

Interoperability Through Realizability:

Expressing High-level Abstractions using Low-level code

Daniel Baker Patterson

Doctor of Philosophy
Khoury College of Computer Sciences
Northeastern University

2022

**Northeastern University
Khouri College of
Computer Sciences**

PhD Thesis Approval

Thesis Title: Interoperability Through Realizability

Author: Daniel Baker Patterson

PhD Program: Computer Science Cybersecurity Personal Health Informatics

PhD Thesis Approval to complete all degree requirements for the above PhD program.

Anil S. Jadhav
Thesis Advisor

8/8/2022
Date

Mr. Fink
Thesis Reader

8 Aug 2022
Date

Ajrin John
Thesis Reader

8/8/2022
Date

Lhey Morrisett
Thesis Reader

8/8/2022
Date

Thesis Reader

Date

KHOURY COLLEGE APPROVAL:

Anil S. Jadhav
Associate Dean for Graduate Programs

8/8/2022
Date

COPY RECEIVED BY GRADUATE STUDENT SERVICES:

Laura Cadi
Recipient's Signature

9 August 2022
Date

Distribution: Once completed, this form should be scanned and attached to the front of the electronic dissertation document (page 1). An electronic version of the document can then be uploaded to the Northeastern University-UMI Website.

Daniel Baker Patterson: *Interoperability Through Realizability*, Doctor of Philosophy, © 2022

For Amy and Tully.

A B S T R A C T

Large software systems are, inevitably, multi-lingual. This arises for complex socio-historical reasons, as large systems persist for years or decades, while the people working on them and the languages, libraries, and tools available to them change. Looking to these systems, I identify the *interoperability challenge*: that it is more difficult for programmers to reason about multi-lingual systems than about single-language programs. A corollary is that many of the key theorems about languages are proven in the absence of interoperability, reality notwithstanding.

In this dissertation, I identify realizability models as a key tool for addressing the interoperability challenge. Realizability models, which use target-level behavior to inhabit source types, allow the behavior of disparate source languages to be brought together. In doing so, we can recover the type of formal language-based reasoning critical to proving universal properties upon which programmers rely. In this dissertation, the property on which we focus is type soundness, which we explore through a variety of case studies and via two different interoperability mechanisms. The first mechanism, which models how typical foreign-function interfaces work, allows foreign values to be imported at existing types. Realizability models are used to demonstrate the soundness of the conversions that happen at the boundaries. The second mechanism, which better models how programmers wish interoperation worked, allows foreign code to be imported at novel types, thus allowing new behavior to be brought in. Even as the source-level mechanism is quite different between these two approaches, the underlying realizability models are similar, underscoring the central thesis: that such realizability models are an effective way of reasoning about cross-language interoperation.

ACKNOWLEDGMENTS

First, to Amal Ahmed, who has been the best advisor and collaborator for the work in this dissertation that one could hope for. Ever since meeting her at the Oregon Programming Languages Summer School, I could not have imagined a different advisor, and that belief has never been misplaced. I also want to thank the members of my committee: Matthias Felleisen, Arjun Guha, and Greg Morrisett, for careful reading and thoughtful comments.

I thank the broader PRL lab, and the SILC group in particular, for making Northeastern a good place to do this research. And, in particular, to the people who worked on these ideas with me: Andrew Wagner, Noble Mushtak, and Lucy Menon. I also want to thank Gabriel Scherer, William Bowman, and Max New, who helped to make the group a welcoming place to me when it was quite a bit smaller than it is now.

Finally, I want to thank my wife Amy, for everything.

S U P P O R T

This work was funded by the National Science Foundation under grants CCF-1816837, CCF-1618732, CCF-1453796, and CCF-1422133. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the funding agencies.

CONTENTS

I PRELIMINARIES	
1 THE CHALLENGE OF LANGUAGE INTEROPERABILITY	10
2 BACKGROUND: REALIZABILITY & SEMANTIC SOUNDNESS	18
II MOVING VALUES ACROSS LANGUAGE BOUNDARIES	
3 A RECIPE FOR SOUND LANGUAGE INTEROPERABILITY	30
4 CASE STUDY: MUTABLE REFERENCES	40
5 CASE STUDY: AFFINE FUNCTIONS	53
6 CASE STUDY: MEMORY MANAGEMENT AND POLYMORPHISM	76
7 DISCUSSION: VALUE INTEROPERABILITY	94
III MOVING BEHAVIOR ACROSS LANGUAGE BOUNDARIES	
8 A RECIPE FOR SOUNDLY INCORPORATING FOREIGN BEHAVIOR	98
9 CASE STUDY: MUTABLE STATE	107
10 CASE STUDY: EXCEPTIONS	133
IV CONCLUSIONS	
11 RELATED WORK	148
12 FUTURE CONSIDERATIONS	154
V APPENDICES	
A VALUE INTEROPERABILITY: MUTABLE REFERENCES	172
B VALUE INTEROPERABILITY: AFFINE FUNCTIONS	195
C VALUE INTEROPERABILITY: MEMORY MANAGEMENT AND POLYMORPHISM	292
D BEHAVIOR INTEROPERABILITY: MUTABLE STATE	372
E BEHAVIOR INTEROPERABILITY: EXCEPTIONS	389

Part I
PRELIMINARIES

1

THE CHALLENGE OF LANGUAGE INTEROPERABILITY

1.1 LANDSCAPE OF INTEROPERABILITY

Ever since the first compilers, programmers have been inventing languages, and the number in widespread use keeps increasing. At the same time, legacy code is remarkably persistent, so it's not clear that any language, once it reaches a certain threshold, will ever stop being used: indeed there are still huge deployments of COBOL ([Teplitzky, 2019](#); [Powner, 2016](#)) and Fortran still underlies many numeric computations ([of Tennessee et al., 2021](#)), both languages created in the 1950s. Even without more robust study, this should not be terribly surprising: constructing software is remarkably difficult, especially software that inherits complex requirements from the world around it. Once such software is working, or mostly working, the idea of starting over again in a new language, and attempting to not only extract all of the behavior from the first system, but correctly implement it in the new one, is daunting, if not wholly inadvisable.

And yet, new languages do offer real benefits: better runtime systems, better type systems, more expressive language features, and often, lessons learned from earlier languages and complex problem domains. High-profile cases prove the rule: e.g., consider Erlang, a language designed for reliability, used not only to build an Ericsson AXD301 telephone switch that reached an alleged 99.9999999% uptime ([Armstrong, 2003](#)), but at WhatsApp to scale to hundreds of millions of users with only a few dozen staff ([Metz, 2015](#)).

While initially at odds, the stubborn persistence of existing languages and the promise of new ones can be reconciled through mechanisms that support language interoperability. Indeed, the notion of a foreign function interface (FFI) to integrate code written in a different language dates back at least to the 1980s where Common Lisp implementations supported doing this in various ways ([Sexton, 1987](#)). Using this sort of mechanism, new components of old systems can be implemented in newer languages, either for some specific linguistic benefit or as a path towards gradual migration. Even in entirely new systems, individual components can be implemented in different languages that suit the task best. While some languages aim for true generality, many newer languages, e.g., the Rust Programming Language, have gained popularity by targeting a more specific domain: for Rust, making low-level programming safer. While certainly it would be

possible to implement everything in a language like Rust, a more likely engineering outcome is a system like Facebook’s, where a high-level language is used to write the majority of business logic driven user-facing code, and backend code is implemented in a variety of lower-level, higher performance languages.

It is worth exploring how, exactly, such multi-language systems can be treated, and in doing so, focus on the type of interoperability that is the subject of this dissertation. It’s important to understand the current state of affairs in order to understand where research (rather than engineering) can yield benefits.

Broadly speaking, there are two ways of connecting languages together. One is via serialization-based interfaces, and the other is via direct-memory-access. The former is the way that, e.g., Apache Thrift ([Foundation, 2022](#)), originally developed and still used at Facebook, works. Similar to any number of other RPC (remote procedure call) libraries, Thrift uses a generic data and function interface definition language (IDL) to generate serialization and deserialization code in a variety of languages. Then, to interoperate, a programmer invokes the generated code (and networking code, as Thrift is designed for networked services). On the receiving end, generated library code deserializes and invokes the corresponding native functions, doing the reverse for return values. There are benefits to this approach. Not only is it relatively easy to add a new language that can interact with all existing languages, but the fact that separate services can be moved to different physical machines can also be useful on its own.

But, this approach also comes with serious costs. Primarily, there is significant overhead in making calls: even in the case of purely local invocations, we still must context switch between processes, use some sort of socket, and serialize and deserialize. This means that it only makes sense to use this at coarse-grain boundaries. Thrift uses the language of “services”, and that makes sense as a domain, but there are many other uses for interoperability: in particular, consider using a high-level dynamic language, and implementing a high-performance data structure in a lower-level static language. Doing this in an RPC framework would make no sense, as any benefit in speed of operations would be more than offset by the performance lost by the framework overhead.

While the RPC frameworks are in some sense the most advanced form of serialization-based interoperability, we also see this form of interoperability in more primitive ways with systems that use a database, files, or network access to interact with different parts that are implemented in different languages. While better engineering can always improve the RPC frameworks, whether that means more tightly integrating them, improving performance, or increasing expressivity, we have not identified theoretical deficiencies in the serialization-based approach.

Engineering can reduce this, e.g., consider Cap’n Proto (<https://capnproto.org/>, retrieved 2022-7-1), which defines a fixed binary format that must be used as the in-memory data structure format by all languages, thus eliminating serialization cost.

Of course, some security mechanisms do try to ensure such a closed environment, but generally at a very different granularity.

The alternative is to directly make calls, involving no serialization and potentially no overhead. This is how foreign-function interfaces (FFI) generally work, and is the technique that underlies core libraries, data structures, runtimes, and more. To understand the scope of this technique, consider that while a program has some notion of a closed world with its own values and functions, when it actually runs, the execution environment is not closed: there is no OCaml machine that runs OCaml expressions, but rather, the OCaml compiler translates OCaml to either bytecode to be run by an interpreter or machine code. In either case, a machine code program runs, and since it is a machine code program, there is no reason why it has to run only code that originated in the OCaml language. Indeed, it likely will run code not written in OCaml when using certain libraries or parts of the runtime. More specifically, our compiled program can jump directly into machine code that was compiled from other languages, or implemented directly in assembly, or to an interpreter for another language that runs its own bytecode. This can happen with very little overhead, no more than is necessary for function calls within a language, where the stack and registers need to be properly managed to satisfy calling conventions.

For this reason, many high-level dynamic languages implement core data structures in low-level languages and jump between them transparently. Runtimes are also rarely written in the language that they support, and thus involve a degree of switching between languages. There is also incredible flexibility to this approach: because it involves a single program in a single memory space, pointers to both data and functions can freely flow across boundaries, to be accessed or invoked by the foreign language as needed.

1.2 THE INTEROPERABILITY CHALLENGE

With the power to directly access memory and transfer control comes a significant cost: if foreign values and code freely flow in, theorems proved assuming an isolated core language are likely to be meaningless. This leads us to identify the **interoperability challenge**: *reasoning about multi-language programs should be no more difficult than reasoning about single-language programs*. Concretely, what we mean is that if a theorem holds about an idealized version of the language that exists in isolation, some analog of that theorem should hold about the language as it exists when interacting with other languages. Or, to put it differently, if I have a program entirely implemented in language L , the challenge says that local reasoning that I do about a part of that program should not change even if I replace part of the rest of the program with an implementation written in a different language. As an example, consider that a term in my program had a particular static type, like a reference: linking with foreign code should not invalidate this fact. Clearly, this is not currently the case: not only do FFIs allow types

to be violated, but they can cause memory-safe languages to segfault, pure languages to no longer respect key equivalences, and a myriad of other problems. We identify this challenge both as a useful prompt for future work on interoperability, which is sorely lacking, and also as a framing for the thesis of this dissertation.

1.3 THESIS

Modeling source types in terms of target-level behavior is an effective way of reasoning about cross-language interoperation.

This dissertation will show that so-called *realizability models*, which model source types using target behavior, are a useful tool for addressing the interoperability challenge. While the challenge is intentionally broad, the dissertation is necessarily narrow: in it we focus on type soundness. Despite the fact that plenty of reasoning that programmers do is often relational or related solely to static semantics, type soundness is a central theorem that is often proved, or at least aspired to, by typed languages. Thus, addressing soundness is both a useful goal on its own, and a good exercise of the thesis. By showing that our realizability models are up to the task of accounting for various forms of type soundness, we give evidence towards their more general utility in addressing the broader challenge.

Type soundness, around which our central results revolve, says that a program that satisfies a syntactic check behaves in a semantically meaningful way, which generally means that it will only have well defined errors. If arbitrary target values or code can be introduced, this theorem will never hold, as our target will generally have many more behaviors and values than our source language, and can thus create corruption or behavior that is not explainable in terms of the source language.

Typically, if a language is proven sound at all, the proof will almost certainly exclude interoperability (generally supported via an FFI). Unfortunately, this means that the soundness theorem is only a crude approximation of reality, as nearly all programs have FFI calls somewhere in their stack. The interoperability challenge demands that we correct this situation, and this dissertation does, in two different ways. First, however, we address one key point of related work.

Even “untyped” languages are usually “type sound” in this sense, they just have a single type: the dynamic type.

