list, and we need to evaluate it sequentially (i.e., when called on the body of a loop or branch).

Algorithm 158 $(n_f, \theta_f, n_{ret}) = \text{EvaluateComputeList}(\overline{cl}, \theta, m)$

$(n_f, \theta_f, n_{ret}) = (0, \theta, m)$
if $(\overline{cl} = \text{add}(\overline{cl}_1, \overline{cl}_2))$ **then**
  $(n_1, \theta_1, m_1) = \text{EvaluateComputeList}(\overline{cl}_1, \theta, m)$
  $(n_2, \theta_f, n_{ret}) = \text{EvaluateComputeList}(\overline{cl}_2, \theta_1, m_1)$
  $n_f = n_1 + n_2$
else if $(\overline{cl} = \text{max}(\overline{cl}_1, \overline{cl}_2))$ **then**
  $(n_1, \theta_1, m_1) = \text{EvaluateComputeList}(\overline{cl}_1, \theta, m)$
  $(n_2, \theta_f, n_{ret}) = \text{EvaluateComputeList}(\overline{cl}_2, \theta_1, m_1)$
  $n_f = max(n_1, n_2)$
else if $(\overline{cl} = \text{first}(\overline{cl}_1, \overline{cl}_2))$ **then**
  $(n_f, \theta_1, m_1) = \text{EvaluateComputeList}(\overline{cl}_1, \theta, m)$
  $(n_2, \theta_f, n_{ret}) = \text{EvaluateComputeList}(\overline{cl}_2, \theta_1, m_1)$
else if $(\overline{cl} = \text{setSize}(x, \overline{cl}_1))$ **then**
  $(n_f, \theta_1, n_{ret}) = \text{EvaluateComputeList}(\overline{cl}_1, \theta, m)$
  $\theta_2[x \rightarrow (ty, n_{max}, n_{curr_max}, n_{curr}, \overline{cl}, \overline{x})] = \theta_1$
  $\theta_f = \theta_2[x \rightarrow (ty, n_{max}, n_{curr_max}, n_f, \overline{cl}, \overline{x})]$
else if $(\overline{cl} = \text{getCurrentSize}(x))$ **then**
  $(ty, n_{max}, n_{curr_max}, n_{curr}, \overline{cl}, \overline{x}) = \theta(x)$
  $n_f = n_{curr}$
else if $(\overline{cl} = \text{getMaxSize}(x))$ **then**
  $(ty, n_{max}, n_{curr_max}, n_{curr}, \overline{cl}, \overline{x}) = \theta(x)$
  $n_f = n_{max}$
else if $(\overline{cl} = \text{branch}(\overline{cl}_1, \overline{cl}_2))$ **then**
  $(n_1, \theta_1, m_1) = \text{EvaluateComputeList}(\overline{cl}_1, \theta, m)$
  $(n_2, \theta_2, m_2) = \text{EvaluateComputeList}(\overline{cl}_2, \theta, m)$
  $\theta_f = \text{MaxSizeMap}(\theta_1, \theta_2)$
  if $(m_1 > n_{ret}) \vee (m_2 > n_{ret})$ **then**
    if $(m_1 > m_2)$ **then**
      $n_{ret} = m_1$
    else
      $n_{ret} = m_2$
    end if
  end if
else if $(\overline{cl} = \text{loop}(n_1, \overline{cl}_1, \overline{cl}_2, \overline{cl}_3))$ **then**
  for all $(i \in \{1...n_1\})$ **do**
    $(n_2, \theta_1, m_1) = \text{EvaluateComputeList}(\overline{cl}_1, \theta, m)$
    $(n_3, \theta_2, m_2) = \text{EvaluateComputeList}(\overline{cl}_2, \theta_1, m_1)$
    $(n_4, \theta_3, m_3) = \text{EvaluateComputeList}(\overline{cl}_3, \theta_2, m_2)$
    $\theta = \theta_3$
    $m = m_3$
  end for
  $(n_f, \theta_f, n_{ret}) = (n_4, \theta, m)$
else if $(\overline{cl} = \text{return}(\overline{cl}_1))$ **then**
  $(n_f, \theta_f, m_1) = \text{EvaluateComputeList}(\overline{cl}_1, \theta, n)$
  if $m_1 > m$ **then**
    $n_{ret} = m_1$
  end if
else if $(\overline{cl} = n)$ **then**
  $n_f = n$
else if $(\overline{cl} = \overline{cl}_1 :: \overline{cl}_2)$ **then**
  $(n_1, \theta_1, m_1) = \text{EvaluateComputeList}(\overline{cl}_1, \theta, m)$
  $(n_f, \theta_f, n_{ret}) = \text{EvaluateComputeList}(\overline{cl}_2, \theta_1, m_1)$
end if
return (n_f, θ_f, n_{ret})

Algorithm 159 $(s_f) \leftarrow \text{ApplyVarSize}(s, \theta)$

```
1: s_f = skip
2: for all s_1 ∈ s do
3:     if (s_1 = ty x) then
4:         (ty, n_max, n_curr_max, n_curr, NULL, NULL) = θ(x)
5:         if (n_curr_max > n_max) then
6:             s_f = s_f; ty < n_max > x
7:         else
8:             s_f = s_f; ty < n_curr_max > x
9:         end if
10:    else if (s_1 = ty x[e]) then
11:        (ty, n_max, n_curr_max, n_curr, NULL, NULL) = θ(x)
12:        if (n_curr > τ(ty)) then
13:            s_f = s_f; ty < n_max > x[e]
14:        else
15:            s_f = s_f; ty < n_curr > x[e]
16:        end if
17:    else
18:        s_f = s_f; s_1
19:    end if
20: end for
21: return s_f
```

Algorithm 159 takes the program s and the variable size map θ as input and returns the updated program with size annotations for all variables except function variables and those which were already labeled. When the evaluated current optimal size for a given type is larger that the expected size of the type, we default to the expected size of the type stored in n_{max}. This is because certain operations, particularly multiplication, may grow the size of the variable rapidly when repeated, despite the actual data and evaluation not exceeding the expected size. It is up to the programmer to ensure that the data will not exceed the maximum expected size for a type. We add in size annotations to declarations using the current maximum size evaluated, n_{curr_max}. We do this because, in the declaration, we need to size the variable to enable the largest it could be at any given point in the program. It is possible to optimize the program beyond this, as we have done in our implementation, when the sizes are utilized with each expression and statement. This is discussed in more detail in the following section, but is beyond the scope of our current formalization.

6.2.3 Implementation

In this section, we will discuss some specifics of the proof of concept implementation. The implementation can be found at `https://github.com/amypritc/SMC2`. We implement a basic optimal variable sizing pass in our SMC^2 version of PICCO, which can be triggered using the option `-vs` when compiling the original source program using `picco`. Additionally, you can trigger the implementation to print out the table

storing the optimal variable sizing information at the end of the pass by also using the debug option `-d` when compiling the original source program using `picco`. It is important to note that we assume that all variable names are unique - implementing alpha renaming is an additional pass that is beyond the scope of this thesis.

Our basic DSL and optimal variable sizing pass implementation currently includes support for declarations, assignments, binary operations, and bounded loops with integer variables. For functions, we currently only support intraprocedural analysis, and will return the maximum size for the return type as the resulting size of a function call. In our implementation, we first added syntax for specifying a loop bound when using **`for`** loops. This syntax allows us to analyze the loop contents without needing to implement an analysis of the loop to find the loop bound. We then added elements to the AST for use within the optimal variable sizing pass, and added the optimization pass into our SMC^2 implementation after the AST is created and before it is printed out to the `.cpp` file. We chose to add this pass here as the AST is already created and can easily be iterated over, but also so this pass is separate from the final printing pass and therefore can be turned on or off as desired.

Our implementation goes beyond the formalization due to SMC^2 storing the sizes within the AST at each node and leveraging them within each individual expression and statement, as each call to an SMC protocol includes sizing. As such, we were able to store the optimal size at each expression within the AST, allowing us to achieve further optimization of the execution time than the simple variable size-per declaration substitution we show here. In order to formalize the specifics of what we were able to do with the optimization per expression, we need to formalize the AST itself by adding metadata to each expression; this is left as future work, and is beyond the scope of our current formalization.

6.2.4 Evaluation

To highlight the feasibility of our approach we provide preliminary performances numbers over both microbenchmarks that mimic behavior found in real-world SMC programs. All experiments were run in a local and distributed manner. We leverage local runs, where all participants in the SMC program execute on the same machine, to analyze overheads and benefits of our approach. We also provide distributed deployment of the same benchmarks to illustrate real-world performance. In the distributed configuration, each participant in the SMC program is executed on a separate machine. We ran our experiments using single-threaded execution on three 2.10GHz machines running Ubuntu 20.04.3 LTS. The machines were connected via 1Gbps Ethernet.

In all test programs, we use variables `a`, `b`, and `c` as private input data of a predetermined bit size, and

variables `x`, `y`, and `z` as private temporary variables to perform the computations with. We use the public variable `i` to manage the loop, which iterates 1000 times in each program. We omit input and output functions and choose to instead hard-code the input in our test programs, as we are testing the improvements of the computations that occur between parties, not local data management. We show and give a bit of description for each program below in subsection Programs.

Runtime Statistics

(a) Local

(b) Distributed

Figure 6.14: Percentage of runtime speedup for optimized variable size tests.

To calculate the averages and standard deviation, we first average the runtimes of each of the 3 parties in a single run (i.e., (Party3 + Party2 + Party1)/3). We then use the average timing for each run to obtain the total average and standard deviation for the runtime of each program. To calculate percent speedup with SMC^2 as the baseline, we used the formula: (SMC^2 avg - optimized avg)/SMC^2 avg * 100. To calculate the standard deviation error bars, we used the formula: ((SMC^2avg - (optimized avg - optimized st dev))/SMC^2 avg*100) - percent speedup. We give the runtime statistics in Table 6.29, as well as visuals of the percent speedup and standard deviation for local and distributed executions of our tests in Figure 6.14.

Test	Local					
	Regular		Optimized		Percent	Standard
	Average	St. Dev.	Average	St. Dev.	Speedup	Deviation
1	0.86641	0.02413	0.30441	0.02290	64.86516	2.64261
2	0.86510	0.02109	0.48030	0.02214	44.48040	2.55899
3	1.11485	0.02423	0.73896	0.02542	33.71633	2.28041
4	1.10950	0.02852	0.95318	0.02014	14.08949	1.81527
5	0.86783	0.02247	0.71565	0.02343	17.53571	2.70010
6	1.14385	0.02648	0.66549	0.02393	41.81957	2.09187

Test	Distributed					
	Regular		Optimized		Percent	Standard
	Average	St. Dev.	Average	St. Dev.	Speedup	Deviation
1	1.55134	0.01454	0.88648	0.00563	42.85701	0.36316
2	1.50511	0.01370	1.07687	0.00845	28.45279	0.56165
3	2.33464	0.02072	1.92087	0.01182	17.72297	0.50612
4	2.33381	0.02148	2.17152	0.01951	6.953904	0.83596
5	1.50167	0.01079	1.34208	0.01324	10.62736	0.88191
6	2.44758	0.01400	1.91382	0.00896	21.80746	0.36607

Table 6.29: Runtime statistics for each optimal variable sizing test program.

Programs

```
1 #include <stdio.h>
2
3 public int main() {
4     private int<3> a = 7, b = 3;
5     private int x, y, z;
6     public int i;
7
8     x = a;                          // x -> size = 3
9     y = b;                          // y -> size = 3
10    bound 1000
11    for(i = 0; i < 1000; i = i + 1){
12        z = x > y;                  // z -> size = 1
13    }
14    return 0;
15 }
```

Figure 6.15: Optimal Variable Sizes - Test 1

Test 1, shown in Figure 6.15, is a loop of comparisons `z = x > y`, with a computed 3 bit optimal size for both `x` and `y` and regular bit size of 32. The optimized program will compute the comparison operation over 3 bits each time versus the 32 bit comparison operation done by the unoptimized SMC^2 program, resulting in a significantly faster runtime for the optimized version. This program is representative of a program designed to securely compare two sets of data (e.g., comparing profits throughout a period of time between companies).

```
1 #include <stdio.h>
2
3 public int main() {
4     private int<12> a = 1780, b = 3456, c = 2345;
5     private int x, z;
6     public int i;
7
8     x = (a + b) >> 1;            // x -> size = (12 + 1) - 1 = 12
9     bound 1000
10    for(i = 0; i < 1000; i = i + 1){
11        z = x > c;              // z -> size = 1
12    }
13    return 0;
14 }
```

Figure 6.16: Optimal Variable Sizes - Test 2

```
1 #include <stdio.h>
2
3 public int main() {
4     private int<12> a = 1780, b = 3456, c = 2345;
5     private int x, z;
6     public int i;
7
8     bound 1000
9     for(i = 0; i < 1000; i = i + 1){
10        x = (a + b) >> 1;       // x -> size = (12 + 1) - 1 = 12
11        z = x > c;              // z -> size = 1
12    }
13    return 0;
14 }
```

Figure 6.17: Optimal Variable Sizes - Test 3

Test 2, shown in Figure 6.16, computes `x = (a + b) >> 1`, finding the optimal size of `x` to be 12 bits, then iterates over a loop of comparisons. The operations performed for the first computation of `x` will take the same time, as the unoptimized version is already set up to use the sizes that are given in the source program. The comparison operation performed in the loop will be performed on two elements of bit size 12 in the optimized program, versus two elements of bit size 32 in the unoptimized program, resulting in a decent speedup for the optimized version. This program is representative of a program designed to take the average of two data sets, and compare it to a third data set.

Test 3, shown in Figure 6.17, computes `x = (a + b) >> 1` and then compares the result of this to another input data `z = x > c` in each loop iteration. As with test 2, the optimal bit size is 12, regular bit size is 32, and this program is representative of a program designed to take the average of two data sets, and compare it to a third data set.

Test 4, shown in Figure 6.18, is like test 3, but with a larger bit length for the input data. This reduces the

```
1 #include <stdio.h>
2
3 public int main() {
4     private int<24> a = 1780, b = 3456, c = 2345;
5     private int x, z;
6     public int i;
7
8     bound 1000
9     for(i = 0; i < 1000; i = i + 1){
10        x = (a + b) >> 1;          // x -> size = (12 + 1) - 1 = 12
11        z = x > c;                 // z -> size = 1
12    }
13    return 0;
14 }
```

Figure 6.18: Optimal Variable Sizes - Test 4

```
1 #include <stdio.h>
2
3 public int main() {
4     private int<24> a = 1780, b = 3456, c = 2345;
5     private int x, z;
6     public int i;
7
8     x = (a + b) >> 1;              // x -> size = (12 + 1) - 1 = 12
9     bound 1000
10    for(i = 0; i < 1000; i = i + 1){
11        z = x > c;                 // z -> size = 1
12    }
13    return 0;
14 }
```

Figure 6.19: Optimal Variable Sizes - Test 5

difference in sizes used for the multiparty operations, but is still shown to speed up the execution time overall, indicating that even programs with data that is closer to the standard bit length of 32 can benefit from this optimization.

Test 5, shown in Figure 6.19, is like test 2 but with a larger optimal bit size of 24. We compare this to the program that will compute the comparisons with the regular 32 bit size, as comparisons will always compute over the larger bit size of the two. This program shows a lesser speedup due to the size difference between the two versions being smaller, but is still a notable speedup.

Test 6, shown in Figure 6.20, differs a bit from the previous tests in that it uses a wider variety of the operations we support for optimal variable size computation for within our initial implementation as well as varying sizes of input data. Each loop iteration will compute all of these operations, with the bit lengths of the variables changing based on what the operations are and what data they are using. This program demonstrated a decent speedup even when run distributed, showing that enabling a pass to calculate optimal

```
1 #include <stdio.h>
2
3 public int main() {
4     private int<2> a = 2;
5     private int<4> b = 14;
6     private int<6> c = 32;
7     private int x, y, z;
8     public int i;
9
10    bound 1000
11    for(i = 0; i < 1000; i = i + 1){
12        x = a << 1;              // x -> size = 2 + 1 = 3
13        z = a - c;               // z -> size = max(2, 6) + 1 = 7
14        y = x + b;               // y -> size = max(3, 4) + 1 = 5
15        y = y * b;               // y -> size = 5 + 4 = 9
16        z = z >= 0;              // z -> size = 1
17        x = x >> 1;              // x -> size = 3 - 1 = 2
18    }
19    return 0;
20 }
```

Figure 6.20: Optimal Variable Sizes - Test 6

variable sizes within multiparty computations can help to improve the runtime of general programs, not just those performing comparisons.

Runtimes

Table 6.30: Test 1 - local SMC^2

Run	Party 3	Party 2	Party 1
1	0.868508	0.870217	0.867193
2	0.874850	0.873187	0.872607
3	0.869223	0.867131	0.867753
4	0.897033	0.893162	0.891625
5	0.860548	0.859315	0.858675
6	0.872684	0.874783	0.871955
7	0.874951	0.873491	0.874809
8	0.900054	0.904284	0.899392
9	0.859444	0.861676	0.858758
10	0.908688	0.905547	0.904206
11	0.893248	0.897406	0.892067
12	0.888241	0.888868	0.893088
13	0.890421	0.886573	0.885491
14	0.833694	0.832760	0.831247
15	0.868163	0.867475	0.865815
16	0.834370	0.832191	0.831512
17	0.831957	0.833260	0.831272
18	0.897107	0.895590	0.894790
19	0.859999	0.861813	0.859240
20	0.851575	0.849424	0.848982
21	0.913960	0.917207	0.912772
22	0.886210	0.884743	0.886227
23	0.862930	0.861949	0.860915
24	0.838028	0.839736	0.837150
25	0.847045	0.848329	0.848353
26	0.835169	0.833304	0.832683
27	0.866459	0.865777	0.868828
28	0.870354	0.869182	0.868281
29	0.832932	0.834779	0.832247
30	0.902152	0.899489	0.898820
31	0.883325	0.884152	0.881424
32	0.914279	0.910649	0.908026
33	0.839794	0.840188	0.838234
34	0.857451	0.856852	0.859492
35	0.886976	0.891077	0.886385
36	0.847653	0.850521	0.846473
37	0.838673	0.841868	0.839379
38	0.885315	0.879736	0.878671
39	0.856761	0.861382	0.856135
40	0.893067	0.889413	0.887620
41	0.904329	0.902389	0.901764
42	0.878628	0.880424	0.877657
43	0.865223	0.868081	0.864067
44	0.839481	0.842249	0.838382
45	0.843150	0.840413	0.839360
46	0.863163	0.863179	0.859798
47	0.836371	0.834623	0.833766
48	0.834800	0.837129	0.834081
49	0.849037	0.847277	0.846609
50	0.844101	0.841703	0.840516

Table 6.31: Test 1 - local optimized

Run	Party 3	Party 2	Party 1
1	0.316902	0.314573	0.313774
2	0.291400	0.290817	0.288737
3	0.339659	0.336775	0.335867
4	0.281088	0.278478	0.277294
5	0.341326	0.343567	0.339636
6	0.309927	0.307017	0.305623
7	0.286355	0.283644	0.282960
8	0.313478	0.310673	0.310257
9	0.303766	0.301186	0.299949
10	0.279509	0.281267	0.278991
11	0.300836	0.297927	0.297093
12	0.303651	0.305929	0.303112
13	0.287337	0.290146	0.286221
14	0.314759	0.313020	0.310909
15	0.293442	0.296749	0.291491
16	0.325748	0.321466	0.320387
17	0.292492	0.287757	0.287108
18	0.316426	0.312256	0.310937
19	0.284641	0.281114	0.278766
20	0.294261	0.289610	0.288878
21	0.319628	0.315930	0.313814
22	0.305030	0.308295	0.303721
23	0.321093	0.315964	0.315250
24	0.301993	0.299427	0.298359
25	0.280100	0.278058	0.277412
26	0.346662	0.344792	0.344133
27	0.281928	0.280080	0.279399
28	0.278969	0.282011	0.277576
29	0.280044	0.277217	0.275869
30	0.319327	0.323368	0.318612
31	0.315627	0.312941	0.311719
32	0.293467	0.290631	0.289522
33	0.367088	0.364105	0.362910
34	0.348557	0.351443	0.347640
35	0.283320	0.281219	0.280499
36	0.336620	0.332452	0.331925
37	0.285046	0.290115	0.284388
38	0.285264	0.280049	0.281876
39	0.322377	0.324418	0.320222
40	0.329799	0.325525	0.324728
41	0.288031	0.284100	0.283454
42	0.299835	0.301847	0.299157
43	0.328249	0.324082	0.323540
44	0.282591	0.280645	0.279508
45	0.282577	0.279805	0.278787
46	0.285109	0.282144	0.280891
47	0.357979	0.353669	0.353226
48	0.315701	0.318440	0.314527
49	0.283268	0.286053	0.283028
50	0.293935	0.296638	0.292648