1.4 AN EXISTING APPROACH

Despite the obvious importance of the interoperability challenge, it is not terribly surprising that FFIs have been typically excluded from soundness proofs. After all, the prevailing technique for proving type soundness (due to Wright and Felleisen (1994)) involves a syntactic progress and preservation

proof to show that types are preserved by reduction. This involves showing that a program continues to satisfy the source type system as it runs, but a program that invokes foreign code will involve running code that could not be well-typed in that type system. If this approach were to be followed, syntactic type soundness must account for the foreign code, too. This is precisely what Matthews and Findler (2007) set out to support with *multi-language semantics*, which is defined over a joint syntax that embeds the syntax of the two languages, say core language \mathbf{L} and foreign language \mathbf{F} . Interoperation between these languages is mediated by a *boundary*, $\tau_{\mathbf{L}} \mathcal{L} \mathcal{F}^{\tau_{\mathbf{F}}} e_{\mathbf{F}}$, which enables foreign code $e_{\mathbf{F}} : \tau_{\mathbf{F}}$ to be used in an \mathbf{L} context that expects code of type $\tau_{\mathbf{L}}$ (while the boundary term $\tau_{\mathbf{F}} \mathcal{F} \mathcal{L}^{\tau_{\mathbf{L}}} e_{\mathbf{L}}$ enables the converse).

This multi-language framework has inspired a significant amount of work on interoperability: between simple and dependently typed languages (Oseira et al., 2012), between languages with unrestricted and substructural types (Tov and Pucella, 2010; Scherer et al., 2018), between a high-level functional language and assembly (Patterson et al., 2017), and between source and target languages of compilers (Ahmed and Blume, 2011; Perconti and Ahmed, 2014; New et al., 2016).

Unfortunately, while Matthews-Findler-style boundaries give an elegant, abstract model for interoperability, they are not the right tool for building soundness proofs that account for FFIs. There are two flaws, corrected by Parts II and III of this dissertation.

1.5 CONTRIBUTIONS

VALUE INTEROPERABILITY In Part II of the dissertation, we present an approach for proving soundness of idealized FFIs as they exist: where foreign values and code are imported at existing types of the language. This approach addresses the first major deficiency in the multi-language approach, which is that the semantics is disconnected from the actual code that runs. It also serves to demonstrate for the first time the power of realizability models in addressing questions of interoperability. We describe in more detail what we mean by a disconnect from the actual code that runs, and why it is a problem with the multi-language approach to proving type soundness.

- **Novel Semantics** The biggest problem with multi-language semantics is that when one defines a multi-language, one defines an entirely new language, with its own dynamic and static semantics. For simple languages, the operational behavior of the embedded languages can be preserved, and we can prove that reductions in the original language correspond to reductions in the multi-language, but even the introduc-

tion of state complicates this, as pure reductions must now thread a heap. More significantly, as we show in a case study in Chapter 6, if one language is garbage collected and the other has manual memory management, assuming we want to allow non-trivial interoperation, we have to make garbage collection explicit. It becomes important what happens when there are references between the two heaps (if allowed), which are questions that, by definition, do not arise in the semantics of the original source languages. These decisions can all be answered in the multi-language semantics, but making the modeling decisions there brings in a significant risk: that what is done in the multi-language does not correspond to what happens in the actual program, as we do not use this multi-language semantics to run programs.

- **Existing Compilers** While we could ensure that our multi-language is sensible by proving correctness of the compiler from our multi-language to a target that actually runs, this target code does not necessarily correspond to code that would be emitted by already existing compilers for the languages in question. Further, this approach is in some ways backwards: since the existing compilers are already our source of truth, better to use them to start.
- **(Un)desirable Conversions** The final problem with the disconnect between multi-languages and the target is that since the multi-language defines conversions via meta-functions, it’s not always obvious if those can be realized in performant code, especially because the conversions that will actually happen will be over target-level representations of values. The source-level metafunctions that the multi-language uses for conversions may obscure beneficial implementations or hide necessary inefficiencies, producing results that may not be useful. As a simple example, indexing a sequence may be fast if the sequence is represented as an array, but slow if represented as a linked list, and both operations will look identical in a metafunction.

While multi-language boundaries are a good source-language abstraction, as they can account for both inline code and import/export-style linking, we can address all the above issues by building realizability models derived from how the source languages are compiled and the glue code that is inserted at the boundaries. To account for that glue code, we add a static “convertibility” judgment $\tau_1 \sim \tau_2$ that is realized at runtime by target-level code that converts from target representations of τ_1 to τ_2 , and the converse. The soundness of that glue code, which mediates between data representations and calling conventions, is proved using the realizability models, since target representations of τ_i are exactly what our models give us. While multi-language boundaries have similar type-directed conversions,

they are at the source, and thus do not account for this code, which is of critical importance for the soundness of the FFI. Once we have our realizability models, proving type soundness is a matter of following the standard semantic type soundness playbook: show that a statically well-typed source program, once compiled, satisfies the realizability model at the corresponding semantic type.

BEHAVIOR INTEROPERABILITY While the above, which covers Part II of the dissertation, does a good job of bringing the multi-language approach closer to the target, and thus onto a more useful footing, there is a more fundamental challenge to type soundness and interoperability inherent to the multi-language approach. The problem is, even if we can account for the conversions between a core language L and a foreign language F , the soundness of our types means that if we want to use novel behavior from language F , we need to write code in F . That is because we can only convert to language L values that behave as L types. Concretely, consider if L was a pure language, and the goal of using the FFI was to use stateful behavior in F to implement a mutable reference library. This is a realistic situation, as often FFIs are used to bring new behavior into otherwise less expressive languages. The issue is that no type in L admits such behavior, and so our L programmer would have to do all of the programming that involved state in F and only incorporate extensionally pure code into L . It isn't realistic to expect an L programmer to know all the foreign languages F_1, \dots, F_n in which their library code is written, and thus in practice what we actually observe is that foreign code is imported at types that do not accurately capture their behavior, often threatening soundness. For example, the OCaml FFI allows C code to be used at OCaml types, despite the fact that having correct headers is no guarantee that the function behaves as an OCaml function should: not violating memory safety, value representations, etc.

Indeed, the most serious limitation of a multi-language semantics is that L programmers cannot benefit from the extra expressive power of F unless they write embedded F programs themselves. More typically, libraries expose new primitives that allow the programmer to continue to program in (nearly) their language. For example, the programmer might import `alloc`, `read`, `write` functions to add heap access but otherwise continue programming in their language. Mirroring this, our goal is to allow L programmers to leverage this power while continuing to program in L . The framework in Part III, called “linking types”, shows how to address this issue by building an extension within L that can capture such novel behavior, and again use realizability models to prove the entire system sound. Proving soundness, in this case, involves showing that the novel behavior is safely *encapsulated* within the extension. Important to note is that while the surface level behavior is much

more expressive than in Part II, the underlying realizability models are very similar. Thus, Part III of the dissertation is both an interoperability design contribution on its own and strong evidence for the thesis: that realizability models are a generally useful tool when confronting the interoperability challenge.

2

BACKGROUND: REALIZABILITY & SEMANTIC SOUNDNESS

In this chapter, we give background on type soundness and logical relations before presenting a tutorial on realizability models. The model in the tutorial, while simple, shows how we build these sorts of models and use them to prove type soundness.

TYPE SOUNDNESS Milner (1978) was the first to propose a theorem for type soundness. The theorem states that if programs possess a syntactic property, satisfying an algorithmic type-checker, they therefore possess a semantic property of being well typed. Being well typed means that they cannot get into an erroneous state, which has led to the pithy slogan “well-typed programs don’t go wrong”. Since the mid 1990s, people commonly prove type soundness using a method created by Wright and Felleisen (1994), in which syntactically well-typed programs are shown to remain well-typed as they evaluate. Concretely, this technique uses a pair of lemmas called “progress” (a well-typed term can take a step of evaluation) and “preservation” (a well-typed term that takes a step remains well-typed); inductively, these mean a well-typed program will never get into a non-well-typed (i.e., erroneous) state. What are these “erroneous states”? In Milner’s case, it was `wrong`, which was the element used in the semantic equations for compositions of expressions that had no meaning. One of the equations, for example, that covered function application, is the following (written using slightly more familiar, if verbose, notation than in (Milner, 1978)):

$$\mathcal{E}[(e_1 e_2)]\eta = \begin{cases} \perp, & \text{if } v_1 = \perp \\ \perp, & \text{if } v_2 = \perp \\ \text{wrong}, & \text{if } v_2 = \text{wrong}, \text{ if } v_1 \in F \\ v_1 v_2, & \text{otherwise} \\ \text{wrong}, & \text{otherwise} \end{cases}$$

where $v_1 \in \mathcal{E}[e_1]\eta$ and $v_2 \in \mathcal{E}[e_2]\eta$

where F is all functions, \perp denotes divergence, and η is the (unused) environment. We can see that an application goes wrong if the first term is not a function or if the second term goes wrong.

For Wright-Felleisen, going wrong is an absence of an operational step that the program can take when the program is not in a valid *terminal*

state: the program has gotten stuck. Showing a concise example is more troublesome, since it is the *absence* that determines stuckness, but a typical setup might include cases for stepping in either the function position or argument position, and then a single rule like the following:

$$(\lambda x.e) \ v \rightarrow e[x \mapsto v]$$

Which means that, like Milner, the term will get stuck (go wrong) if the term in the function position does not turn out to be (i.e., evaluate to) a function. While they look quite different, these are two different ways of approaching the same idea, though Milner’s slogan certainly won the rhetorical battle.

It is important to note, of course, that “not going wrong” does not mean that programs cannot have errors! The errors simply must be interpreted by the semantics of the language. For example, most languages allow arbitrary integers to be divided, and diving by zero may be an unrecoverable error. In that case, in Milner’s formulation the program `n/0` would be given the meaning `dividebyzero` or something else to indicate the unrecoverable error, rather than `wrong`. In the Wright-Felleisen formulation, `n/0` would step to a sentinel error value which would be a valid terminal configuration for a program.

More subtly, any meaningful properties that you wish to capture in a type soundness theorem should be distinguished by a program going right or wrong: if violation of the invariant that the type is supposed to enforce does not result in wrongness/stuckness, the type soundness theorem may not be providing a meaningful guarantee that the invariant is enforced, as all that it says is that a well-typed program does not have a conflict with the operational semantics. As an example of how this can get us in trouble, if we design a type system for pointers that is intended to prevent memory aliasing but we have an operation “clone” that copies a pointer, a proof that such a system is sound does not guarantee anything about the aliasing property, as soundness only ensures programs do not get stuck. Instead, a separate theorem needs to be stated that says that there are no aliases in memory after each program step. An alternative approach, which we follow in this dissertation, is to prove soundness by defining a *logical relation*. Here, the property of interest (whether soundness, or soundness plus other properties, like lack of aliasing) is built in, and after showing the language satisfies the logical relation, the property of interest follows “for free” as a corollary.

LOGICAL RELATIONS While Wright-Felleisen prove soundness over the structure of reduction, we take a different approach. We use logical relations to prove a *semantic soundness* theorem. Logical relations are a technique

As a logician, he was working with Gödel’s System T, which is essentially a simply typed lambda calculus with natural numbers.

that most attribute as a generalization of a proof by Tait (1967), and thus has been sometimes called *Tait’s Method*¹. Tait was originally concerned with proving that there was no infinite reduction of a well-typed term in (essentially) the simply typed lambda calculus (i.e., they were *terminating*).

Naive induction over the typing derivation or syntax does not work: once the argument e' is substituted into the body of an abstraction $\lambda x.e$, a novel term $e[x \mapsto e']$ results that did not exist before, and thus is not covered by the induction hypothesis. The essence of *Tait’s Method*, and all subsequent work on logical relations, is to build stronger inductive (type-indexed) relations \mathcal{R}_τ that include the property of interest (in this dissertation, type soundness) and then show that the original term $e : \tau$ belongs to the relation at the corresponding type τ . Since the relation was built to include the property of interest, the desired result now follows as a corollary.

In Tait’s case, the relation R_τ was built out of *terminating* terms: in particular, while the relation at base type is just made up of the base values, the relation at function type $\tau_1 \rightarrow \tau_2$ only admits function values that, when values in the relation at τ_1 are substituted in, result in terms in the relation (so, terminating) at τ_2 . This resolves the issue above, as now rather than being a novel term not covered by the induction hypothesis, $e[x \mapsto e']$ is by definition in the relation, and thus terminating. More involved is how to prove that well-typed simply typed lambda calculus terms are in this relation in the first place, but that is the subject of the next section.

Beyond the foundational characteristics inherited from Tait, there have been several important developments. First, (Girard, 1971; Girard et al., 1989) showed how to extend the technique to account for polymorphism. Next, there were various efforts to incorporate recursion (Pitts, 1998, 2000), mutable references with various restrictions (Pitts and Stark, 1993, 1998; Stark, 1994; Benton and Leperchey, 2005), then recursive types (Birkedal and Harper, 1997; Crary and Harper, 2007), but these models were either limited or cumbersome.

Later, Appel and McAllester (2001) developed the technique of step-indexing to avoid circularity in the presence of recursive types. This was extended to languages with dynamically allocated mutable references by Ahmed (2004), work we rely upon in this dissertation.

While all of the above involve giving interpretation of types of a language using terms (even if untyped) from the same language, we build upon an idea championed by Benton and collaborators called “low-level semantics for high-level types” (or “realistic realizability”) (Benton, 2006). They created *realizability* models where the types came from high-level languages but the

¹ The actual name “logical relations” seems to have come from either G. Plotkin describing Tait’s work, or possibly describing M. Gordon describing Tait’s work. c.f., discussion <https://cstheory.stackexchange.com/questions/7179/what-is-the-origin-of-logical-relations> retrieved 2022/1/20

terms that inhabited the relation were low-level terms, and exercised this to prove type soundness of two standalone languages. Specifically, Benton and Zarfaty (2007) proved an imperative While language sound and Benton and Tabareau (2009) proved type soundness for a simply typed functional language, both times interpreting source types as relations on terms of an idealized Assembly and allowing for compiled code to be linked with a verified memory allocation module implemented in Assembly (Benton, 2006).

REALIZABILITY MODEL TUTORIAL

In this section, we build a realizability model, in the style of Benton's *realistic realizability*, for a simply typed function language, `SimpleFunLang`, and use it to prove type soundness. Our target for the tutorial is high-level: the untyped lambda calculus, which we call *Lambda*. We do not give a source operational semantics for `SimpleFunLang`: its operational semantics is *defined* by compilation. Our compiler is straightforward, but despite this, our target is much more expressive than our source: in particular, all programs in our source terminate (it is a simply typed lambda calculus), whereas the untyped lambda calculus can encode general recursion (and, more straightforwardly, the term Ω , $(\lambda x.xx)(\lambda x.xx)$, runs forever). This section should serve as (1) a tutorial on logical relations / refresher to those familiar with them, (2) a preview of our common syntactic conventions, and (3) a demonstration of the realizability technique, all in a small, self-contained manner.

`SimpleFunLang` has a single base type (`B`) with two inhabitants (`b1` and `b2`) and an operation (`bop`) that only succeeds on one of them (as an identity), variables (`x`), functions (`fun(x : τ){e}`), and application (`e(e)`).

`SimpleFunLang`

$$\begin{array}{ll} \text{Type } \tau & ::= \quad B \mid \tau \rightarrow \tau \\ \text{Expression } e & ::= \quad b_1 \mid b_2 \mid b_{op} \mid x \mid \text{fun}(x : \tau)\{e\} \mid e(e) \\ \text{Value } v & ::= \quad b_1 \mid b_2 \mid \text{fun}(x : \tau)\{e\} \end{array}$$

$$\begin{array}{c} \frac{}{\Gamma \vdash b_1 : B} \quad \frac{}{\Gamma \vdash b_2 : B} \quad \frac{\Gamma \vdash e : B}{\Gamma \vdash b_{op}(e) : B} \quad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \\ \\ \frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash \text{fun}(x : \tau)\{e\} : \tau \rightarrow \tau'} \quad \frac{\Gamma \vdash e : \tau \rightarrow \tau' \quad \Gamma \vdash e' : \tau}{\Gamma \vdash e(e') : \tau'} \end{array}$$

Lambda has boolean base values, if, functions, and application, but no types, and defines reduction using evaluation contexts E to lift the primitive

reduction step (\rightarrow). It also has a expression, *fail*, that causes execution to terminate. Note that this is defined not as a primitive step \rightarrow but an ordinary step \rightarrow , as it needs to discard the context E .

Lambda

$$\begin{array}{lll} \text{Expression } e & ::= & \text{true} \mid \text{false} \mid \text{if } e \ e \ e \mid x \mid \lambda x : \tau. e \mid e \ e \mid \text{fail} \\ \text{Value } v & ::= & \text{true} \mid \text{false} \mid \lambda x : \tau. e \\ \text{Evaluation context } E & ::= & [.] \mid \text{if } C \ e \ e \mid C \ e \mid (\lambda x. e) \ C \end{array}$$

$$\frac{\text{if true } e_1 \ e_2 \rightarrow e_1}{E[\text{true}] \rightarrow e_1} \quad \frac{\text{if false } e_1 \ e_2 \rightarrow e_2}{E[\text{false}] \rightarrow e_2} \quad \frac{(\lambda x. e) \ v \rightarrow [x \mapsto v] e}{E[(\lambda x. e) \ v] \rightarrow [x \mapsto v] e}$$

$$\frac{e \rightarrow e'}{E[e] \rightarrow E[e']}$$

We compile **SimpleFunLang** to *Lambda* with \rightsquigarrow , and given a **SimpleFunLang** term e write the compiled *Lambda* term as e^+ .

$e \rightsquigarrow e^+$
$b_1 \rightsquigarrow \text{true}$
$b_2 \rightsquigarrow \text{false}$
$b_{\text{op}} \rightsquigarrow \lambda x. \text{if } x \ x \ \text{fail}$
$\text{fun}(x : \tau)\{e\} \rightsquigarrow \lambda x. e^+$
$e(e') \rightsquigarrow e^+ \ e'^+$

The soundness theorem we wish to prove is the following:

Theorem 2.0.1 (Soundness, intended). *If $\cdot \vdash e : \tau$, and $e^+ \xrightarrow{*} e'$ then one of:*

- e' is a value.
- e' is fail.
- $e' \rightarrow e''$ for some e'' .

Indeed, it is actually quite rare for languages, aside from those with verified compilers, to have independent formal specifications of their source operational semantics.

This is a little different from a soundness theorem that is defined *solely* in terms of a source operational semantics as it says that the *compiled* term does not get stuck, but matches the common scenario of languages whose operational semantics is defined by translation to another language.

We prove this by first building a realizability model for our language that respects our compiler $(\cdot)^+$. The model has two parts: a value relation, with defines for each type τ a set $\mathcal{V}[\tau]$ of *Lambda* values that behave as that type. Second, we define an expression relation $\mathcal{E}[\tau]$ that defines the sets of *Lambda* terms that behave as type τ .

$$\begin{aligned}
 \mathcal{V}[\mathbf{B}] &= \{\text{true}, \text{false}\} \\
 \mathcal{V}[\tau_1 \rightarrow \tau_2] &= \{\lambda x.e \mid \forall v \in \mathcal{V}[\tau_1]. [x \mapsto v]e \in \mathcal{E}[\tau_2]\} \\
 \mathcal{E}[\tau] &= \{e \mid \exists e'. e \xrightarrow{*} e' \wedge (e' = \text{fail} \vee e' \in \mathcal{V}[\tau])\}
 \end{aligned}$$

The first case of the value relation is straightforward: since \mathbf{b}_1 and \mathbf{b}_2 are the two `SimpleFunLang` values that have the type \mathbf{B} , the only *Lambda* terms that *make sense* in the set $\mathcal{V}[\mathbf{B}]$ are the terms they compile to, *true* and *false*. Our only other *Lambda* values are functions, which don't *seem like* they behave as \mathbf{B} so $\mathcal{V}[\mathbf{B}]$ is the set of exactly those two values.

A NOTE ON PRECISION If “seem like” and “make sense” appear terribly imprecise, this is a consequence of the flexibility of logical relations: since the sets can contain anything that we want, we will only find out when we try to use the relation to prove things if we put in the right elements. Thus, our initial design may indeed be based on (mathematical) intuition, and only when we get stuck in a proof may we come back with more concrete reason as to why a certain element either must be in or cannot be in one of the sets.

More interesting is $\mathcal{V}[\tau_1 \rightarrow \tau_2]$. This set contains *Lambda* functions, $\lambda x.e$, but restricted in the following way: we only admit functions into the set if, given a value v in $\mathcal{V}[\tau_1]$ (the argument), substituting it for the body results in an expression in $\mathcal{E}[\tau_2]$. Thus, these are well typed according to the model, rather than a syntactic judgment.

The expression relation, while critical to the whole system, is perhaps unsurprising: $\mathcal{E}[\tau]$ is made up of *Lambda* terms that run to either *fail* or a value in $\mathcal{V}[\tau]$. Note that while *Lambda* is non-terminating, our relation is only made up of terminating programs, which is both what we want, and not ultimately going to cause problems because any `SimpleFunLang` term that typechecked would terminate after compilation to *Lambda*. Thus, we need not concern ourselves with, or include, terms that do not terminate.

To use this relation, we first need to account for the fact that our typing rules are over *open* terms $\Gamma \vdash e : \tau$, whereas our relation is built out of *closed* terms (i.e., no analog to Γ). While our type soundness theorem is stated over closed terms, if we try to prove that closed well-typed terms are in our relation, we will get stuck on the function case, as the body has a free variable (the argument).

Instead, we first define the notion of a closing substitution, which interprets an environment Γ as a set of substitutions that map the variables in Γ to values that are in $\mathcal{V}[\tau]$ at the right type. Note that this relies upon a variable x being interpreted in the model, and therefore compiling, to x ; in

Before the Halting problem pitchforks are bared, please note: we will never need to decide if an arbitrary Lambda program belongs to our relation.

a setting where the target had different binding structure than the source, this definition may be more complex.

$$\begin{aligned}\mathcal{G}[\cdot] &= \{\cdot\} \\ \mathcal{G}[\mathbf{x} : \tau, \Gamma] &= \{(x \mapsto v, \gamma) \mid v \in \mathcal{V}[\tau] \wedge \gamma \in \mathcal{G}[\Gamma]\}\end{aligned}$$

With that in mind, we can define a shorthand notation that says that given a substitution that behaves as prescribed by types in Γ , the *Lambda* term e behaves as prescribed by τ :

$$\Gamma \models e : \tau \triangleq \forall \gamma \in \mathcal{G}[\Gamma]. \gamma(e) \in \mathcal{E}[\tau]$$

Note, importantly, that this is defined with a *Lambda* term e , not a compiled `SimpleFunLang` term e^+ . This allows the statements we make to be slightly more general, as they will work over arbitrary elements in the relation, rather than just those in the image of the compiler.

Now we complete the proof in a two-step manner. (1) We show that any well-typed term $e : \tau$, when compiled to a *Lambda* term e , is in the model at $\mathcal{E}[\tau]$, and (2) show that any term in the model is type-sound. (2) is easy, since we built the notion of type soundness into the definition of $\mathcal{E}[\tau]$. We prove (1) by induction over the structure of the typing judgment, by proving a “compatibility” lemma for each typing rule in `SimpleFunLang`. “Compatibility” lemmas, given below, show that the model (logical relation) is compatible with the static type system.

Lemma 2.0.2 (Compatibility b_1). $\Gamma \models \text{true} : \mathbf{B}$

Proof. Unfolding the definition, we need to show that for any $\gamma \in \mathcal{G}[\Gamma]. \gamma(\text{true}) \in \mathcal{E}[\mathbf{B}]$. Clearly, $\gamma(\text{true}) = \text{true}$, so if we unfold the definition of $\mathcal{E}[\mathbf{B}]$, we can see we need to show:

$$\exists e'. \text{true} \xrightarrow{*} e' \wedge (e' = \text{fail} \vee e' \in \mathcal{V}[\mathbf{B}])$$

Since true is already a value, it won’t step, which means we can take $e' = \text{true}$, and thus need to show $(\text{true} = \text{fail} \vee \text{true} \in \mathcal{V}[\mathbf{B}])$. Clearly, the $\text{true} \neq \text{fail}$, but the other disjunct is (since $\mathcal{V}[\mathbf{B}] = \{\text{true}, \text{false}\}$), so we are done. \square

Lemma 2.0.3 (Compatibility b_2). $\Gamma \models \text{false} : \mathbf{B}$

Proof. $\gamma(\text{false}) = \text{false} \in \mathcal{E}[\mathbf{B}]$, with $e' = \text{false}$. \square

The typing rules with premises are more interesting, as those premises turn into hypotheses. First, b_{op} and variables.

Lemma 2.0.4 (Compatibility b_{op}). *If $\Gamma \models e : \mathbf{B}$ then $\Gamma \models (\lambda x. \text{if } x \ x \ \text{fail}) \ e : \mathbf{B}$*

Proof. $\gamma((\lambda x.\text{if } x \ x \ \text{fail}) \ e) = (\lambda x.\text{if } x \ x \ \text{fail}) \ \gamma(e)$. To show that is in $\mathcal{E}[\![\mathbf{B}]\!]$, we appeal to the hypothesis, which says $\gamma(e) \xrightarrow{*} e'$, where $e' = \text{fail}$ or $e' \in \mathcal{V}[\![\mathbf{B}]\!]$.

If $e' = \text{fail}$, then by inspection of the operational semantics, the entire term will step to *fail* in one more step.

If $e' \in \mathcal{V}[\![\mathbf{B}]\!]$, we can similarly compose the context at each point with an outer context $(\lambda x.\text{if } x \ x \ \text{fail}) [\cdot]$, which means we know that $(\lambda x.\text{if } x \ x \ \text{fail}) \ \gamma(e) \xrightarrow{*} (\lambda x.\text{if } x \ x \ \text{fail}) \ e'$, where $e' \in \{\text{true}, \text{false}\}$. Consider the two cases.

1. If $e' = \text{true}$, then $(\lambda x.\text{if } x \ x \ \text{fail}) \ e' \rightarrow \text{if true true fail} \rightarrow \text{true}$, and we are done since $\text{true} \in \mathcal{V}[\![\mathbf{B}]\!]$.
2. If $e' = \text{false}$, then $(\lambda x.\text{if } x \ x \ \text{fail}) \ e' \rightarrow \text{if false true fail} \rightarrow \text{fail}$, but this also finishes the proof, as that is a valid result for $\mathcal{E}[\![\mathbf{B}]\!]$.