Table 6.32: Test 1 - distributed SMC^2

Run	Party 3	Party 2	Party 1
1	1.56247	1.56298	1.56253
2	1.56817	1.56866	1.56980
3	1.55295	1.55362	1.55455
4	1.56917	1.57010	1.57109
5	1.55180	1.55261	1.55343
6	1.54366	1.54449	1.54516
7	1.53855	1.53958	1.54049
8	1.53472	1.53539	1.53616
9	1.56714	1.56833	1.56919
10	1.56705	1.56764	1.56902
11	1.56600	1.56667	1.56766
12	1.55738	1.55820	1.55911
13	1.53885	1.53952	1.54049
14	1.54980	1.55064	1.55163
15	1.57676	1.57758	1.57871
16	1.53696	1.53795	1.53870
17	1.55868	1.55965	1.56078
18	1.55768	1.55828	1.55947
19	1.53568	1.53624	1.53721
20	1.53502	1.53561	1.53687
21	1.53450	1.53546	1.53676
22	1.55829	1.55913	1.56019
23	1.55082	1.55152	1.55259
24	1.54788	1.54895	1.54990
25	1.53058	1.53105	1.53222
26	1.55991	1.56045	1.56156
27	1.55331	1.55423	1.55527
28	1.57096	1.57131	1.57227
29	1.55150	1.55201	1.55307
30	1.52953	1.53031	1.53130
31	1.53329	1.53378	1.53503
32	1.57952	1.58035	1.58139
33	1.54421	1.54514	1.54639
34	1.57853	1.57927	1.58035
35	1.53371	1.53459	1.53572
36	1.56858	1.56910	1.56990
37	1.53881	1.53945	1.54009
38	1.53778	1.53897	1.53963
39	1.54521	1.54597	1.54686
40	1.53394	1.53478	1.53581
41	1.56852	1.56922	1.57048
42	1.54221	1.54295	1.54390
43	1.52755	1.52839	1.52928
44	1.54982	1.55078	1.55160
45	1.53677	1.53744	1.53819
46	1.54346	1.54425	1.54538
47	1.57116	1.57155	1.57269
48	1.55376	1.55443	1.55578
49	1.53479	1.53550	1.53624
50	1.54827	1.54882	1.54993

Table 6.33: Test 1 - distributed optimized

Run	Party 3	Party 2	Party 1
1	0.888133	0.889322	0.890095
2	0.896760	0.897719	0.898462
3	0.887323	0.887976	0.889139
4	0.893222	0.893890	0.894884
5	0.881776	0.882704	0.883995
6	0.882017	0.882815	0.883808
7	0.875366	0.876274	0.877364
8	0.883016	0.883834	0.884640
9	0.882106	0.883170	0.884113
10	0.885118	0.885863	0.887077
11	0.894901	0.895699	0.897047
12	0.884488	0.885167	0.886206
13	0.885675	0.885921	0.887275
14	0.885685	0.886126	0.887769
15	0.889443	0.889865	0.890731
16	0.887337	0.887879	0.888552
17	0.886668	0.887306	0.888514
18	0.885839	0.886380	0.887392
19	0.886622	0.887136	0.888198
20	0.881153	0.881767	0.882753
21	0.878778	0.879529	0.880517
22	0.877782	0.878291	0.879220
23	0.882033	0.889140	0.882909
24	0.895151	0.896564	0.897164
25	0.878688	0.879514	0.880448
26	0.888644	0.889437	0.890523
27	0.879807	0.880617	0.881216
28	0.883848	0.884417	0.885308
29	0.883657	0.884264	0.885079
30	0.891085	0.891563	0.892602
31	0.884472	0.885070	0.886397
32	0.879899	0.880921	0.881652
33	0.886781	0.887555	0.888449
34	0.894025	0.894764	0.896028
35	0.887801	0.888464	0.889384
36	0.886098	0.886895	0.887977
37	0.888262	0.889161	0.890500
38	0.884349	0.885169	0.885933
39	0.878941	0.879808	0.880863
40	0.882711	0.883658	0.884598
41	0.879849	0.880367	0.881374
42	0.879238	0.880240	0.881356
43	0.879875	0.880668	0.881953
44	0.892477	0.893193	0.894563
45	0.885873	0.886703	0.887750
46	0.882768	0.883352	0.884341
47	0.889235	0.889818	0.890736
48	0.900411	0.900939	0.902278
49	0.897040	0.897774	0.898764
50	0.878181	0.878911	0.880070

Table 6.34: Test 2 - local SMC^2

Run	Party 3	Party 2	Party 1
1	0.891668	0.895457	0.890954
2	0.837612	0.836029	0.834608
3	0.866312	0.864993	0.864400
4	0.838134	0.840096	0.837476
5	0.837876	0.836472	0.835906
6	0.883322	0.882122	0.880984
7	0.880129	0.880900	0.883350
8	0.856170	0.857407	0.855587
9	0.893019	0.895024	0.892356
10	0.835201	0.833319	0.834093
11	0.886413	0.887801	0.885241
12	0.834628	0.832663	0.831995
13	0.852675	0.853985	0.852176
14	0.882949	0.886442	0.882255
15	0.871571	0.873764	0.870298
16	0.874361	0.872413	0.871090
17	0.865299	0.863003	0.861863
18	0.831887	0.834156	0.831259
19	0.832848	0.831075	0.830451
20	0.830114	0.831934	0.829497
21	0.885133	0.881776	0.880183
22	0.865484	0.866944	0.866660
23	0.899854	0.896455	0.895623
24	0.899622	0.893325	0.892145
25	0.889801	0.890995	0.896890
26	0.867174	0.862901	0.862369
27	0.875491	0.878302	0.874499
28	0.891293	0.883357	0.882694
29	0.866069	0.868495	0.864232
30	0.859616	0.857988	0.859328
31	0.882439	0.885857	0.883820
32	0.835479	0.839104	0.834777
33	0.833541	0.837216	0.832323
34	0.873825	0.871938	0.871488
35	0.871667	0.868580	0.867569
36	0.868554	0.866412	0.865042
37	0.891310	0.896422	0.889998
38	0.870169	0.866229	0.865604
39	0.880029	0.884356	0.878724
40	0.839018	0.841884	0.839597
41	0.834652	0.836550	0.834076
42	0.872299	0.874829	0.871885
43	0.870250	0.872211	0.869632
44	0.882912	0.886191	0.881687
45	0.871254	0.873001	0.870509
46	0.893178	0.896716	0.892126
47	0.873400	0.870251	0.869720
48	0.843858	0.839630	0.838896
49	0.843337	0.842021	0.840277
50	0.869880	0.867760	0.867082

Table 6.35: Test 2 - local optimized

Run	Party 3	Party 2	Party 1
1	0.495619	0.494109	0.492748
2	0.513158	0.509342	0.508972
3	0.468367	0.471167	0.467680
4	0.458835	0.457528	0.456822
5	0.457355	0.455292	0.454611
6	0.509419	0.508929	0.507439
7	0.504641	0.502496	0.501860
8	0.458772	0.456714	0.456021
9	0.459236	0.457161	0.456431
10	0.476800	0.477208	0.474331
11	0.454545	0.453021	0.452674
12	0.516585	0.512566	0.511668
13	0.521884	0.524200	0.520041
14	0.484894	0.486298	0.486442
15	0.475277	0.471167	0.470397
16	0.467467	0.468180	0.466748
17	0.466769	0.469493	0.466255
18	0.524050	0.522608	0.524960
19	0.524314	0.527203	0.523157
20	0.484094	0.482504	0.483666
21	0.489378	0.487964	0.487649
22	0.520279	0.514315	0.513299
23	0.477554	0.475430	0.474821
24	0.456559	0.453857	0.453187
25	0.493726	0.491444	0.490810
26	0.454311	0.452419	0.451748
27	0.455815	0.459056	0.455220
28	0.479482	0.477833	0.477116
29	0.498404	0.495964	0.495301
30	0.457554	0.454648	0.453969
31	0.480113	0.478525	0.479961
32	0.478802	0.477584	0.476867
33	0.456350	0.457742	0.455572
34	0.456816	0.455396	0.454695
35	0.468368	0.466665	0.465954
36	0.505539	0.504125	0.502534
37	0.475155	0.473047	0.472358
38	0.506159	0.508229	0.504289
39	0.486110	0.483792	0.483202
40	0.498584	0.496305	0.495558
41	0.481907	0.480534	0.479864
42	0.501353	0.505257	0.500617
43	0.474980	0.472029	0.472717
44	0.501738	0.500133	0.502051
45	0.501244	0.498655	0.497886
46	0.456229	0.458014	0.455302
47	0.451765	0.453984	0.451024
48	0.467551	0.464798	0.463822
49	0.455140	0.453677	0.452781
50	0.457766	0.459186	0.457195

Table 6.36: Test 2 - distributed SMC^2

Run	Party 3	Party 2	Party 1
1	1.51566	1.51766	1.51882
2	1.51496	1.51447	1.51492
3	1.49518	1.49592	1.49717
4	1.53086	1.53127	1.53230
5	1.51167	1.51254	1.51392
6	1.48693	1.48710	1.48877
7	1.49269	1.49357	1.49504
8	1.51947	1.52018	1.52132
9	1.49216	1.49284	1.49378
10	1.52123	1.52182	1.52299
11	1.49396	1.49502	1.49580
12	1.50427	1.50492	1.50599
13	1.49408	1.49460	1.49562
14	1.49769	1.49832	1.49944
15	1.49066	1.49138	1.49263
16	1.49793	1.49878	1.49961
17	1.49736	1.49824	1.49964
18	1.53507	1.53595	1.53691
19	1.48958	1.49018	1.49160
20	1.49817	1.49891	1.50005
21	1.50293	1.50426	1.50493
22	1.49883	1.49940	1.50008
23	1.53036	1.53118	1.53198
24	1.49741	1.49799	1.49940
25	1.50686	1.50758	1.50879
26	1.49970	1.50043	1.50137
27	1.49625	1.49692	1.49808
28	1.53121	1.53192	1.53334
29	1.52713	1.52784	1.52882
30	1.49392	1.49456	1.49578
31	1.49594	1.49698	1.49770
32	1.49128	1.49248	1.49343
33	1.49758	1.49830	1.49948
34	1.49794	1.49890	1.49989
35	1.53298	1.53384	1.53507
36	1.49487	1.49561	1.49676
37	1.52171	1.52276	1.52392
38	1.51299	1.51373	1.51498
39	1.49996	1.50079	1.50201
40	1.49226	1.49258	1.49367
41	1.50749	1.50817	1.50919
42	1.50763	1.50848	1.51016
43	1.50168	1.50235	1.50353
44	1.50253	1.50335	1.50452
45	1.47927	1.47984	1.48133
46	1.49834	1.49900	1.50045
47	1.49090	1.49165	1.49294
48	1.52027	1.52106	1.52269
49	1.50915	1.50974	1.51110
50	1.49295	1.49348	1.49444