□

Lemma 2.0.5 (Compatibility `x`). *If $\mathbf{x} : \tau \in \Gamma$ then $\Gamma \models x : \tau$*

Proof. Since $\mathbf{x} : \tau \in \Gamma$, we know that $\gamma(x) = v$ for some $v \in \mathcal{V}[\![\tau]\!]$. We thus take $e' = v$ and are done. □

The last two compatibility lemmas, for function definitions and function application, are the most interesting.

Lemma 2.0.6 (Compatibility `fun`). *If $\Gamma, \mathbf{x} : \tau \models e : \tau'$ then $\Gamma \models \lambda x.e : \tau \rightarrow \tau'$*

Proof. Our obligation is to show that $\gamma(\lambda x.e) \in \mathcal{E}[\![\tau \rightarrow \tau']\!]$. Since x is a bound variable, we can consider it disjoint from the domain of γ , and thus push the substitution into the body of the function. Thus, $\lambda x.\gamma(e)$ does not step, and since it is not *fail*, we need to show that it is in $\mathcal{V}[\![\tau \rightarrow \tau']\!]$. That means we must consider arbitrary v from $\mathcal{V}[\![\tau]\!]$ and show that $[x \mapsto v]\gamma(e) \in \mathcal{E}[\![\tau']\!]$.

To show this, we will appeal to our premise. In particular, we instantiate it with $\gamma' = \gamma, x \mapsto v$, which is thus in $\mathcal{G}[\![\Gamma, \mathbf{x} : \tau]\!]$. This means that $\gamma'(e) \in \mathcal{E}[\![\tau']\!]$, which is exactly equivalent to what we needed to show. □

Lemma 2.0.7 (Compatibility `e(e')`). *If $\Gamma \models e_1 : \tau \rightarrow \tau'$ and $\Gamma \models e_2 : \tau$ then $\Gamma \models e_1 \ e_2 : \tau'$*

Proof. We have to show that $\gamma(e_1 \ e_2) = \gamma(e_1) \ \gamma(e_2) \in \mathcal{E}[\![\tau']\!]$. We consider our first premise, instantiated with γ . It tells us that $\gamma(e_1) \xrightarrow{*} e'_1$, where $e'_1 = \text{fail}$ or $e'_1 \in \mathcal{V}[\![\tau \rightarrow \tau']\!]$. Consider the two cases:

1. In the first case, we can lift the reductions into the context $E \ \gamma(e_2)$ to show that the overall term will similarly step to *fail*, and thus we are done.

2. In the latter case, we next turn to our second hypothesis, again instantiating it with γ . This means that $\gamma(e_2) \xrightarrow{*} e'_2$, where $e'_2 = \text{fail}$ or $e'_2 \in \mathcal{V}[\tau]$. Again, we proceed by case analysis.

- a) In the former case, we can stitch together both reductions, lifting $\gamma(e_1) \xrightarrow{*} e'_1$ into the context $E \gamma(e_2)$ to get that $\gamma(e_1) \gamma(e_2) \xrightarrow{*} e'_1 \gamma(e_2)$ and then lifting the second with the context $e'_1 E$, noting that e'_1 is a value to get that the overall term runs to *fail*.
- b) If v'_2 is in $\mathcal{V}[\tau]$, we can perform the same context lifting to thus get that $\gamma(e_1) \gamma(e_2) \xrightarrow{*} e'_1 v'_2$.

Now, from the definition of $\mathcal{V}[\tau \rightarrow \tau']$, we know that $e'_1 = \lambda x.e$ for some e , and thus $e'_1 v'_2 \rightarrow [x \mapsto v'_2]e$. Further, from the definition of $\mathcal{V}[\tau \rightarrow \tau']$, we know that this term is in $\mathcal{E}[\tau']$, since $v'_2 \in \mathcal{V}[\tau]$. What that means is that there exists e' such that $[x \mapsto v'_2]e \xrightarrow{*} e'$, where $e' = \text{fail}$ or $e' \in \mathcal{V}[\tau']$. Since that e' is the result of the reduction of our original term, this completes the proof.

□

Now that we have completed the compatibility lemmas, we can prove the main theorem of the logical relation, usually called the Fundamental Property. This connects the static semantics to our semantic model.

Theorem 2.0.8 (Fundamental Property). *If $\Gamma \vdash e : \tau$ then $\Gamma \vDash e^+ : \tau$.*

Proof. We prove this by induction over the structure of the typing derivation. Thus, we have one case per typing rule. Each follow from the corresponding compatibility lemma, but we spell out some of the details for completeness.

$\Gamma \vdash b_1 : B$ — In this case, we have no inductive hypothesis, and thus have to show that $\Gamma \vDash b_1^+ : B$, but this is exactly what we proved with our compatibility lemma.

$\Gamma \vdash b_2 : B$ — Same as B_2 .

$\Gamma \vdash b_{\text{op}}(e) : B$ — Our inductive hypothesis tells us that $\Gamma \vDash e^+ : B$, and we need to show that $\Gamma \vDash b_{\text{op}}(e)^+ : B$. But expanding the compiled term, we can see that this is exactly the compatibility lemma statement, instantiated with e^+ .

$\Gamma \vdash x : \tau$ — As with the first two cases, this has no inductive hypotheses, but follows exactly from the compatibility lemma.

$\Gamma \vdash \text{fun}(x : \tau_1)\{e\} : \tau_1 \rightarrow \tau_2$ — As with b_{op} , our inductive hypothesis gives us exactly what we need to instantiate the `fun` compatibility lemma, which then yields the result.

$\Gamma \vdash e(e') : \tau'$ — Here, we need to show that $\Gamma \models e(e')^+ : \tau'$, given $\Gamma \models e^+ : \tau \rightarrow \tau'$ and $\Gamma \models e'^+ : \tau$. We can expand the obligation to $\Gamma \models e^+ e'^+ : \tau'$, after which everything follows from the corresponding compatibility lemma, instantiated with e^+ and e'^+ .

□

As a corollary of the Fundamental Property, we can now prove our goal, type soundness:

Corollary 2.0.9 (Type Soundness). *If $\cdot \vdash e : \tau$ and $e^+ \xrightarrow{*} e'$ then one of:*

- e' is a value.
- e' is fail.
- $e' \rightarrow e''$ for some e'' .

Proof. We first apply the Fundamental Property (Theorem 2.0.8), which tells us that $\cdot \models e^+ : \tau$. This means that there exists an e_f such that $e_f = \text{fail}$ or $e_f \in \mathcal{V}[\tau]$. Consider e' . If $e' = e_f$, then we are done, since every element in $\mathcal{V}[\tau]$ is a value. If e' is not a value, then since e_f cannot take a step and *Lambda* is deterministic (unproved, but straightforward), e' must exist within the reduction $e^+ \xrightarrow{*} e_f$. This means there exists some e'' such that $e' \rightarrow e''$, and so we are done. □

Now, we note an interesting thing. While type soundness was our original goal, the Fundamental Property was our central result, and indeed, it is a *stronger* result than Corollary 2.0.9. In particular, we can prove as a second corollary that all well-typed terms, after compilation, terminate:

Corollary 2.0.10 (Termination). *If $\cdot \vdash e : \tau$ then $\exists e'. e^+ \xrightarrow{*} e' \not\rightarrow$.*

Proof. Since neither *fail* nor elements in $\mathcal{V}[\tau]$ can step, this is a straightforward consequence of the Fundamental Property (Theorem 2.0.8), which tells us that $e^+ \in \mathcal{E}[\tau]$ and thus that there exists such an e' . □

This is not a trivial result, since our *Lambda* language clearly includes divergent terms, but our realizability model only includes a subset that is terminating. Further, while we could have avoided this and structured our $\mathcal{E}[\tau]$ to allow divergence by stating $\forall e'. e \xrightarrow{*} e' \not\rightarrow \implies (e' = \text{fail} \vee e' \in \mathcal{V}[\tau])$, our models may often be stronger than what is necessary to prove “lack of stuckness”.

Termination is not the only way our result is stronger: we have also proved what is sometimes called “strong soundness”, which means that not only does our term either run to *fail* or a value, but that value has the correct type! This is sometimes somewhat subtle to state, as runtime values

may not be able to be type checked, existing only at runtime (so we may not be able to write e' is a value and $\cdot \vdash e' : \tau$, as there may be no static judgment for the value e'), but what we have proved is essentially a semantic analog to this.

Indeed, the notion that the proof method we use is giving us more than simple “lack of stuckness” is not exclusively a property of semantic soundness: the same is true of proofs via Wright-Felleisen progress and preservation, as not only do they clearly need to enrich the language with a notion of typechecking runtime values (as they ensure that at every step, the term is well-typed), but this enriched type system may include more detail than was possible in the statics. While these two approaches to proving type soundness (the “syntactic” and “semantic”) may be typically presented as completely at odds, the reality is that in many cases, what is done in one can be adapted to the other. While some results, like termination, may not readily be adapted to a syntactic approach, others can.

However, a place where the semantic approach really shines, and the reason why we rely so heavily on it, is the notion of realizability. As we will see in later sections, we use our models to ascribe different semantic types to the same target code, which then allows us to reason about how the code that behaves like a type τ can be converted or wrapped to behave like a different type τ' . Also, we can show that target code that did not originate in our source language nonetheless satisfies our semantic types, which is useful for reasoning about the behavior of low-level library code, for instance. This comes along with other flexibility in realizability models: to at once be able to ignore target features (like divergence of *Lambda*) that we do not need or cannot account for but also to ascribe rich statics onto untyped or weakly typed target languages.

Part II

MOVING VALUES ACROSS LANGUAGE
BOUNDARIES

3

A RECIPE FOR SOUND LANGUAGE INTEROPERABILITY

In this chapter, we present a recipe for proving soundness of languages that exchange values over an FFI, and in particular, give foreign values accurate types using the existing local type system. This chapter demonstrates the framework by extending the simple language used in the tutorial in Chapter 2. By seeing the full framework in the small, the reader should be prepared for the more substantive case studies in Chapters 4, 5, and 6.

Language interoperability is about including code written in another language in a given program. This may be to extend a legacy system, to use a particular featureful or well-tested library, or simply because different portions of a program benefit from different linguistic abstractions. Even in a system that includes many languages, each individual boundary is between a pair of languages, and the interaction generally involves first converting core values into representations suitable to the foreign language, executing the foreign code on those values, and then converting the result. To facilitate this, there may be additional code that manages details relating to differences in calling conventions: placing values in particular places on a call stack, manipulating registers, etc. This orchestrated control transfer can, of course, be more interleaved, as what is passed back when control returns may be a higher-order value (a function, function pointer, closure, etc) that can later be invoked itself. But, in all the cases that we consider in this section, and indeed, in what we observe in the design of existing foreign-function interfaces, what crosses the language boundaries are *values*. Accounting for the soundness of this transaction is the subject of this part of the dissertation.

In Chapters 4, 5, and 6, we will explore in detail particular pairs of languages and show how to prove type soundness in the presence of interoperability between them. In this chapter, we present the general recipe for our approach. The source languages we will use are `SimpleFunLangP` and `SimpleFunLangQ`. We will write `SimpleFunLangP` types and terms in `turquoise` and `SimpleFunLangQ` types and terms in `orange`, though will often include subscripts `P` and `Q` to aid legibility in the absence of colors. `SimpleFunLangP` and `SimpleFunLangQ` each extend the example source language from Chapter 2 with a different base type, `P` and `Q` respectively. These base types each have a single inhabitant (`p` and `q` respectively), and are thus semantically equivalent to a unit type. While thus not particularly

interesting on their own, they will allow us to demonstrate the various parts of the framework in full clarity. In the subsequent chapters of this part of the dissertation we will show more realistic challenging cases of interoperability, but with that complexity comes a wealth of details. These details, while necessary to account for type soundness, partly obscure the framework.

THE FRAMEWORK

The inputs to the framework are two source languages, SimpleFunLangP and SimpleFunLangQ , a target language \textit{Lambda} , and compilers $e^+ = e$ and $\textcolor{orange}{e}^+ = e$. This section serves both as a roadmap of what is to come and a reference to refer back to. The first two steps, defining boundary syntax and what types ought to be convertible (§3.0.1 and §3.0.2), must be performed by the **designer** of the interoperability system, whereas the last three, building realizability models and using them to prove soundness of conversions and the interoperating source languages (§3.0.3, §3.0.4, and §3.0.5), should be performed by the **verifier** of the system.

A NOTE ON VERIFICATION While clearly, we are advocates for proving theorems, we would be remiss to ignore that most languages are never proven type sound, and thus advocating extended versions of type soundness is, perhaps, a fools errand. We note that despite Milner’s theorem rarely being proved outside of academia, the notion of type soundness has been an incredibly useful one in rendering languages more reliable. Indeed, most languages aspire to type soundness, even if a formal proof is never attempted. In the same way, we hope that our work can serve both as a theoretical tool that can be used for true proofs, in the rare cases they are achievable, but also as a useful guide for language designers and implementers to make their systems more reliable. Towards this end, we have made an effort to make a delineation between what portions of our work constitute elements of actual languages, compilers, or runtime systems (that are carried out by the **designer**), and what portions are the theoretical scaffolding used to prove the former to be correct (carried out by the **verifier**). Much of the work of the **designer** that pertains to this part of the dissertation is likely already happening: indeed, most of the novelty is in the proof of soundness, rather than the interoperability. In Part III, the situation is a bit different, as we are addressing cases where traditional FFI’s cannot express the interoperability that we desire, and thus there is novel work for both the **designer** and the **verifier**.