Table 6.37: Test 2 - distributed optimized

Run	Party 3	Party 2	Party 1
1	1.09731	1.09731	1.09978
2	1.07185	1.07369	1.07320
3	1.06483	1.06519	1.06611
4	1.06849	1.06926	1.07048
5	1.06788	1.06865	1.06934
6	1.07396	1.07470	1.07568
7	1.07166	1.07139	1.07346
8	1.07218	1.07299	1.07414
9	1.10020	1.10102	1.10197
10	1.07125	1.07195	1.07300
11	1.07446	1.07529	1.07583
12	1.07121	1.07190	1.07295
13	1.07659	1.07621	1.07849
14	1.09666	1.09753	1.09844
15	1.07006	1.07072	1.07189
16	1.07419	1.07488	1.07609
17	1.08297	1.08358	1.08452
18	1.07693	1.07784	1.07919
19	1.07312	1.07266	1.07499
20	1.07068	1.07141	1.07248
21	1.07638	1.07692	1.07802
22	1.08334	1.08397	1.08504
23	1.09670	1.09727	1.09790
24	1.07110	1.07174	1.07282
25	1.07558	1.07516	1.07706
26	1.09291	1.09334	1.09468
27	1.07993	1.08054	1.08150
28	1.07027	1.07097	1.07208
29	1.07025	1.07086	1.07156
30	1.07352	1.07424	1.07605
31	1.09294	1.09254	1.09468
32	1.07721	1.07802	1.07890
33	1.07357	1.07444	1.07547
34	1.07116	1.07173	1.07302
35	1.06903	1.06937	1.07019
36	1.06935	1.06980	1.07075
37	1.07477	1.07437	1.07627
38	1.07287	1.07366	1.07491
39	1.07368	1.07473	1.07557
40	1.07506	1.07574	1.07687
41	1.06533	1.06608	1.06676
42	1.07324	1.07341	1.07487
43	1.06875	1.06846	1.07084
44	1.07794	1.08001	1.08127
45	1.07959	1.08044	1.08154
46	1.07255	1.07314	1.07451
47	1.07131	1.07234	1.07267
48	1.07644	1.07746	1.07847
49	1.07984	1.07972	1.08185
50	1.07306	1.07389	1.07508

Table 6.38: Test 3 - local SMC^2

Run	Party 3	Party 2	Party 1
1	1.13104	1.12646	1.12739
2	1.15680	1.15459	1.15414
3	1.11698	1.12138	1.11623
4	1.09831	1.09497	1.09414
5	1.09055	1.09226	1.08969
6	1.10710	1.10833	1.10577
7	1.11626	1.11315	1.11264
8	1.07924	1.08091	1.08133
9	1.13216	1.13545	1.13098
10	1.15596	1.15167	1.15036
11	1.11188	1.11441	1.11242
12	1.12671	1.12228	1.12159
13	1.07009	1.07344	1.07140
14	1.07158	1.07292	1.07080
15	1.07955	1.08090	1.07828
16	1.08088	1.07787	1.07721
17	1.11884	1.11773	1.11712
18	1.14857	1.14764	1.15114
19	1.09932	1.09778	1.09703
20	1.08106	1.07972	1.07907
21	1.07726	1.07583	1.07511
22	1.12053	1.11848	1.11795
23	1.11325	1.11397	1.11590
24	1.10167	1.10250	1.10375
25	1.12213	1.12420	1.12156
26	1.14897	1.14329	1.14209
27	1.11894	1.11663	1.11695
28	1.14841	1.14416	1.14248
29	1.13874	1.13784	1.14267
30	1.09843	1.09652	1.09580
31	1.12390	1.12078	1.12006
32	1.11841	1.11718	1.11620
33	1.07838	1.08290	1.07770
34	1.11195	1.11179	1.11051
35	1.12098	1.11948	1.11756
36	1.12583	1.12992	1.12399
37	1.09419	1.09574	1.09670
38	1.10549	1.10211	1.10144
39	1.16890	1.16832	1.17053
40	1.13870	1.13648	1.13853
41	1.08814	1.08479	1.08445
42	1.13468	1.13561	1.13855
43	1.10828	1.11085	1.10770
44	1.14617	1.15138	1.14488
45	1.10681	1.10272	1.10192
46	1.12893	1.13293	1.12793
47	1.11565	1.11719	1.11498
48	1.14605	1.14446	1.14267
49	1.11582	1.11851	1.11511
50	1.13580	1.13428	1.13339

Table 6.39: Test 3 - local optimized

Run	Party 3	Party 2	Party 1
1	0.749725	0.747282	0.737363
2	0.704936	0.703478	0.702798
3	0.746494	0.744244	0.743601
4	0.732828	0.733695	0.731130
5	0.724558	0.723216	0.723846
6	0.706549	0.706009	0.709932
7	0.726216	0.727274	0.724571
8	0.756389	0.752371	0.751108
9	0.749838	0.751710	0.751220
10	0.790074	0.785912	0.786994
11	0.699078	0.702160	0.698529
12	0.800316	0.801086	0.798765
13	0.750651	0.745781	0.746460
14	0.737928	0.738909	0.743162
15	0.771391	0.772601	0.769681
16	0.771979	0.774293	0.770012
17	0.757577	0.758373	0.761176
18	0.781396	0.779110	0.778322
19	0.794284	0.792259	0.793610
20	0.706721	0.705329	0.707053
21	0.758371	0.756689	0.758872
22	0.728093	0.731170	0.727547
23	0.747503	0.744244	0.746376
24	0.749567	0.745836	0.745483
25	0.699278	0.700822	0.701375
26	0.697998	0.700160	0.698924
27	0.700039	0.698534	0.697793
28	0.745962	0.746718	0.745430
29	0.717986	0.720681	0.717331
30	0.748508	0.744532	0.743098
31	0.742194	0.743314	0.741445
32	0.749714	0.745710	0.743877
33	0.746119	0.744311	0.743649
34	0.708436	0.706298	0.705600
35	0.762285	0.762838	0.760619
36	0.740853	0.738306	0.737616
37	0.739103	0.738518	0.737016
38	0.739071	0.737283	0.736235
39	0.722407	0.724427	0.722370
40	0.718988	0.717281	0.716732
41	0.760370	0.757497	0.755713
42	0.730364	0.731880	0.732319
43	0.721416	0.720023	0.719344
44	0.705025	0.702336	0.702887
45	0.723370	0.720699	0.719873
46	0.776031	0.780165	0.774877
47	0.726915	0.723031	0.723758
48	0.760764	0.759095	0.757451
49	0.734845	0.733301	0.732466
50	0.730622	0.728696	0.728009

Table 6.40: Test 3 - distributed SMC^2

Run	Party 3	Party 2	Party 1
1	2.34157	2.34208	2.34451
2	2.34617	2.34690	2.34620
3	2.32045	2.32101	2.32148
4	2.39287	2.39366	2.39488
5	2.32415	2.32502	2.32621
6	2.32624	2.32712	2.32861
7	2.32889	2.33024	2.33115
8	2.33081	2.33159	2.33231
9	2.33214	2.33289	2.33370
10	2.35170	2.35198	2.35357
11	2.32057	2.32112	2.32206
12	2.32982	2.33051	2.33181
13	2.30757	2.30832	2.30951
14	2.34013	2.33985	2.34073
15	2.31064	2.30994	2.31034
16	2.31258	2.31154	2.31285
17	2.35282	2.35099	2.35181
18	2.30763	2.30837	2.30936
19	2.35455	2.35553	2.35633
20	2.31031	2.31123	2.31238
21	2.35540	2.35564	2.35598
22	2.34718	2.34588	2.34738
23	2.35055	2.35006	2.35115
24	2.31603	2.31503	2.31610
25	2.39928	2.39865	2.39965
26	2.33940	2.33887	2.33970
27	2.31983	2.32013	2.32049
28	2.32493	2.32564	2.32675
29	2.31282	2.31373	2.31464
30	2.34530	2.34614	2.34702
31	2.35335	2.35423	2.35486
32	2.31942	2.32031	2.32132
33	2.35079	2.34920	2.34981
34	2.33498	2.33508	2.33637
35	2.34113	2.34214	2.34325
36	2.31892	2.31841	2.31956
37	2.31702	2.31782	2.31898
38	2.32734	2.32721	2.32806
39	2.33395	2.33499	2.33474
40	2.33072	2.33108	2.33221
41	2.31627	2.31700	2.31808
42	2.32281	2.32380	2.32507
43	2.32878	2.32958	2.33088
44	2.32179	2.32268	2.32383
45	2.32473	2.32397	2.32404
46	2.38503	2.38203	2.38344
47	2.31912	2.31966	2.32074
48	2.36818	2.36755	2.36815
49	2.31127	2.31043	2.31169
50	2.33164	2.33242	2.33365

Table 6.41: Test 3 - distributed optimized

Run	Party 3	Party 2	Party 1
1	1.92617	1.92695	1.92797
2	1.92804	1.93007	1.92971
3	1.90952	1.91014	1.91156
4	1.92891	1.92965	1.93083
5	1.91378	1.91451	1.91507
6	1.91668	1.91747	1.91841
7	1.92968	1.92972	1.93112
8	1.91361	1.91456	1.91560
9	1.92246	1.92302	1.92394
10	1.92025	1.92115	1.92227
11	1.90034	1.90123	1.90189
12	1.91617	1.91680	1.91778
13	1.90689	1.90662	1.90774
14	1.91921	1.92000	1.92128
15	1.92263	1.92335	1.92441
16	1.90568	1.90639	1.90763
17	1.93163	1.93261	1.93329
18	1.91513	1.91596	1.91710
19	1.92887	1.92864	1.93036
20	1.91157	1.91241	1.91362
21	1.94057	1.94110	1.94224
22	1.91942	1.92007	1.92132
23	1.92520	1.92555	1.92654
24	1.91495	1.91569	1.91686
25	1.90756	1.90721	1.90904
26	1.90450	1.90499	1.90659
27	1.90611	1.90693	1.90799
28	1.90220	1.90280	1.90411
29	1.90632	1.90722	1.90808
30	1.93912	1.93962	1.94095
31	1.94298	1.94275	1.94451
32	1.93109	1.93192	1.93334
33	1.91412	1.91488	1.91600
34	1.92066	1.92131	1.92225
35	1.90919	1.90978	1.91065
36	1.92136	1.92206	1.92331
37	1.92156	1.92106	1.92273
38	1.92272	1.92345	1.92461
39	1.92088	1.92149	1.92287
40	1.90860	1.90943	1.91062
41	1.94716	1.94794	1.94869
42	1.90721	1.90776	1.90882
43	1.94203	1.94158	1.94338
44	1.93595	1.93694	1.93817
45	1.92647	1.92729	1.92845
46	1.90537	1.90618	1.90750
47	1.91036	1.91151	1.91163
48	1.91900	1.91967	1.92062
49	1.94040	1.94025	1.94221
50	1.92441	1.92499	1.92603

Table 6.42: Test 4 - local SMC^2

Run	Party 3	Party 2	Party 1
1	1.14050	1.14145	1.14372
2	1.14184	1.14295	1.14360
3	1.10228	1.10110	1.10255
4	1.09165	1.09420	1.09208
5	1.09887	1.10018	1.10019
6	1.08504	1.08736	1.08558
7	1.08265	1.08136	1.08256
8	1.14324	1.14115	1.13993
9	1.08061	1.07923	1.07799
10	1.08431	1.08002	1.07957
11	1.07883	1.07814	1.08076
12	1.08523	1.08674	1.08769
13	1.07967	1.08078	1.07840
14	1.14462	1.14789	1.14351
15	1.11043	1.10864	1.11479
16	1.12080	1.12201	1.11962
17	1.12393	1.12018	1.11963
18	1.10784	1.10956	1.10702
19	1.11135	1.11006	1.10961
20	1.08906	1.08697	1.08625
21	1.16146	1.15771	1.15811
22	1.09434	1.09576	1.09301
23	1.15391	1.15265	1.15454
24	1.11568	1.11788	1.11495
25	1.13141	1.13419	1.13084
26	1.18784	1.18381	1.18447
27	1.08767	1.08960	1.08829
28	1.07630	1.07498	1.07436
29	1.08241	1.08176	1.08483
30	1.09883	1.09807	1.10019
31	1.10497	1.10419	1.10664
32	1.12265	1.12516	1.12227
33	1.13964	1.13896	1.13750
34	1.07275	1.07552	1.07210
35	1.06981	1.07266	1.06927
36	1.14434	1.14695	1.14557
37	1.10440	1.10532	1.10652
38	1.08442	1.08061	1.07994
39	1.11437	1.11177	1.11107
40	1.12863	1.12742	1.12667
41	1.13241	1.13399	1.13171
42	1.12616	1.12935	1.12486
43	1.07708	1.07977	1.07637
44	1.12055	1.12154	1.11925
45	1.17454	1.17763	1.17241
46	1.13031	1.13145	1.13536
47	1.08786	1.08686	1.08514
48	1.08642	1.08283	1.08247
49	1.08813	1.08638	1.08567
50	1.08389	1.08196	1.08303

Table 6.43: Test 4 - local optimized

Run	Party 3	Party 2	Party 1
1	0.963960	0.962170	0.960005
2	0.985631	0.984903	0.988224
3	0.945429	0.947592	0.946111
4	0.926027	0.927287	0.925075
5	0.959674	0.958775	0.962850
6	0.993276	0.991902	0.991201
7	0.928095	0.931009	0.928782
8	0.985702	0.982661	0.981478
9	0.943208	0.941140	0.940420
10	0.931352	0.929193	0.928467
11	0.967272	0.965785	0.964767
12	0.980439	0.983454	0.979605
13	0.948339	0.947056	0.946437
14	0.950888	0.946514	0.945817
15	0.942699	0.940658	0.939969
16	0.947173	0.946444	0.948572
17	0.994645	0.991248	0.989158
18	0.955303	0.956593	0.954504
19	0.953356	0.954931	0.952127
20	0.927913	0.931230	0.928641
21	0.930321	0.931449	0.929479
22	0.932349	0.930838	0.930169
23	0.925005	0.928296	0.924391
24	0.964811	0.965499	0.968093
25	0.995909	0.993204	0.992175
26	0.970465	0.969733	0.966474
27	0.959858	0.963873	0.959081
28	0.973561	0.972816	0.971573
29	0.990583	0.989060	0.987512
30	0.949983	0.952631	0.949288
31	0.935877	0.939699	0.934909
32	0.926859	0.927495	0.930137
33	0.933827	0.932648	0.933760
34	0.972193	0.970662	0.968421
35	0.957156	0.960686	0.956260
36	0.947446	0.944620	0.943986
37	0.966242	0.968908	0.965326
38	0.950687	0.952124	0.950021
39	0.962127	0.961526	0.963350
40	0.939051	0.938357	0.937160
41	0.933857	0.932341	0.931688
42	0.931141	0.932175	0.930209
43	0.982224	0.981355	0.980709
44	0.948196	0.945729	0.945031
45	0.952272	0.953601	0.953493
46	0.927521	0.928916	0.926907
47	0.933172	0.937676	0.932528
48	0.960724	0.961954	0.963497
49	0.934639	0.932648	0.933803
50	0.961133	0.958237	0.956127

Table 6.44: Test 4 - distributed SMC^2

Run	Party 3	Party 2	Party 1
1	2.32823	2.33030	2.33040
2	2.38042	2.37808	2.38094
3	2.32755	2.32576	2.32708
4	2.38298	2.38165	2.38213
5	2.31284	2.31172	2.31346
6	2.32917	2.32855	2.32916
7	2.35747	2.35778	2.35888
8	2.34597	2.34698	2.34781
9	2.33942	2.34021	2.34164
10	2.31390	2.31466	2.31535
11	2.33936	2.33981	2.34126
12	2.33658	2.33713	2.33814
13	2.31769	2.31612	2.31734
14	2.31792	2.31871	2.31965
15	2.33204	2.33168	2.33267
16	2.32898	2.32956	2.33003
17	2.39794	2.39773	2.40018
18	2.33488	2.33413	2.33548
19	2.31274	2.31142	2.31212
20	2.31331	2.31267	2.31353
21	2.34999	2.34860	2.34996
22	2.34677	2.34598	2.34642
23	2.32436	2.32438	2.32581
24	2.33420	2.33489	2.33627
25	2.33085	2.33098	2.33212
26	2.36378	2.36486	2.36590
27	2.31631	2.31560	2.31647
28	2.31013	2.30961	2.30994
29	2.31442	2.31208	2.31391
30	2.30976	2.30849	2.30964
31	2.31711	2.31725	2.31817
32	2.34696	2.34762	2.34873
33	2.32492	2.32575	2.32677
34	2.32810	2.32855	2.32903
35	2.32383	2.32166	2.32324
36	2.34621	2.34705	2.34791
37	2.32196	2.32052	2.32179
38	2.31099	2.31028	2.31128
39	2.31106	2.30892	2.30977
40	2.35638	2.35478	2.35518
41	2.30796	2.30643	2.30838
42	2.33201	2.33116	2.33225
43	2.36732	2.36729	2.36830
44	2.35696	2.35747	2.35854
45	2.32039	2.32104	2.32239
46	2.32113	2.32220	2.32251
47	2.35099	2.35133	2.35285
48	2.31744	2.31562	2.31676
49	2.30828	2.30852	2.30978
50	2.36282	2.36380	2.36465

Table 6.45: Test 4 - distributed optimized

Run	Party 3	Party 2	Party 1
1	2.16087	2.16120	2.16328
2	2.16271	2.16480	2.16423
3	2.16832	2.16896	2.16999
4	2.25211	2.25288	2.25388
5	2.17003	2.17078	2.17155
6	2.15817	2.15885	2.16000
7	2.18712	2.18670	2.18895
8	2.15347	2.15427	2.15533
9	2.17009	2.17085	2.17191
10	2.14256	2.14343	2.14462
11	2.18716	2.18807	2.18862
12	2.18100	2.18163	2.18316
13	2.14788	2.14770	2.14925
14	2.19296	2.19366	2.19479
15	2.18750	2.18848	2.18948
16	2.16045	2.16090	2.16209
17	2.15214	2.15246	2.15298
18	2.14776	2.14845	2.14965
19	2.16438	2.16416	2.16557
20	2.17230	2.17316	2.17425
21	2.14423	2.14478	2.14598
22	2.17753	2.17826	2.17913
23	2.16021	2.16090	2.16128
24	2.16306	2.16364	2.16487
25	2.18780	2.18771	2.18945
26	2.20773	2.20842	2.20969
27	2.14096	2.14162	2.14286
28	2.15794	2.15869	2.15955
29	2.15423	2.15505	2.15550
30	2.16829	2.16915	2.17042
31	2.15761	2.15751	2.15904
32	2.18137	2.18205	2.18293
33	2.17177	2.17280	2.17386
34	2.15927	2.15984	2.16101
35	2.18857	2.18950	2.19014
36	2.17764	2.17841	2.17958
37	2.16800	2.16763	2.16904
38	2.14784	2.14851	2.14951
39	2.16987	2.17070	2.17245
40	2.16209	2.16276	2.16425
41	2.16073	2.16132	2.16214
42	2.19175	2.19226	2.19335
43	2.18383	2.18360	2.18532
44	2.16067	2.16179	2.16294
45	2.19754	2.19851	2.19940
46	2.19184	2.19253	2.19384
47	2.18606	2.18714	2.18810
48	2.16822	2.16899	2.16991
49	2.16132	2.16222	2.16324
50	2.16945	2.17041	2.17158