SimpleFunLangP

Type τ	::= $P \mid B \mid \tau \rightarrow \tau$
Expression e	::= $p \mid b_1 \mid b_2 \mid b_{op} \mid x \mid \text{fun}(x : \tau)\{e\} \mid e(e)$
Value v	::= $p \mid b_1 \mid b_2 \mid \text{fun}(x : \tau)\{e\}$

SimpleFunLangQ

Type τ	::= $Q \mid B \mid \tau \rightarrow \tau$
Expression e	::= $q \mid b_1 \mid b_2 \mid b_{op} \mid x \mid \text{fun}(x : \tau)\{e\} \mid e(e)$
Value v	::= $q \mid b_1 \mid b_2 \mid \text{fun}(x : \tau)\{e\}$

Lambda

Expression e	::= $true \mid false \mid if \ e \ e \ e \mid x \mid \lambda x : \tau.e \mid e \ e \mid fail$
Value v	::= $true \mid false \mid \lambda x : \tau.e$
Evaluation context E	::= $[.] \mid if \ E \ e \ e \mid E \ e \mid (\lambda x.e) \ E$

Our static semantics are essentially identical to that of **SimpleFunLang** in the tutorial presented in Chapter 2; the only additions are the terms p and q , which have types P and Q respectively. Our compilers are also similar; the only notable difference is that we compile p and q to *true*. Our static semantics will ensure that we do not mix this term with a term of type B or B , so this is a perfectly acceptable compilation choice, and one of the only base values that we have in *Lambda*, which is unchanged from Chapter 2.

p	$\rightsquigarrow true$	q	$\rightsquigarrow true$
b_1	$\rightsquigarrow true$	b_1	$\rightsquigarrow true$
b_2	$\rightsquigarrow false$	b_2	$\rightsquigarrow false$
b_{op}	$\rightsquigarrow \lambda x.\text{if } x \ x \ fail$	b_{op}	$\rightsquigarrow \lambda x.\text{if } x \ x \ fail$
$\text{fun}(x : \tau)\{e\}$	$\rightsquigarrow \lambda x.e^+$	$\text{fun}(x : \tau)\{e\}$	$\rightsquigarrow \lambda x.e^+$
$e(e')$	$\rightsquigarrow e^+ \ e'^+$	$e(e')$	$\rightsquigarrow e^+ \ e'^+$

3.0.1 *Boundary syntax*

To include code from another language, the designer requires some way of invoking such code. While there are various ways of doing this in real toolchains, here she adopts a general approach based on the notion of *language boundaries*.

As in (Matthews and Findler, 2007).

If a program written in the language **SimpleFunLangP** is to include code from the language **SimpleFunLangQ**, the **SimpleFunLangP** designer should add a boundary form $(e)_\tau$. This allows a term $e : \tau_Q$ to be used in an

`SimpleFunLangP` context at type τ_P , for some τ_P and τ_Q . This boundary strategy is very general. It clearly allows inline code, as used by libraries that support inline C, because the term e can be an arbitrary snippet of code. But, it also captures the more common scenario where bindings are declared as imports and then used accordingly. This is because our terms are open, and thus a term can have a `SimpleFunLangQ` binding $f : \tau \rightarrow \tau'$ free, declared in the global environment. Then the use of the imported term would be $(\{f\})_{\tau_P \rightarrow \tau'_P}$ for appropriate types τ_P and τ'_P . In order to support this pattern, we generally typecheck terms of both languages under an environment that includes bindings from both languages. If we were only interested in inline (closed) code, we could avoid this, but we think this more realistically matches the import/export scenario. This means our typing judgments, while largely inheriting from those for `SimpleFunLang` in Chapter 2, will change in shape, as shown below in the rules for the two base types we have added.

$$\frac{}{\Gamma; \Gamma \vdash p : P} \quad \frac{}{\Gamma; \Gamma \vdash q : Q}$$

While we show the additions to syntax for the boundary rules, we will show the additions to the static semantics in the next section, because we need an additional step before we can write down the typing rules.

`SimpleFunLangP` Expression $e ::= \dots | (\{e\})_{\tau_P}$

`SimpleFunLangQ` Expression $e ::= \dots | (\{e\})_{\tau_Q}$

Note that while in our examples, we equip both languages with boundaries, the framework does not require this.

3.0.2 Convertibility rules

To know whether a term $(\{e\})_{\tau_P}$ is well-typed, the designer needs to know if a `SimpleFunLangQ` term $e : \tau_Q$ can be converted to an `SimpleFunLangP` type τ_P . In the case that e has a function type, we typically will expect to convert to a corresponding function type, converting the arguments and the return values. This isn't of course, a requirement: functions could be converted to objects, or other structures that capture the same meaning.

But there is no way to know, a priori, what types can be converted, and thus the framework requires that the designer specify this explicitly. In particular, she must provide judgments of the form $\tau_P \sim \tau_Q$ to indicate that these two types are interconvertible, allowing for the possibility of dynamic conversion errors. Since our notion of linking depends upon

both language SimpleFunLangP and SimpleFunLangQ being compiled to a common target $Lambda$, this conversion needs to be witnessed by $Lambda$ code that performs the conversion. $C_{\tau_P \mapsto \tau_Q}$ denotes the code that performs a target-level conversion from τ_P to τ_Q . For example, if $Lambda$ had numbers instead of booleans and we had a source convertibility relation $\text{bool} \sim \text{int}$, where the former compiles to the numbers 0 and 1, then the conversion $C_{\text{bool} \mapsto \text{int}}$ is a no-op (since compiled booleans are already $Lambda$ language numbers), but $C_{\text{int} \mapsto \text{bool}}$ must do something different. It could raise a dynamic conversion error if given an int other than 0 or 1, or it could collapse all other numbers into one of those, or something else. The particular choice depends on the languages in question, and what the designer of the interoperability system thinks makes sense: the framework only requires that the decision made preserves type soundness.

In the case of SimpleFunLangP and SimpleFunLangQ , we convert between P and Q , and we convert between functions that, in turn, contain convertible types.

$$\frac{}{C_{P \mapsto Q}, C_{Q \mapsto P} : P \sim Q}$$

$$\frac{C_{\tau_1 \mapsto \tau_1}, C_{\tau_1 \mapsto \tau_1} : \tau_1 \sim \tau_1 \quad C_{\tau_2 \mapsto \tau_2}, C_{\tau_2 \mapsto \tau_2} : \tau_2 \sim \tau_2}{C_{\tau_1 \rightarrow \tau_1 \mapsto \tau_1 \rightarrow \tau_1}, C_{\tau_1 \rightarrow \tau_2 \mapsto \tau_1 \rightarrow \tau_2} : \tau_1 \rightarrow \tau_2 \sim \tau_1 \rightarrow \tau_2}$$

For discussion of making them not symmetric, see Chapter 7.

These are symmetric rules, so the order that we write the judgment does not have any particular significance. Further, we will sometimes leave out the code block identifiers in the convertibility judgments and simply write $P \sim Q$, but note that the judgments must *always* be witnessed by that target code, as the compiler needs to insert it when compiling boundary terms.

With these rules, we can now write the typing rules for boundary terms:

$$\frac{\tau \sim \tau \quad \Gamma; \Gamma \vdash e : \tau}{\Gamma; \Gamma \vdash (\lambda e)_{\tau} : \tau} \quad \frac{\tau \sim \tau \quad \Gamma; \Gamma \vdash e : \tau}{\Gamma; \Gamma \vdash (\lambda e)_{\tau} : \tau}$$

For our example, since we compile both p and q to the same $Lambda$ value *true*, the conversions between them are no-ops. Our higher order conversions perform the typical higher-order contract operation (a la (Findler and Felleisen, 2002)): converting arguments before invoking the function on the converted value, and then converting the result at the end. The careful reader might note that since no conversion is a non-identity, this function conversion introduces a pointless eta expansion. We include it here nonetheless: both to not be overly clever, but more importantly, because in larger case studies, this same pattern on functions will reappear, without the ability to simplify.

$$\begin{aligned}
 C_{\textcolor{teal}{P} \rightarrow \textcolor{orange}{Q}}(e) &= e \\
 C_{\textcolor{orange}{Q} \rightarrow \textcolor{teal}{P}}(e) &= e \\
 C_{\tau_1 \rightarrow \tau_1 \mapsto \tau_1 \rightarrow \tau_2}(e) &= \lambda x. C_{\tau_2 \rightarrow \tau_2}(e \ C_{\tau_1 \mapsto \tau_1}(x)) \\
 C_{\tau_1 \rightarrow \tau_2 \mapsto \tau_1 \rightarrow \tau_2}(e) &= \lambda x. C_{\tau_2 \rightarrow \tau_2}(e \ C_{\tau_1 \mapsto \tau_1}(x))
 \end{aligned}$$

Using these, we can now extend our compilers to account for boundary terms. Our compilers have always operated over well-typed terms, but here, we rely on that explicitly, as the types direct the conversions that we insert.

$$\Gamma; \Gamma \vdash (\textcolor{teal}{e})_\tau \rightsquigarrow C_{\tau \rightarrow \tau}(e^+) \quad \text{where } \Gamma; \Gamma \vdash e : \tau \text{ and } \tau \sim \tau$$

$$\Gamma; \Gamma \vdash (\textcolor{orange}{e})_\tau \rightsquigarrow C_{\tau \rightarrow \tau}(e^+) \quad \text{where } \Gamma; \Gamma \vdash e : \tau \text{ and } \tau \sim \tau$$

At this point, we have described all of the implementation work that the language or FFI designer will have to do: add some boundary syntax, declare conversions, implement the conversions in target code, and update the compiler to insert that conversion code when compiling boundaries. Clearly, implementing those conversions needs to be done with some care, and with an idea of what the types mean and how they are realized in the target language. While we describe this as a linear process, the process of verification may result in changes to the conversions as the verifier realizes that what the designer decided wasn't quite right.

3.0.3 Realizability models for both languages

In order to prove type soundness, and in particular, account for the boundaries and convertibility rules from §3.0.1 and §3.0.2, the verifier needs to build a logical relation for both languages. This relation is atypical in two ways. First, it is a *realizability* model, which means that while it is indexed by source types, it is inhabited by target terms. That is, the verifier must first define an interpretation of values for each source type τ , written $\mathcal{V}[\tau]$, as the set of *Lambda* language *values* v that behave as τ . That is, $\mathcal{V}[\textcolor{teal}{B}]$ is not the set of *SimpleFunLangP* language values of type B (i.e., b_1 and b_2), but rather, the *Lambda* values that behave as that *SimpleFunLangP* type (i.e., *true* and *false*). The compiler has to satisfy this relation, sending values of type B to $\mathcal{V}[\textcolor{teal}{B}]$, but the latter may include more values. There is also an expression relation, written $\mathcal{E}[\tau]$, that is the set of *Lambda* language terms that evaluate to values in $\mathcal{V}[\tau]$ (or diverge, or run to a well-defined error).

The second novel aspect is that the relation is indexed with the types of *both* of our source languages; in this example, *SimpleFunLangP* and *SimpleFunLangQ*. Since they compile to the same target, this is sensible: the inhabitants of $\mathcal{V}[\textcolor{teal}{P}]$ and $\mathcal{V}[\textcolor{orange}{Q}]$ (for example) are both *Lambda* values.

By bringing the types of both languages into a common setting, the verifier gains powerful reasoning principles; for example, we can ask if $\mathcal{V}[\![\text{P}]\!] = \mathcal{V}[\![\text{Q}]\!]$.

Since the only difference between the languages in this section and **SimpleFunLang** from Chapter 2 are the additional base types, the logical relations look almost the same. Note that since the expression relation is the same for both languages, we do not include two cases for it, writing τ for either $\textcolor{teal}{\tau}$ or $\textcolor{brown}{\tau}$.

$$\begin{aligned}
 \mathcal{V}[\![\text{P}]\!] &= \{\text{true}\} \\
 \mathcal{V}[\![\text{B}]\!] &= \{\text{true}, \text{false}\} \\
 \mathcal{V}[\![\tau_1 \rightarrow \tau_2]\!] &= \{\lambda x.e \mid \forall v \in \mathcal{V}[\![\tau_1]\!]. [x \mapsto v]e \in \mathcal{E}[\![\tau_2]\!]\} \\
 \mathcal{V}[\![\text{Q}]\!] &= \{\text{true}\} \\
 \mathcal{V}[\![\text{B}]\!] &= \{\text{true}, \text{false}\} \\
 \mathcal{V}[\![\tau_1 \rightarrow \tau_2]\!] &= \{\lambda x.e \mid \forall v \in \mathcal{V}[\![\tau_1]\!]. [x \mapsto v]e \in \mathcal{E}[\![\tau_2]\!]\} \\
 \mathcal{E}[\![\tau]\!] &= \{e \mid \exists e'. e \xrightarrow{*} e' \wedge (e' = \text{fail} \vee e' \in \mathcal{V}[\![\tau]\!])\}
 \end{aligned}$$

Looking at this, we can clearly see that, as expected, $\mathcal{V}[\![\text{P}]\!] = \mathcal{V}[\![\text{Q}]\!]$.

3.0.4 Soundness of conversions

Using the realizability models defined in §3.0.3, the verifier can prove that the convertibility rules defined in §3.0.2 are sound. In particular, if $\tau_{\text{P}} \sim \tau_{\text{Q}}$, then she should show that the conversions $C_{\tau_{\text{P}} \mapsto \tau_{\text{Q}}}$ and $C_{\tau_{\text{Q}} \mapsto \tau_{\text{P}}}$ actually translate expressions between the types correctly:

$$\forall e \in \mathcal{E}[\![\tau_{\text{P}}]\!]. C_{\tau_{\text{P}} \mapsto \tau_{\text{Q}}}(e) \in \mathcal{E}[\![\tau_{\text{Q}}]\!] \wedge \forall e \in \mathcal{E}[\![\tau_{\text{Q}}]\!]. C_{\tau_{\text{Q}} \mapsto \tau_{\text{P}}}(e) \in \mathcal{E}[\![\tau_{\text{P}}]\!]$$

Since the model defines type interpretations, this ensures that the conversions do exactly what is expected.

We have two convertibility rules, and thus two proofs to do, though each involves both directions. For $\text{P} \sim \text{Q}$, we need to show that for any $e \in \mathcal{E}[\![\text{P}]\!]$, $C_{\text{P} \mapsto \text{Q}}(e) \in \mathcal{E}[\![\text{Q}]\!]$, and the converse. Since $C_{\text{P} \mapsto \text{Q}}(e) = e$, this amounts to showing that $e \in \mathcal{E}[\![\text{P}]\!] \iff e \in \mathcal{E}[\![\text{Q}]\!]$. This follows from the fact that $\mathcal{V}[\![\text{P}]\!] = \mathcal{V}[\![\text{Q}]\!]$.

The function case is a little more interesting; we consider one direction, from **SimpleFunLangP** to **SimpleFunLangQ**; the other is identical. What we need to show is that if $e \in \mathcal{E}[\![\tau_1 \rightarrow \tau_2]\!]$ then $C_{\tau_1 \rightarrow \tau_1 \mapsto \tau_1 \rightarrow \tau_1}(e) \in \mathcal{E}[\![\tau_1 \rightarrow \tau_2]\!]$, where $\tau_1 \sim \textcolor{teal}{\tau}_1$ and $\tau_2 \sim \textcolor{brown}{\tau}_2$. Expanding the conversion, we see the term under consideration is $\lambda x. C_{\tau_2 \mapsto \tau_2}(e C_{\tau_1 \mapsto \tau_1}(x))$. Since this is already a value, it does not step, which means that to satisfy $\mathcal{E}[\![\tau_1 \rightarrow \tau_2]\!]$, it suffices to show that it is in $\mathcal{V}[\![\tau_1 \rightarrow \tau_2]\!]$. Syntactically, it clearly is a *Lambda* function, so we need

to show that given an arbitrary v from $\mathcal{V}[\tau_1]$, $[x \mapsto v]C_{\tau_2 \mapsto \tau_2}(e C_{\tau_1 \mapsto \tau_1}(x)) \in \mathcal{E}[\tau_2]$. Simplifying slightly, using the fact that the only variables introduced by conversions are bound, we have the term $C_{\tau_2 \mapsto \tau_2}(e C_{\tau_1 \mapsto \tau_1}(v))$. From our inductive hypothesis, we know that this will be in $\mathcal{E}[\tau_2]$, as needed, if $e C_{\tau_1 \mapsto \tau_1}(v)$ is in $\mathcal{E}[\tau_2]$. Now, we know that $e \in \mathcal{E}[\tau_1 \rightarrow \tau_2]$, which means that either it runs to *fail*, in which case the entire term will have failed and we are done, or it runs to some value e' in $\mathcal{V}[\tau_1 \rightarrow \tau_2]$. Now, we know that if applied to a value in $\mathcal{V}[\tau_1]$, this will result in a term in $\mathcal{E}[\tau_2]$, and thus we will be done. So it suffices to show that $C_{\tau_1 \mapsto \tau_1}(v)$ runs to a value in $\mathcal{V}[\tau_1]$. This also follows from our inductive hypothesis, since we know $v \in \mathcal{V}[\tau_1]$ and the conversion will result in a term in $\mathcal{E}[\tau_1]$. This means that it either runs to *fail* (in which case the entire term does, and we are done) or it runs to a value in $\mathcal{V}[\tau_1]$, exactly as needed.

3.0.5 Soundness of entire languages

Proving the conversions sound (§3.0.4) is the central goal, of course, but the verifier also needs to ensure that the model defined in §3.0.3 is actually faithful to the languages. She does this by following the standard approach for proving semantic type soundness outlined in Chapter 2. First, for each typing rule in both source languages, she proves that a corresponding lemma holds in terms of the model. In order to do that, we first need to define closing substitutions and an open logical relation:

$$\begin{aligned}\mathcal{G}[\cdot] &= \{\cdot\} \\ \mathcal{G}[\mathbf{x} : \tau, \Gamma] &= \{(x \mapsto v, \gamma) \mid v \in \mathcal{V}[\tau] \wedge \gamma \in \mathcal{G}[\Gamma]\} \\ \mathcal{G}[\mathbf{x} : \tau, \Gamma] &= \{(x \mapsto v, \gamma) \mid v \in \mathcal{V}[\tau] \wedge \gamma \in \mathcal{G}[\Gamma]\} \\ \Gamma; \Gamma \models e : \tau &\triangleq \forall \gamma \in \mathcal{G}[\Gamma], \gamma \in \mathcal{G}[\Gamma]. \gamma(\gamma(e)) \in \mathcal{E}[\tau] \\ \Gamma; \Gamma \models e : \tau &\triangleq \forall \gamma \in \mathcal{G}[\Gamma], \gamma \in \mathcal{G}[\Gamma]. \gamma(\gamma(e)) \in \mathcal{E}[\tau]\end{aligned}$$

Then, she proves the following lemmas. Here, we elide all proofs except the boundary cases, as those are the only ones that differ materially from what was presented in Chapter 2.

Lemma 3.0.1 (Compatibility p). $\Gamma; \Gamma \models \text{true} : \mathbf{P}$

Lemma 3.0.2 (Compatibility q). $\Gamma; \Gamma \models \text{true} : \mathbf{Q}$

Lemma 3.0.3 (Compatibility b₁). $\Gamma; \Gamma \models \text{true} : \mathbf{B}$

Lemma 3.0.4 (Compatibility b₁). $\Gamma; \Gamma \models \text{true} : \mathbf{B}$

Lemma 3.0.5 (Compatibility b₂). $\Gamma; \Gamma \models \text{false} : \mathbf{B}$

Lemma 3.0.6 (Compatibility b₂). $\Gamma; \Gamma \models \text{false} : \mathbf{B}$

Lemma 3.0.7 (Compatibility b_{op}). *If $\Gamma; \Gamma \models e : B$ then $\Gamma; \Gamma \models (\lambda x. \text{if } x \ x \ \text{fail}) \ e : B$*

Lemma 3.0.8 (Compatibility b_{op}). *If $\Gamma; \Gamma \models e : B$ then $\Gamma; \Gamma \models (\lambda x. \text{if } x \ x \ \text{fail}) \ e : B$*

Lemma 3.0.9 (Compatibility x). *If $x : \tau \in \Gamma$ then $\Gamma; \Gamma \models x : \tau$*

Lemma 3.0.10 (Compatibility x). *If $x : \tau \in \Gamma$ then $\Gamma; \Gamma \models x : \tau$*

Lemma 3.0.11 (Compatibility fun). *If $\Gamma; \Gamma, x : \tau \models e : \tau'$ then $\Gamma; \Gamma \models \lambda x. e : \tau \rightarrow \tau'$*

Lemma 3.0.12 (Compatibility fun). *If $\Gamma; \Gamma, x : \tau \models e : \tau'$ then $\Gamma; \Gamma \models \lambda x. e : \tau \rightarrow \tau'$*

Lemma 3.0.13 (Compatibility $e(e')$). *If $\Gamma; \Gamma \models e_1 : \tau \rightarrow \tau'$ and $\Gamma; \Gamma \models e_2 : \tau$ then $\Gamma; \Gamma \models e_1 \ e_2 : \tau'$*

Lemma 3.0.14 (Compatibility $e(e')$). *If $\Gamma; \Gamma \models e_1 : \tau \rightarrow \tau'$ and $\Gamma; \Gamma \models e_2 : \tau$ then $\Gamma; \Gamma \models e_1 \ e_2 : \tau'$*

Lemma 3.0.15 (Compatibility $(e)_{\tau}$). *If $\tau \sim \tau$ and $\Gamma; \Gamma \models e : \tau$ then $\Gamma; \Gamma \models C_{\tau \mapsto \tau}(e) : \tau$*

Proof. This follows directly from the proof in 3.0.4. □

Lemma 3.0.16 (Compatibility $(e)_{\tau}$). *If $\tau \sim \tau$ and $\Gamma; \Gamma \models e : \tau$ then $\Gamma; \Gamma \models C_{\tau \mapsto \tau}(e) : \tau$*

Proof. This follows directly from the proof in 3.0.4. □

Then, by induction over a typing derivation, she can show that any term that is statically well-typed belongs to the model: the Fundamental Property. Belonging to the model, in turn, can be used to show type soundness as a corollary, since the expression relation captures exactly what we mean by type soundness.

Theorem 3.0.17 (Fundamental Property). *If $\Gamma; \Gamma \vdash e : \tau$ then $\Gamma; \Gamma \models e^+ : \tau$, and if $\Gamma; \Gamma \vdash e : \tau$ then $\Gamma; \Gamma \models e^+ : \tau$.*

Proof. By induction over the typing derivations, using the compatibility lemmas. □

Corollary 3.0.18 (Type Soundness for `SimpleFunLangP`). *If $\cdot; \cdot \vdash e : \tau$ and $e^+ \xrightarrow{*} e'$ then one of:*

- e' is a value.

- e' is fail.
- $e' \rightarrow e''$ for some e'' .

Proof. By application of the Fundamental Property and definition of the logical relation. \square

Corollary 3.0.19 (Type Soundness for SimpleFunLangQ). *If $\cdot; \cdot \vdash e : \tau$ and $e^+ \xrightarrow{*} e'$ then one of:*

- e' is a value.
- e' is fail.
- $e' \rightarrow e''$ for some e'' .

Proof. By application of the Fundamental Property and definition of the logical relation. \square

4

CASE STUDY: MUTABLE REFERENCES

Results from this section (Chapters 4, 5, 6) appeared in (Patterson et al., 2022), which was joint work with Noble Mushtak, Andrew Wagner, and Amal Ahmed.

Aliased mutable data is challenging to deal with no matter the context, but aliasing across languages is especially difficult because giving a pointer to a foreign language can allow for *unknown* data to be written to its address. Specifically, if the pointer has a particular type in the host language, then only certain data should be written to it, but the foreign language may not respect or even know about that restriction. One existing approach to this problem is to create proxies, where data is guarded or converted before being read or written (Dimoulas et al., 2012; Strickland et al., 2012; Mates et al., 2019). While sound, this comes with significant runtime overhead. Here, our framework suggests a different safe possibility.

LANGUAGES In this case study, we explore this problem using two simply-typed functional source languages with dynamically allocated mutable references, **RefHL** and **RefLL** (for “higher-level” and “lower-level”). **RefHL** has boolean, sum, and product types, whereas **RefLL** has arrays ($[e_1, \dots, e_n] : [\tau]$). Their syntax is given in Figure 4.1 and their static semantics in Figure 4.2.

RefHL and **RefLL** are compiled (Figure 4.4—note that we write e^+ to indicate e' , where $e \rightsquigarrow e'$) into an untyped stack-based language called StackLang (inspired by (Kleffner, 2017)), whose syntax and small-step operational semantics — a relation on configurations $\langle H ; S ; P \rangle$ comprised of a heap, stack, and program — are given in Figure 4.3; here we describe a few highlights. First, we note that StackLang values include not only numbers, thunks, and locations, but arrays of values, a simplification we made for the sake of presentation. We could have put all large values on the heap, but that would have necessitated pointer offsetting, more indirection when encoding values, and generally cluttered up the presentation.

Second, notice the interplay between thunk and lam: thunks are suspended computations, whereas lam is an instruction (not a value) responsible solely for substitution. We can see how these features are combined, or used separately, in our compilers (Figure 4.4). Finally, note that for any instruction where the precondition on the stack is not met, the configuration steps to a program with fail TYPE (a dynamic type error).

*The notion that
lambdas only
substitute follows
Call-by-push-value
(Levy, 2001).*

RefHL Type $\tau ::= \text{unit} \text{bool} \tau + \tau \tau \times \tau \tau \rightarrow \tau \text{ref } \tau$ Expr. $e ::= () \text{true} \text{false} x \text{inl } e \text{inr } e (e, e)$ $ \text{fst } e \text{snd } e \text{if } e \text{ e e e} \lambda x : \tau. e e \text{ e}$ $ \text{match } e \text{ x}\{e\} \text{ y}\{e\} \text{ref } e !e e := e (e)_{\tau}$
RefLL Type $\tau ::= \text{int} [\tau] \tau \rightarrow \tau \text{ref } \tau$ Expr. $e ::= n x [e, \dots] e[e] \lambda x : \tau. e e \text{ e} e + e$ $ \text{if0 } e \text{ e e} \text{ref } e !e e := e (e)_{\tau}$

Figure 4.1: Syntax for **RefHL** and **RefLL**.