Table 6.46: Test 5 - local SMC^2

Run	Party 3	Party 2	Party 1
1	0.892470	0.895347	0.891705
2	0.833727	0.835280	0.835335
3	0.855584	0.856775	0.854182
4	0.867249	0.864978	0.864362
5	0.897705	0.900824	0.896875
6	0.868038	0.866599	0.865972
7	0.861226	0.862524	0.863180
8	0.887344	0.889874	0.889618
9	0.886904	0.888886	0.885624
10	0.842471	0.840115	0.839433
11	0.866403	0.867315	0.869224
12	0.892790	0.888952	0.888159
13	0.882539	0.883543	0.886474
14	0.856526	0.854605	0.853987
15	0.887616	0.884308	0.883890
16	0.842425	0.844461	0.844648
17	0.852887	0.854115	0.852196
18	0.889793	0.893151	0.889282
19	0.842582	0.840691	0.840047
20	0.862042	0.862482	0.860162
21	0.837995	0.833866	0.832561
22	0.838093	0.836227	0.837787
23	0.832771	0.834211	0.831436
24	0.839837	0.842256	0.838367
25	0.888654	0.891839	0.887565
26	0.852701	0.850927	0.851763
27	0.859602	0.861661	0.859001
28	0.833389	0.831541	0.832588
29	0.833458	0.831986	0.831403
30	0.898285	0.899099	0.896670
31	0.849426	0.850644	0.848649
32	0.898630	0.896722	0.895458
33	0.852434	0.849639	0.849151
34	0.889374	0.887833	0.887024
35	0.863549	0.864679	0.861553
36	0.887668	0.889025	0.884501
37	0.904637	0.902803	0.901631
38	0.892043	0.890019	0.888128
39	0.864321	0.867141	0.867441
40	0.854694	0.850335	0.849783
41	0.906781	0.905945	0.906542
42	0.888565	0.886491	0.885052
43	0.852559	0.850932	0.850364
44	0.899600	0.896184	0.894988
45	0.864524	0.863166	0.862519
46	0.878380	0.875936	0.875385
47	0.875955	0.878630	0.879123
48	0.903071	0.899592	0.898268
49	0.857422	0.856012	0.855269
50	0.853933	0.855735	0.853298

Table 6.47: Test 5 - local optimized

Run	Party 3	Party 2	Party 1
1	0.736036	0.730749	0.729971
2	0.721534	0.719451	0.718763
3	0.713156	0.714375	0.712514
4	0.718496	0.720393	0.720496
5	0.682949	0.681676	0.682395
6	0.681737	0.678549	0.677906
7	0.683124	0.684245	0.681822
8	0.732746	0.730653	0.732052
9	0.728249	0.724026	0.723448
10	0.733078	0.727948	0.726870
11	0.709209	0.710666	0.707377
12	0.728851	0.723565	0.722287
13	0.698899	0.701305	0.700354
14	0.700398	0.698131	0.697714
15	0.747874	0.748162	0.744627
16	0.716000	0.714702	0.714102
17	0.698450	0.695898	0.695187
18	0.731568	0.732244	0.736617
19	0.731647	0.728118	0.726946
20	0.702104	0.703471	0.701441
21	0.697449	0.699993	0.696792
22	0.729554	0.728063	0.726087
23	0.733278	0.732157	0.729438
24	0.749312	0.744658	0.743504
25	0.719514	0.716835	0.716135
26	0.762853	0.764313	0.760733
27	0.766196	0.766100	0.764594
28	0.716541	0.714209	0.715894
29	0.683019	0.680755	0.681443
30	0.686507	0.690430	0.685822
31	0.735601	0.735225	0.732116
32	0.718460	0.714675	0.713873
33	0.693039	0.690191	0.687959
34	0.684695	0.681397	0.680996
35	0.719314	0.718024	0.717308
36	0.737686	0.735225	0.733510
37	0.686820	0.684115	0.683769
38	0.738299	0.739696	0.737029
39	0.705827	0.707357	0.705168
40	0.677695	0.679898	0.677089
41	0.711069	0.708383	0.707347
42	0.686284	0.682201	0.681564
43	0.727157	0.726620	0.723332
44	0.723944	0.718363	0.719211
45	0.697799	0.699266	0.697034
46	0.683694	0.683417	0.682334
47	0.733015	0.731555	0.729633
48	0.762123	0.766315	0.760889
49	0.736797	0.736410	0.733260
50	0.745671	0.739517	0.741467

Table 6.48: Test 5 - distributed SMC^2

Run	Party 3	Party 2	Party 1
1	1.50233	1.50406	1.50519
2	1.51322	1.51266	1.51341
3	1.49658	1.49737	1.49894
4	1.49497	1.49559	1.49666
5	1.49201	1.49217	1.49276
6	1.49784	1.49862	1.49955
7	1.49530	1.49597	1.49675
8	1.50136	1.50231	1.50319
9	1.49230	1.49269	1.49361
10	1.48558	1.48623	1.48733
11	1.49216	1.49117	1.49202
12	1.48718	1.48789	1.48873
13	1.51596	1.51629	1.51720
14	1.52567	1.52643	1.52819
15	1.51915	1.52036	1.52139
16	1.50638	1.50723	1.50840
17	1.49647	1.49653	1.49766
18	1.50987	1.51032	1.51169
19	1.48457	1.48506	1.48588
20	1.50407	1.50489	1.50587
21	1.49892	1.49955	1.50061
22	1.49456	1.49530	1.49636
23	1.50508	1.50534	1.50646
24	1.52899	1.52875	1.53091
25	1.49403	1.49415	1.49485
26	1.52848	1.52901	1.53014
27	1.50606	1.50680	1.50800
28	1.49459	1.49518	1.49635
29	1.48860	1.48873	1.48953
30	1.49281	1.49337	1.49455
31	1.50227	1.50232	1.50365
32	1.48761	1.48829	1.48959
33	1.49310	1.49371	1.49514
34	1.51146	1.51262	1.51348
35	1.50241	1.50311	1.50397
36	1.49842	1.49911	1.50046
37	1.50712	1.50796	1.50914
38	1.50118	1.50175	1.50287
39	1.48310	1.48408	1.48509
40	1.48932	1.48894	1.49008
41	1.49680	1.49764	1.49874
42	1.49766	1.49857	1.49988
43	1.51758	1.51825	1.51981
44	1.49837	1.49919	1.50049
45	1.49941	1.50035	1.50165
46	1.50837	1.50836	1.50921
47	1.50181	1.50278	1.50347
48	1.50284	1.50356	1.50476
49	1.50048	1.50127	1.50251
50	1.50201	1.50242	1.50353

Table 6.49: Test 5 - distributed optimized

Run	Party 3	Party 2	Party 1
1	1.34574	1.34560	1.34685
2	1.33227	1.33118	1.33303
3	1.32883	1.32945	1.33093
4	1.33213	1.33333	1.33425
5	1.32342	1.32390	1.32484
6	1.33163	1.33228	1.33335
7	1.33846	1.33852	1.33999
8	1.34807	1.34831	1.34933
9	1.34363	1.34419	1.34523
10	1.33759	1.33823	1.33923
11	1.38520	1.38551	1.38678
12	1.34487	1.34567	1.34708
13	1.34419	1.34292	1.34420
14	1.34043	1.34098	1.34213
15	1.33436	1.33504	1.33601
16	1.34544	1.34604	1.34746
17	1.37375	1.37374	1.37526
18	1.35436	1.35450	1.35595
19	1.33324	1.33361	1.33474
20	1.33281	1.33416	1.33482
21	1.33664	1.33753	1.33846
22	1.36485	1.36574	1.36688
23	1.34787	1.34818	1.34896
24	1.33784	1.33827	1.33935
25	1.33815	1.33841	1.33986
26	1.34044	1.34151	1.34231
27	1.33625	1.33701	1.33787
28	1.33553	1.33609	1.33713
29	1.33751	1.33784	1.33899
30	1.33379	1.33486	1.33578
31	1.33455	1.33504	1.33573
32	1.36299	1.36372	1.36466
33	1.35423	1.35490	1.35626
34	1.32149	1.32216	1.32327
35	1.32743	1.32813	1.32914
36	1.34208	1.34342	1.34383
37	1.34023	1.34047	1.34146
38	1.34780	1.34851	1.34957
39	1.38335	1.38417	1.38529
40	1.33596	1.33669	1.33775
41	1.33653	1.33636	1.33760
42	1.33177	1.33270	1.33386
43	1.33815	1.33854	1.33959
44	1.34221	1.34281	1.34373
45	1.33118	1.33178	1.33291
46	1.33472	1.33549	1.33658
47	1.33731	1.33776	1.33893
48	1.33468	1.33520	1.33651
49	1.34473	1.34553	1.34621
50	1.32774	1.32841	1.32949

Table 6.50: Test 6 - local SMC^2

Run	Party 3	Party 2	Party 1
1	1.16664	1.16518	1.16444
2	1.14848	1.14542	1.14473
3	1.12920	1.12790	1.12724
4	1.14700	1.15026	1.14634
5	1.14949	1.14653	1.14536
6	1.13827	1.13875	1.13875
7	1.12975	1.12774	1.12836
8	1.17780	1.18107	1.17696
9	1.11726	1.11896	1.11583
10	1.11044	1.11317	1.10981
11	1.14446	1.14780	1.14375
12	1.22460	1.22062	1.22018
13	1.13542	1.13740	1.13486
14	1.23013	1.22703	1.22637
15	1.17713	1.17488	1.17412
16	1.13458	1.13835	1.13375
17	1.16491	1.17090	1.16409
18	1.14309	1.14486	1.14237
19	1.13502	1.13366	1.13563
20	1.14653	1.14955	1.14621
21	1.15262	1.14823	1.14885
22	1.16810	1.16762	1.16988
23	1.13800	1.14003	1.13744
24	1.14976	1.14572	1.14496
25	1.15848	1.15350	1.15275
26	1.14047	1.13858	1.13799
27	1.13980	1.14176	1.14209
28	1.13065	1.12716	1.12657
29	1.17034	1.16716	1.16627
30	1.14423	1.14078	1.14011
31	1.12967	1.13234	1.12906
32	1.13539	1.13400	1.13617
33	1.17303	1.17423	1.17292
34	1.18115	1.18464	1.18036
35	1.13757	1.13975	1.13699
36	1.12964	1.13107	1.12904
37	1.11319	1.11541	1.11255
38	1.10777	1.10821	1.11027
39	1.15285	1.14835	1.14754
40	1.16331	1.16637	1.16259
41	1.14442	1.14234	1.14172
42	1.11005	1.10877	1.10818
43	1.11719	1.11434	1.11389
44	1.14323	1.14378	1.14551
45	1.13994	1.14170	1.13946
46	1.10517	1.10450	1.10712
47	1.10853	1.10662	1.10597
48	1.10024	1.10215	1.09969
49	1.17347	1.16918	1.16814
50	1.11423	1.11176	1.11099