CONVERTIBILITY In our source languages, we may syntactically embed a term from one language into the other using the boundary forms $(e)_{\tau_A}$ and $(e)_{\tau_B}$. The typing rules for boundary terms require that the boundary types be convertible, written $\tau_A \sim \tau_B$. Those typing rules are:

$$\frac{\Gamma; \Gamma \vdash e : \tau_A \quad \tau_A \sim \tau_B}{\Gamma; \Gamma \vdash (e)_{\tau_B} : \tau_B} \qquad \frac{\Gamma; \Gamma \vdash e : \tau_B \quad \tau_A \sim \tau_B}{\Gamma; \Gamma \vdash (e)_{\tau_A} : \tau_A}$$

Note that the convertibility judgment is a declarative, extensible judgment that describes closed types in one language that are interconvertible with closed types in the other, allowing for the possibility of well-defined runtime errors. By separating this judgment from the rest of the type system, the language designer can allow additional conversions to be added later, whether by implementers or even end-users. The second thing to note is that this presentation allows for open terms to be converted, so we must maintain a type environment for both languages during typechecking (both Γ and Δ), as we have to carry information from the site of binding—possibly through conversion boundaries—to the site of variable use. A simpler system, which we have explored, would only allow closed terms to be converted. In that case, the typing rules still use the $\tau_A \sim \tau_B$ judgment but do not thread foreign environments (using only Γ for **RefHL** and only Γ for **RefLL**).

Figure 4.5 contains the convertibility rules we have defined for this case study. Each come with target-language instruction sequences that perform the conversions, written $C_{\tau_A \mapsto \tau_B}$ (some are no-ops). An instruction sequence $C_{\tau_A \mapsto \tau_B}$, while ordinary target code, when appended to a program in the model at type τ_A , should result in a program in the model at type τ_B . An implementer can write these conversions based on understanding of the sets of target terms that inhabit each source type, before defining a proper semantic model (or possibly, without defining one, if formal soundness is not required). They would do this based on inspection of the compiler and the target.

From Figure 4.4, we see that **bool** and **int** both compile to target integers, and importantly, that **if** compiles to **if0**, which means the compiler interprets

$$\boxed{\Gamma; \Gamma \vdash e : \tau}$$

$$\frac{}{\Gamma; \Gamma \vdash () : \text{unit}} \quad \frac{}{\Gamma; \Gamma \vdash \text{true} : \text{bool}} \quad \frac{}{\Gamma; \Gamma \vdash \text{false} : \text{bool}} \quad \frac{x : \tau \in \Gamma}{\Gamma; \Gamma \vdash x : \tau}$$

$$\frac{\Gamma; \Gamma \vdash e : \tau_1 \quad \vdash \tau_2}{\Gamma; \Gamma \vdash \text{inl } e : \tau_1 + \tau_2} \quad \frac{\Gamma; \Gamma \vdash e : \tau_2 \quad \vdash \tau_1}{\Gamma; \Gamma \vdash \text{inr } e : \tau_1 + \tau_2}$$

$$\frac{\Gamma; \Gamma \vdash e : \text{bool} \quad \Gamma; \Gamma \vdash e_1 : \tau \quad \Gamma; \Gamma \vdash e_2 : \tau}{\Gamma; \Gamma \vdash \text{if } e \ e_1 \ e_2 : \tau}$$

$$\frac{\Gamma; \Gamma \vdash e : \tau_1 + \tau_2 \quad \Gamma; \Gamma, x : \tau_1 \vdash e_1 : \tau \quad \Gamma; \Gamma, y : \tau_2 \vdash e_2 : \tau}{\Gamma; \Gamma \vdash \text{match } e \ x \{e_1\} \ y \{e_2\} : \tau}$$

$$\frac{\Gamma; \Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma; \Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2} \quad \frac{\Gamma; \Gamma \vdash e : \tau_1 \rightarrow \tau_2 \quad \Gamma; \Gamma \vdash e' : \tau_1}{\Gamma; \Gamma \vdash e \ e' : \tau_2}$$

$$\frac{\Gamma; \Gamma \vdash e_1 : \tau_1 \quad \Gamma; \Gamma \vdash e_2 : \tau_2}{\Gamma; \Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2} \quad \frac{\Gamma; \Gamma \vdash e : \tau_1 \times \tau_1}{\Gamma; \Gamma \vdash \text{fst } e : \tau_1} \quad \frac{\Gamma; \Gamma \vdash e : \tau_1 \times \tau_1}{\Gamma; \Gamma \vdash \text{snd } e : \tau_2}$$

$$\frac{\Gamma; \Gamma \vdash e : \tau}{\Gamma; \Gamma \vdash \text{ref } e : \text{ref } \tau} \quad \frac{\Gamma; \Gamma \vdash e : \text{ref } \tau}{\Gamma; \Gamma \vdash !e : \tau} \quad \frac{\Gamma; \Gamma \vdash e_1 : \text{ref } \tau \quad \Gamma; \Gamma \vdash e_2 : \tau}{\Gamma; \Gamma \vdash e_1 := e_2 : \text{unit}}$$

$$\frac{\Gamma; \Gamma \vdash e : \tau \quad \tau \sim \tau}{\Gamma; \Gamma \vdash (e)_\tau : \tau}$$

$$\boxed{\Gamma; \Gamma \vdash e : \tau}$$

$$\frac{}{\Gamma; \Gamma \vdash n : \text{int}} \quad \frac{\Gamma; \Gamma \vdash e_1 : \text{int} \quad \Gamma; \Gamma \vdash e_2 : \text{int}}{\Gamma; \Gamma \vdash e_1 + e_2 : \text{int}} \quad \frac{x : \tau \in \Gamma}{\Gamma; \Gamma \vdash x : \tau}$$

$$\frac{\Gamma; \Gamma \vdash e_1 : \tau \quad \dots \quad \Gamma; \Gamma \vdash e_n : \tau}{\Gamma; \Gamma \vdash [e_1, \dots, e_n] : [\tau]} \quad \frac{\Gamma; \Gamma \vdash e_1 : [\tau] \quad \Gamma; \Gamma \vdash e_2 : \text{int}}{\Gamma; \Gamma \vdash e_1[e_2] : \tau}$$

$$\frac{\Gamma; \Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma; \Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2} \quad \frac{\Gamma; \Gamma \vdash e : \tau_1 \rightarrow \tau_2 \quad \Gamma; \Gamma \vdash e' : \tau_1}{\Gamma; \Gamma \vdash e \ e' : \tau_2}$$

$$\frac{\Gamma; \Gamma \vdash e : \text{int} \quad \Gamma; \Gamma \vdash e_1 : \tau \quad \Gamma; \Gamma \vdash e_2 : \tau}{\Gamma; \Gamma \vdash \text{if0 } e \ e_1 \ e_2 : \tau}$$

$$\frac{\Gamma; \Gamma \vdash e : \tau}{\Gamma; \Gamma \vdash \text{ref } e : \text{ref } \tau} \quad \frac{\Gamma; \Gamma \vdash e : \text{ref } \tau}{\Gamma; \Gamma \vdash !e : \tau} \quad \frac{\Gamma; \Gamma \vdash e_1 : \text{ref } \tau \quad \Gamma; \Gamma \vdash e_2 : \tau}{\Gamma; \Gamma \vdash e_1 := e_2 : \text{unit}}$$

$$\frac{\Gamma; \Gamma \vdash e : \tau \quad \tau \sim \tau}{\Gamma; \Gamma \vdash (e)_\tau : \tau}$$

Figure 4.2: Static semantics for **RefHL** and **RefLL**.

Program P	$\ ::= \cdot \mid i.P$	Value v	$\ ::= n \mid \text{thunk } P \mid \ell \mid [v, \dots]$
Instruction i	$\ ::= \text{push } v \mid \text{add} \mid \text{less?} \mid \text{if0 } P \mid \text{lam } x.P \mid \text{call}$		
	$\mid \text{idx} \mid \text{len} \mid \text{alloc} \mid \text{read} \mid \text{write} \mid \text{fail } c$		
Error Code c	$\ ::= \text{TYPE} \mid \text{IDX} \mid \text{CONV}$		
Heap H	$\ ::= \{\ell:v,\dots\}$	Stack S	$\ ::= v, \dots, v \mid \text{Fail } c$
	$\langle H; S; \text{push } v, P \rangle \rightarrow \langle H; S, v; P \rangle$		$(S \neq \text{Fail } c)$
	$\langle H; \text{Fail } c; \text{push } v, P \rangle \rightarrow \langle H; \text{Fail } c; \text{fail } \text{TYPE} \rangle$		
	$\langle H; S, n_2, n_1; \text{add}, P \rangle \rightarrow \langle H; S, (n_1 + n_2); P \rangle$		
	$\langle H; S; \text{add}, P \rangle \rightarrow \langle H; S; \text{fail } \text{TYPE} \rangle$		$(S \neq S', n_2, n_1)$
	$\langle H; S, n_2, n_1; \text{less?}, P \rangle \rightarrow \langle H; S, 0; P \rangle$		$(n_1 < n_2)$
	$\langle H; S, n_2, n_1; \text{less?}, P \rangle \rightarrow \langle H; S, 1; P \rangle$		$(n_1 \geq n_2)$
	$\langle H; S; \text{less?}, P \rangle \rightarrow \langle H; S; \text{fail } \text{TYPE} \rangle$		$(S \neq S', n_2, n_1)$
	$\langle H; S, 0; \text{if0 } P_1, P_2, P \rangle \rightarrow \langle H; S; P_1, P \rangle$		
	$\langle H; S, n; \text{if0 } P_1, P_2, P \rangle \rightarrow \langle H; S; P_2, P \rangle$		$(n \neq 0)$
	$\langle H; S; \text{if0 } P_1, P_2, P \rangle \rightarrow \langle H; S; \text{fail } \text{TYPE} \rangle$		$(S \neq S', n)$
	$\langle H; S, v; \text{lam } x. P_1, P_2 \rangle \rightarrow \langle H; S; [x \mapsto v]P_1, P_2 \rangle$		
	$\langle H; S; \text{lam } x. P_1, P_2 \rangle \rightarrow \langle H; S; \text{fail } \text{TYPE} \rangle$		$(S \neq S', v)$
	$\langle H; S, \text{thunk } P_1; \text{call}, P_2 \rangle \rightarrow \langle H; S; P_1, P_2 \rangle$		
	$\langle H; S; \text{call}, P_2 \rangle \rightarrow \langle H; S; \text{fail } \text{TYPE} \rangle$		$(S \neq S', \text{thunk } P_1)$
	$\langle H; S, [v_0, \dots, v_{n_2}], n_1; \text{idx}, P \rangle \rightarrow \langle H; S, v_{n_1}; P \rangle$		$(n_1 \in [0, n_2])$
	$\langle H; S, [v_0, \dots, v_{n_2}], n_1; \text{idx}, P \rangle \rightarrow \langle H; S; \text{fail } \text{IDX} \rangle$		$(n_1 \notin [0, n_2])$
	$\langle H; S; \text{idx}, P \rangle \rightarrow \langle H; S; \text{fail } \text{TYPE} \rangle$		$(S \neq S', [v_0, \dots, v_{n_2}], n_1)$
	$\langle H; S, [v_0, \dots, v_n]; \text{len}, P \rangle \rightarrow \langle H; S, (n+1); P \rangle$		
	$\langle H; S; \text{len}, P \rangle \rightarrow \langle H; S; \text{fail } \text{TYPE} \rangle$		$(S \neq S', [v_0, \dots, v_n])$
	$\langle H; S, v; \text{alloc}, P \rangle \rightarrow \langle H \uplus \{\ell \mapsto v\}; S, \ell; P \rangle$		
	$\langle H; \cdot; \text{alloc}, P \rangle \rightarrow \langle H; \cdot; \text{fail } \text{TYPE} \rangle$		
	$\langle H \uplus \{\ell \mapsto v\}; S, \ell; \text{read}, P \rangle \rightarrow \langle H \uplus \{\ell \mapsto v\}; S, v; P \rangle$		
	$\langle H; S; \text{read}, P \rangle \rightarrow \langle H; S; \text{fail } \text{TYPE} \rangle$		$(S \neq S', \ell)$
	$\langle H \uplus \{\ell \mapsto \cdot\}; S, \ell, v; \text{write}, P \rangle \rightarrow \langle H \uplus \{\ell \mapsto v\}; S; P \rangle$		
	$\langle H; S; \text{write}, P \rangle \rightarrow \langle H; S; \text{fail } \text{TYPE} \rangle$		$(S \neq S', \ell, v)$
	$\langle H; S; \text{fail } c, P \rangle \rightarrow \langle H; \text{Fail } c; \cdot \rangle$		

Figure 4.3: Syntax and operational semantics for StackLang

$$\begin{aligned} \text{SWAP} &\triangleq \text{lam } x.(\text{lam } y.\text{push } x, \text{push } y) \\ \text{DROP} &\triangleq \text{lam } x.() \quad \text{DUP} \triangleq \text{lam } x.(\text{push } x, \text{push } x) \end{aligned}$$

e \rightsquigarrow P

(\rightsquigarrow push 0
true	\rightsquigarrow push 0
false	\rightsquigarrow push 1
x	\rightsquigarrow push x
inl e	\rightsquigarrow e^+ , lam x. (push [0, x])
inr e	\rightsquigarrow e^+ , lam x. (push [1, x])
if e e ₁ e ₂	\rightsquigarrow e^+ , if0 $e_1^+ e_2^+$
match e x{e ₁ } y{e ₂ }	\rightsquigarrow e^+ , DUP, push 1, idx, SWAP, push 0, idx, if0 (lam x. e_1^+) (lam y. e_2^+)
(e ₁ , e ₂)	\rightsquigarrow e_1^+, e_2^+ , lam x ₂ .lam x ₁ . (push [x ₁ , x ₂])
fst e	\rightsquigarrow e^+ , push 0, idx
snd e	\rightsquigarrow e^+ , push 1, idx
$\lambda x : \tau.e$	\rightsquigarrow push (thunk lam x. e^+)
$e_1 e_2$	\rightsquigarrow e_1^+, e_2^+ , SWAP, call
ref e	\rightsquigarrow e^+ , alloc
!e	\rightsquigarrow e^+ , read
$e_1 := e_2$	\rightsquigarrow e_1^+, e_2^+ , write, push 0
$(\langle e \rangle_\tau)$	\rightsquigarrow e^+ , $C_{\tau \mapsto \tau}$

e \rightsquigarrow P

n	\rightsquigarrow push n
x	\rightsquigarrow push x
[e ₁ , ..., e _n]	\rightsquigarrow e_1^+, \dots, e_n^+ , lam x _n lam x ₁ . (push [x ₁ , ..., x _n])
$e_1[e_2]$	\rightsquigarrow e_1^+, e_2^+ , idx
if0 e e ₁ e ₂	\rightsquigarrow e^+ , if0 $e_1^+ e_2^+$
$\lambda x : \tau.e$	\rightsquigarrow push (thunk lam x. e^+)
$e_1 e_2$	\rightsquigarrow e_1^+, e_2^+ , SWAP, call
$e_1 + e_2$	\rightsquigarrow e_1^+, e_2^+ , add
ref e	\rightsquigarrow e^+ , alloc
!e	\rightsquigarrow e^+ , read
$e_1 := e_2$	\rightsquigarrow e_1^+, e_2^+ , write, push 0
$(\langle e \rangle_\tau)$	\rightsquigarrow e^+ , $C_{\tau \mapsto \tau}$

Figure 4.4: Compilers for **RefHL** and **RefLL**.