Table 6.51: Test 6 - local optimized

Run	Party 3	Party 2	Party 1
1	0.668696	0.666209	0.665564
2	0.685263	0.681641	0.680907
3	0.672267	0.669501	0.670556
4	0.735243	0.733186	0.732244
5	0.638218	0.633990	0.633178
6	0.724135	0.719776	0.718991
7	0.653292	0.655395	0.652645
8	0.634857	0.632186	0.631651
9	0.666026	0.664104	0.663474
10	0.653798	0.655012	0.653104
11	0.639681	0.641762	0.639052
12	0.679118	0.674335	0.674924
13	0.651690	0.650193	0.652134
14	0.657898	0.655452	0.654858
15	0.648569	0.650179	0.647951
16	0.665692	0.667294	0.664706
17	0.646767	0.648440	0.646160
18	0.684404	0.687578	0.683757
19	0.658656	0.654617	0.653919
20	0.674990	0.669710	0.668653
21	0.689790	0.693213	0.689064
22	0.691946	0.687771	0.687022
23	0.670328	0.671618	0.671269
24	0.630657	0.634741	0.629975
25	0.665263	0.667378	0.664850
26	0.670449	0.667702	0.667017
27	0.666294	0.671466	0.665454
28	0.670438	0.668684	0.667385
29	0.648765	0.645800	0.644967
30	0.657960	0.656742	0.655979
31	0.675366	0.671561	0.670597
32	0.677816	0.677278	0.679696
33	0.683596	0.687985	0.682798
34	0.653160	0.651835	0.650388
35	0.685180	0.680935	0.680290
36	0.661336	0.661595	0.659617
37	0.663334	0.662444	0.661038
38	0.653121	0.650411	0.649726
39	0.716883	0.714626	0.716196
40	0.631997	0.630698	0.629958
41	0.692422	0.689224	0.688625
42	0.663466	0.664746	0.662484
43	0.679765	0.678825	0.676976
44	0.631733	0.630551	0.631419
45	0.631596	0.630283	0.630775
46	0.664866	0.662989	0.662256
47	0.722124	0.716621	0.715835
48	0.668908	0.667685	0.666424
49	0.641249	0.639734	0.639174
50	0.632310	0.630365	0.631098

Table 6.52: Test 6 - distributed SMC^2

Run	Party 3	Party 2	Party 1
1	2.45985	2.46092	2.46132
2	2.43933	2.44018	2.44138
3	2.43015	2.43108	2.43199
4	2.45521	2.45592	2.45730
5	2.44059	2.44132	2.44251
6	2.45002	2.45078	2.45191
7	2.47251	2.47221	2.47349
8	2.45556	2.45614	2.45721
9	2.44673	2.44744	2.44863
10	2.43473	2.43561	2.43661
11	2.46897	2.46971	2.47042
12	2.43704	2.43777	2.43884
13	2.44535	2.44511	2.44627
14	2.44383	2.44428	2.44508
15	2.42411	2.42483	2.42571
16	2.44077	2.44144	2.44250
17	2.46005	2.46093	2.46188
18	2.45163	2.45237	2.45313
19	2.46712	2.46657	2.46776
20	2.43416	2.43477	2.43561
21	2.43222	2.43286	2.43380
22	2.44419	2.44490	2.44612
23	2.45695	2.45778	2.45911
24	2.43872	2.43959	2.44048
25	2.43462	2.43418	2.43564
26	2.43036	2.43163	2.43272
27	2.45114	2.45191	2.45297
28	2.47535	2.47700	2.47811
29	2.44900	2.44986	2.45120
30	2.44047	2.44106	2.44226
31	2.46736	2.46715	2.46796
32	2.44109	2.44161	2.44285
33	2.43985	2.44042	2.44207
34	2.44044	2.44117	2.44256
35	2.43048	2.43113	2.43250
36	2.43567	2.43666	2.43782
37	2.44844	2.44800	2.44904
38	2.43249	2.43311	2.43402
39	2.44281	2.44325	2.44436
40	2.44514	2.44641	2.44721
41	2.43241	2.43303	2.43412
42	2.42968	2.43051	2.43130
43	2.42994	2.42956	2.43077
44	2.45376	2.45447	2.45539
45	2.46219	2.46299	2.46423
46	2.47630	2.47688	2.47813
47	2.44165	2.44234	2.44338
48	2.44657	2.44706	2.44832
49	2.45015	2.44980	2.45098
50	2.48423	2.48483	2.48561

Table 6.53: Test 6 - distributed optimized

Run	Party 3	Party 2	Party 1
1	1.92887	1.93075	1.93074
2	1.90686	1.90763	1.90855
3	1.90282	1.90348	1.90463
4	1.90558	1.90633	1.90777
5	1.92090	1.92151	1.92251
6	1.92509	1.92570	1.92663
7	1.91017	1.91089	1.91178
8	1.91338	1.91380	1.91467
9	1.91475	1.91506	1.91621
10	1.92603	1.92709	1.92804
11	1.92604	1.92678	1.92787
12	1.92646	1.92717	1.92858
13	1.91228	1.91296	1.91429
14	1.90709	1.90760	1.90863
15	1.90773	1.90835	1.90945
16	1.91053	1.91122	1.91253
17	1.91833	1.91902	1.92045
18	1.91418	1.91512	1.91604
19	1.90413	1.90483	1.90588
20	1.90657	1.90731	1.90848
21	1.90966	1.91067	1.91178
22	1.91764	1.91849	1.91933
23	1.90376	1.90416	1.90549
24	1.92051	1.92149	1.92234
25	1.91817	1.91909	1.92024
26	1.91829	1.91887	1.92014
27	1.90197	1.90280	1.90409
28	1.92109	1.92172	1.92316
29	1.93152	1.93243	1.93327
30	1.89837	1.89907	1.90001
31	1.90204	1.90287	1.90375
32	1.90164	1.90224	1.90352
33	1.90106	1.90179	1.90293
34	1.90457	1.90523	1.90598
35	1.91030	1.91117	1.91226
36	1.90379	1.90453	1.90551
37	1.90672	1.90715	1.90804
38	1.90600	1.90672	1.90783
39	1.90323	1.90382	1.90493
40	1.91400	1.91466	1.91569
41	1.90675	1.90749	1.90882
42	1.91558	1.91667	1.91762
43	1.91138	1.91241	1.91310
44	1.90995	1.91081	1.91177
45	1.92498	1.92569	1.92700
46	1.91269	1.91348	1.91427
47	1.92193	1.92250	1.92348
48	1.91201	1.91308	1.91401
49	1.91573	1.91644	1.91777
50	1.93532	1.93604	1.93687

7 Conclusion

In this thesis, we have presented the progression of a formal model for a general SMC compiler. We started from Basic SMC^2 in Chapter 3, which supported the basic SMC concepts for general purpose programs in C. We extended this model to Location-tracking SMC^2 in Chapter 4, providing full support for the C language This lead to our final model, Multiparty SMC^2, in Chapter 5, a fully comprehensive, multiparty formal model supporting both safe and unsafe features of C in a more efficient manner. Our model does not artificially restrict what C features can be present in private branches – restrictions are instead guided by which operations our model has shown to be unsafe. We provide support through additional tracking meta-data to enable further general-purpose features, such as pointers, which are unsafe in current SMC techniques. The intuition, shown in our motivation, is that state-of-the-art SMC techniques cannot track complex memory indirections that can occur when using pointers. By providing this tracking, we have shown that these operations can be made safe.

We provide an implementation of our model in PICCO in Chapter 6 (Section 6.1), modifying it's style of single-statement resolution to reflect our optimized conditional code block tracking scheme, and show an improvement in execution runtime over microbenchmarks and maintenance of the anticipated runtime in real-world programs. We further extend our model to include a DSL in Chapter 6 (Section 6.2), formalizing the syntax of the DSL, the algorithms for use in the optimization to find optimal variable sizes, and the translation from the DSL and optimization to Multiparty SMC^2 semantics. We provide a basic implementation of our DSL as an optimization pass on top of our Multiparty SMC^2 implementation in PICCO, with initial benchmarks showing that this optimization can be useful in improving the runtime of programs in several cases. With all of these elements, we have successfully created an efficient formal model for general-purpose SMC that allows the programmer many freedoms in writing programs and the implementor ease in extending the model to include new features and optimizations, all while maintaining correctness and security.

As future work, our model can be extended to support explicit declassification, through a primitive PICCO calls `smcopen`. Consider Figure 7.1, which highlights a modification to our original gender based salary computation (lines 16-17) from Figure 2.1. Explicitly declassifying the sum and count earlier in the program,

```
22 avgFemaleSalPub=smcopen(avgFemaleSalary);
23 femaleCountPub=smcopen(femaleCount);
24 avgMaleSalPub=smcopen(avgMaleSalary); maleCountPub=smcopen(maleCount);
25 avgFemaleSalPub=(avgFemaleSalPub/femaleCountPub)/2+historicFemaleSalAvg/2;
26 avgMaleSalPub=(avgMaleSalPub/maleCountPub)/2+historicMaleSalAvg/2;
27
28 for (i=1; i<numParticipants+1; i++){
29   smcoutput(avgFemaleSalPub, i);
30   smcoutput(avgMaleSalPub, i);
31 }
```

Figure 7.1: Securely calculating the gender pay gap for 100 organizations with additional information released.

allows us to change the average computation to a public computation. This reduces the number of high cost communications and cryptographic computations in the program. To support explicit declassification in our model we would need to extend our semantics with gradual release [47].

As for future work for the DSL, there are many possibilities, some of which we will discuss here. Interprocedural analysis for functions is a non-trivial extension of the DSL implementation, but was not essential in the first round of implementation to show that the DSL and this particular optimization can be of great value when it comes to creating more efficient SMC programs. Implementing interprocedural analysis as we have planned in our formalization above would be a next logical step in improving the implementation.

The implementation of support for branches involves creating a copy of the variables whose sizes are modified within the evaluation of the **`then`** branch. Then if there is an **`else`** branch, evaluating the **`else`** branch can be done as normal. Next, you will iterate through the copies that were changed within the **`then`** branch, updating the variable table to store the maximum resulting value for each of the variables, as we have formalized in Algorithm 150.

The implementation of support for arrays involves ensuring that the size expected of an element of the array is passed along properly. This simple implementation will not result in efficiency on its own, but will need to be extended to take into consideration things like loops where we modify a different index of the array within each loop iteration, and therefore may need to only update the size of the array once. Such a loop that is not inter-dependent on elements of the array (e.g., does not perform swapping or combination of array elements) may also be parallelizable; the analysis of such elements may benefit from the introduction of a `foreach` loop or some such syntax to indicate that it likely falls into this case. However, this is optimization requires further analysis beyond what we do here, and is therefore out of scope of this thesis.

Adding support for the dot product operator `@` used in the Hamming distance example shown in Figure 6.12 involves looking up the length of the arrays. Once you know the length `n` of the arrays, the optimal size for the result of this operation is `log(n)`, as the arrays should both be of the same length. This is not currently supported within our formalization of the DSL, but could easily be added by extending the optimal size map θ to include the metadata for length, and then Algorithm 148 would need to be extended to use the expression parameter to find the length of the arrays that we are performing this operation on.

Another interesting DSL feature that loop bounds can enable is the ability to use private conditions within loops - the loop would essentially then run for "bound" number of iterations and have a private-conditioned branching statement on whether to keep the results of executing the loop in that iteration. While this is not essential functionality and can be bypassed with the programmer writing the loop as such with a public-conditioned outer loop and a private-conditioned inner branch, it would allow the programmer to write more expressive programs without having to think of how to work around only allowing public-conditioned loops.

Bibliography

[1] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[2] A. Yao, "How to generate and exchange secrets," in *IEEE Symposium on Foundations of Computer Science*, 1986, pp. 162–167.

[3] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," vol. 31, no. 4, 1985, pp. 469–472.

[4] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology – EUROCRYPT*, 1999, pp. 223–238.

[5] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, "SoK: General purpose compilers for secure multi-party computation," in *IEEE Symposium on Security and Privacy (S&P)*, 2019, pp. 1220–1237.

[6] B. Mood, D. Gupta, H. Carter, K. Butler, and P. Traynor, "Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation," in *IEEE European Symposium on Security and Privacy (Euro S&P)*, 2016.

[7] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay – Secure two-party computation system," in *USENIX Security Symposium*, 2004.

[8] W. Henecka, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "TASTY: Tool for automating secure two-party computations," in *ACM Conference on Computer and Communications Security (CCS)*, 2010, pp. 451–462.

[9] D. Demmler, T. Schneider, and M. Zohner, "ABY – A framework for efficient mixed-protocol secure two-party computation," in *Network and Distributed System Secuity Symposium (NDSS)*, 2015.

[10] B. Kreuter, a. shelat, B. Mood, and K. Butler, "PCF: A portable circuit format for scalable two-party secure computation," in *USENIX Security Symposium*, 2013, pp. 321–336.

[11] E. Songhori, S. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, "TinyGarble: Highly compressed and scalable sequential garbled circuits," in *IEEE Symposium on Security and Privacy*, 2015, pp. 411–428.

[12] C. Liu, Y. Huang, E. Shi, J. Katz, and M. Hicks, "Automating efficient RAM-model secure computation," in *IEEE Symposiym on Security and Privacy (S&P)*, 2014, pp. 623–638.

[13] C. Liu, X. Wang, K. Nayak, Y. Huang, and E. Shi, "ObliVM: A programming framework for secure computation," in *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[14] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *European Symposium on Research in Computer Security (ESORICS)*, 2008, pp. 192–206.

[15] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP: A system for secure multi-party computation," in *ACM Conference on Computer and Communications Security (CCS)*, 2008, pp. 257–266.

[16] I. Damgård, M. Geisler, M. Krøigaard, and J. Nielsen, "Asynchronous multiparty computation: Theory and implementation," in *Public Key Cryptography (PKC)*, 2009, pp. 160–179.

[17] R. Bendlin, I. Damgard, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *Advances in Cryptography – EUROCRYPT*, 2011, pp. 169–188.

[18] I. Damgard, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology – CRYPTO*, 2012, pp. 643–662.

[19] J. Nielsen, P. Nordholt, C. Orlandi, and S. Burra, "A new approach to practical active-secure two-party computation," in *Advances in Cryptology – CRYPTO*, 2012, pp. 681–700.

[20] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith, "Secure two-party computations in ANSI C," in *ACM Conference on Computer and Communications Security (CCS)*, 2012, pp. 772–783.

[21] Y. Zhang, A. Steele, and M. Blanton, "PICCO: A general-purpose compiler for private distributed computation," in *ACM Conference on Computer and Communications Security (CCS)*, 2013, pp. 813–826.

[22] Y. Zhang, M. Blanton, and G. Almashaqbeh, "Implementing support for pointers to private data in a general-purpose secure multi-party compiler," *ACM Transactions on Privacy and Security (TOPS)*, vol. 21, no. 2, 2018.

[23] V. Sokk, "An improved type system for a privacy-aware programming language and its practical applications," Master's thesis, University of Tartu, 2016.

[24] A. Rastogi, M. Hammer, and M. Hicks, "Wysteria: A programming language for generic, mixed-mode multiparty computations," in *IEEE Symposium on Security and Privacy*, 2014, pp. 655–670.

[25] A. Rastogi, N. Swamy, and M. Hicks, "WYS*: A DSL for verified secure multi-party computations," in *International Conferece on Principles of Security and Trust (POST)*, 2019, pp. 99–122.

[26] M. Pettai and P. Laud, "Automatic proofs of privacy of secure multi-party computation protocols against active adversaries," in *IEEE Computer Security Foundations Symposium (CSF)*, 2015, pp. 75–89.

[27] R. Jagomägis, "SecreC: A privacy-aware programming language with applications in data mining," Master's thesis, University of Tartu, 2010.

[28] D. Volpano and G. Smith, "A type-based approach to program security," in *International Joint Conference CAAP/FASE on Theory and Practice of Software Development (TAPSOFT)*, 1997, pp. 607–621.

[29] M. Abadi, A. Banerjee, N. Heintze, and J. Riecke, "A core calculus of dependency," in *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 1999, pp. 147–160.

[30] T. Austin and C. Flanagan, "Efficient purely-dynamic information flow analysis," in *Workshop on Programming Languages and Analysis for Security (PLAS)*, 2009, pp. 113–124.

[31] A. Sabelfeld and A. Russo, "From dynamic to static and back: Riding the roller coaster of information-flow control research," in *Perspectives of Systems Informatics (PSI)*, 2009, pp. 352–365.

[32] D. Devriese and F. Piessens, "Noninterference through secure multi-execution," in *IEEE Symposium on Security and Privacy (S&P)*, 2010, pp. 109–124.

[33] B. Ford, "Plugging side-channel leaks with timing information flow control," in *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2012.

[34] J. Planul and J. Mitchell, "Oblivious program execution and path-sensitive non-interference," in *IEEE Computer Security Foundations Symposium (CSF)*, 2013, pp. 66–80.

[35] J. Mitchell, R. Sharma, D. Stefan, and J. Zimmerman, "Information-flow control for programming on encrypted data," in *IEEE Computer Security Foundations Symposium (CSF)*, 2012, pp. 45–60.

[36] P. Laud and A. Pankova, "Optimizing secure computation programs with private conditionals," in *International Conference on Information and Communications Security (ICICS)*, 2016, pp. 418–430.

[37] M. Patrignani and D. Garg, "Secure compilation and hyperproperty preservation," in *IEEE Computer Security Foundations Symposium (CSF)*, 2017, pp. 392–404.

[38] C. Abate, R. Blanco, Ş. Ciobâcă, A. Durier, D. Garg, C. Hritcu, M. Patrignani, É. Tanter, and J. Thibault, "Trace-relating compiler correctness and secure compilation," in *European Symposium on Programming(ESOP)*, 2020, pp. 1–28.

[39] A. Lapets, F. Jansen, K. Albab, R. Issa, L. Qin, M. Varia, and A. Bestavros, "Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities," in *ACM SIGCAS Conference on Computing and Sustainable Societies (COMPASS)*, 2018.

[40] X. Leroy, A. Appel, S. Blazy, and G. Stewart, "The CompCert memory model, Version 2," INRIA, Tech. Rep. RR-7987, 2012.

[41] R. Gennaro, M. Rabin, and T. Rabin, "Simplified VSS and fast-track multiparty computations with applications to threshold cryptography," in *ACM Symposium on Principles of Distributed Computing (PODC)*, 1998, pp. 101–111.

[42] D. Octeau, D. Luchaup, M. Dering, S. Jha, and P. McDaniel, "Composite constant propagation: Application to android inter-component communication analysis," in *International Conference on Software Engineering (ICSE) - Volume 1*, 2015, pp. 77–88. [Online]. Available: http://dl.acm.org/citation.cfm?id=2818754.2818767

[43] O. Shkaravska, R. Kersten, and M. van Eekelen, "Test-based inference of polynomial loop-bound functions," in *International Conference on the Principles and Practice of Programming in Java (PPPJ)*, 2010, pp. 99–108. [Online]. Available: http://doi.acm.org/10.1145/1852761.1852776

[44] M. Zielenkiewicz, J. Chrzaszcz, and A. Schubert, "Java loops are mainly polynomial," in *International Conference on Current Trends in Theory and Practice of Informatics*, 2015, pp. 603–614. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-46078-8_50

[45] J. Yang, T. Hance, T. H. Austin, A. Solar-Lezama, C. Flanagan, and S. Chong, "Precise, dynamic information flow for database-backed applications," in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2016, pp. 631–647. [Online]. Available: http://doi.acm.org/10.1145/2908080.2908098

[46] A. Yip, X. Wang, N. Zeldovich, and M. F. Kaashoek, "Improving application security with data flow assertions," in *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, 2009, pp. 291–304. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629604

[47] A. Askarov and A. Sabelfeld, "Gradual release: Unifying declassification, encryption and key release policies," in *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*. IEEE Computer Society, 2007, pp. 207–221. [Online]. Available: https://doi.org/10.1109/SP.2007.22