`false` as any non-zero integer. Hence, conversions between `bool` and `int` are identities.

We convert pairs to two-element arrays, assuming both elements of the pair can be converted to the same type. Since pairs are represented by StackLang arrays, the conversion code projects out and converts each element before wrapping both back up in an array. If we provided a specialized conversion where the element types were already equivalent (say, `bool` and `int`), this code could be a no-op, since the structure of a pair and a two element array is equivalent in the target. To convert the other direction, we first have to ensure that the array only has two elements, failing otherwise. Otherwise, the conversion is analogous.

For sums, we use the tags 0 and 1, and as for `if`, we use `if0` to branch in the compilation of `match`. Therefore, we can choose if the `inl` and `inr` tags should be represented by 0 and 1, or by 0 and any other integer n . Given that tags could be added later, we choose the former, thus converting a sum to an array of integers is mostly a matter of converting the payload. In the other direction, we have to handle the case that the array is too short, and error.

The final case, between `ref bool` and `ref int`, is the reason for this case study. Intuitively, if you exchange pointers, any value of the new type can now be written at that address, and thus must have been compatible with the old type (as aliases could still exist). Thus, we require that `bool` and `int` are somehow “identical” in the target, so conversions are unnecessary.

SEMANTIC MODEL Declaring that a type `bool` is “identical” to `int` or that τ is convertible to τ' and providing the conversion code is not sufficient for soundness. In order to show that these conversions are sound, and indeed to understand which conversions are even possible, we define a model for source types that is inhabited by target terms. Since both languages compile to the same target, the range of their relations will be the same (i.e., composed of terms and values from StackLang), and thus we will be able to easily and directly compare the inhabitants of two types, one from each language.

Our model, which is a standard step-indexed unary logical relation for a language with mutable state (essentially following Ahmed (2004)), is presented in Figure 4.6.

We give value interpretations for each source type τ , written $\mathcal{V}[\![\tau]\!]$ as sets of target *values* v paired with *worlds* W that inhabit that type. A world W is comprised of a step index k and a *heap typing* Ψ , which maps locations to type interpretations in Typ . As is standard, Typ is the set of valid type interpretations, which must be closed under world extension. A future world W' extends W , written $W' \sqsupseteq W$, if W' has a potentially lower

$$\begin{array}{c}
\overline{C_{\text{bool} \mapsto \text{int}}, C_{\text{int} \mapsto \text{bool}} : \text{bool} \sim \text{int}} \\[10pt]
\overline{C_{\text{ref bool} \mapsto \text{ref int}}, C_{\text{ref int} \mapsto \text{ref bool}} : \text{ref bool} \sim \text{ref int}} \\[10pt]
\frac{C_{\tau_1 \mapsto \tau}, C_{\tau \mapsto \tau_1} : \tau_1 \sim \tau \quad C_{\tau_2 \mapsto \tau}, C_{\tau \mapsto \tau_2} : \tau_2 \sim \tau}{C_{\tau_1 \times \tau_2 \mapsto [\tau]}, C_{[\tau] \mapsto \tau_1 \times \tau_2} : \tau_1 \times \tau_2 \sim [\tau]} \\[10pt]
\frac{C_{\tau_1 \mapsto \text{int}}, C_{\text{int} \mapsto \tau_1} : \tau_1 \sim \text{int} \quad C_{\tau_2 \mapsto \text{int}}, C_{\text{int} \mapsto \tau_2} : \tau_2 \sim \text{int}}{C_{\tau_1 + \tau_2 \mapsto [\text{int}]}, C_{[\text{int}] \mapsto \tau_1 + \tau_2} : \tau_1 + \tau_2 \sim [\text{int}]} \\[10pt]
\begin{array}{ll}
C_{\text{bool} \mapsto \text{int}} & \triangleq . \\
C_{\text{int} \mapsto \text{bool}} & \triangleq . \\
C_{\text{ref bool} \mapsto \text{ref int}} & \triangleq . \\
C_{\text{ref int} \mapsto \text{ref bool}} & \triangleq . \\
C_{\tau_1 \times \tau_2 \mapsto [\tau]} & \triangleq \text{DUP, push 0, idx, } C_{\tau_1 \mapsto \tau}, \text{SWAP, push 1, idx, } C_{\tau_2 \mapsto \tau}, \\
& \text{lam } x_2.\text{lam } x_1.\text{push } [x_1, x_2] \\
C_{[\tau] \mapsto \tau_1 \times \tau_2} & \triangleq \text{DUP, len, push 2, SWAP, less?, if0 fail CONV,} \\
& \text{DUP, push 0, idx, } C_{\tau \mapsto \tau_1}, \text{SWAP, push 1, idx, } C_{\tau \mapsto \tau_2}, \\
& \text{lam } x_2.\text{lam } x_1.\text{push } [x_1, x_2] \\
C_{\tau_1 + \tau_2 \mapsto [\text{int}]} & \triangleq \text{DUP, push 1, idx, SWAP, push 0, idx, DUP,} \\
& \text{if0 (SWAP, } C_{\tau_1 \mapsto \text{int}} \text{) (SWAP, } C_{\tau_2 \mapsto \text{int}} \text{),} \\
& \text{lam } x_v.\text{lam } x_t.\text{push } [x_t, x_v] \\
C_{[\text{int}] \mapsto \tau_1 + \tau_2} & \triangleq \text{DUP, len, push 2, SWAP, less?, if0 fail CONV,} \\
& \text{DUP, push 1, idx, SWAP, push 0, idx, DUP,} \\
& \text{if0 (SWAP, } C_{\text{int} \mapsto \tau_1} \text{)} \\
& \quad (\text{DUP, push } -1, \text{ add, if0 (SWAP, } C_{\text{int} \mapsto \tau_2} \text{) fail CONV}), \\
& \quad \text{lam } x_v.\text{lam } x_t.\text{push } [x_t, x_v]
\end{array}
\end{array}$$

Figure 4.5: Conversions for **RefHL** and **RefLL**.

$$\begin{aligned}
AtomVal_n &= \{(W, v) \mid W \in World_n\} \\
World_n &= \{(k, \Psi) \mid k < n \wedge \Psi \subset HeapTy_k\} \\
HeapTy_n &= \{\ell \mapsto Typ_n, \dots\} \\
Typ_n &= \{R \in 2^{AtomVal_n} \mid \forall (W, v) \in R. \\
&\quad \forall W'. W \sqsubseteq W' \implies (W', v) \in R\} \\
\mathcal{V}[\text{unit}] &= \{(W, 0)\} \\
\mathcal{V}[\text{bool}] &= \{(W, \text{n})\} \\
\mathcal{V}[\tau_1 \times \tau_2] &= \{(W, [v_1, v_2]) \mid (W, v_1) \in \mathcal{V}[\tau_1] \wedge (W, v_2) \in \mathcal{V}[\tau_2]\} \\
\mathcal{V}[\tau_1 + \tau_2] &= \{(W, [0, v]) \mid (W, v) \in \mathcal{V}[\tau_1]\} \\
&\quad \cup \{(W, [1, v]) \mid (W, v) \in \mathcal{V}[\tau_2]\} \\
\mathcal{V}[\tau_1 \rightarrow \tau_2] &= \{(W, \text{thunk lam } x.P) \mid \forall v, W' \sqsupseteq W. (W', v) \in \mathcal{V}[\tau_1] \\
&\quad \implies (W', [x \mapsto v]P) \in \mathcal{E}[\tau_2]\} \\
\mathcal{V}[\text{ref } \tau] &= \{(W, \ell) \mid W.\Psi(\ell) = [\mathcal{V}[\tau]]_{W.k}\} \\
\mathcal{V}[\text{int}] &= \{(W, \text{n})\} \\
\mathcal{V}[[\tau]] &= \{(W, [v_1, \dots, v_n]) \mid (W, v_i) \in \mathcal{V}[\tau]\} \\
\mathcal{V}[\tau_1 \rightarrow \tau_2] &= \{(W, \text{thunk lam } x.P) \mid \forall v, W' \sqsupseteq W. (W', v) \in \mathcal{V}[\tau_1] \\
&\quad \implies (W', [x \mapsto v]P) \in \mathcal{E}[\tau_2]\} \\
\mathcal{V}[\text{ref } \tau] &= \{(W, \ell) \mid W.\Psi(\ell) = [\mathcal{V}[\tau]]_{W.k}\} \\
\mathcal{E}[\tau] &= \{(W, P) \mid \forall H: W, S \neq \text{Fail } _, H', S', j < W.k. \\
&\quad \langle H ; S ; P \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle \implies S' = \text{Fail } c \wedge c \in \{\text{CONV}, \text{IDX}\} \\
&\quad \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\tau]))\} \\
[[\Gamma; \Gamma \vdash e : \tau]] &\equiv \forall W \gamma_\Gamma. (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma] \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma] \\
&\implies (W, \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e^+))) \in \mathcal{E}[\tau] \\
[[\Gamma; \Gamma \vdash e : \tau]] &\equiv \forall W \gamma_\Gamma. (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma] \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma] \\
&\implies (W, \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e^+))) \in \mathcal{E}[\tau]
\end{aligned}$$

Figure 4.6: Logical relation for **RefHL** and **RefLL**.

step budget $j \leq W.k$ and all locations in $W.\Psi$ still have the same types (to approximation j).

Intuitively, $(W, v) \in \mathcal{V}[\tau]$ says that the target value v belongs to (or behaves like a value of) type τ in world W . For example, $\mathcal{V}[\text{unit}]$ is inhabited by 0 in any world. A more interesting case is $\mathcal{V}[\text{bool}]$, which is the set of all target integers, not just 0 and 1, though we could choose to define our model that way (provided we compiled `bools` to 0 or 1). An array $\mathcal{V}[[\tau]]$ is inhabited by an array of target values v_i in world W if each v_i is in $\mathcal{V}[\tau]$ with W .

Functions follow the standard pattern for logical relations, appropriately adjusted for our stack-based target language: $\mathcal{V}[\tau_1 \rightarrow \tau_2]$ is inhabited by values thunk $\lambda m x.P$ in world W if, for any future world W' and argument v in $\mathcal{V}[\tau_1]$ at that world, the result of substituting the argument into the body $([x \mapsto v]P)$ is in the expression relation at the result type $\mathcal{E}[\tau_2]$. Reference types $\mathcal{V}[\text{ref } \tau]$ are inhabited by a location ℓ in world W if the current world's heap typing $W.\Psi$ maps ℓ to the value relation $\mathcal{V}[\tau]$ approximated to the step index in the world $W.k$. (The j -approximation of a type, written $[\mathcal{V}[\tau]]_j$, restricts $\mathcal{V}[\tau]$ to inhabitants with worlds in $World_j$.)

Our expression relation $\mathcal{E}[\tau]$ defines when a program P in world W behaves as a computation of type τ . It says that for any heap H that satisfies the current world W , written $H : W$, and any non-Fail stack S , if the machine $\langle H ; S ; P \rangle$ terminates in j steps (where j is less than our step budget $W.k$), then either it ran to a non-type error or there exists some value v and some future world W' such that the resulting stack S' is the original stack with v on top, the resulting heap H' satisfies the future world W' and W' and v are in $\mathcal{V}[\tau]$.

At the bottom of Fig. 4.6, we show a syntactic shorthand, $[\Gamma; \Gamma \vdash e : \tau]$, for showing that well-typed source programs, when compiled and closed off with well-typed substitutions γ that map variables to target values, are in the expression relation. Note $\mathcal{G}[\Gamma]$ contains closing substitutions γ in world W that assign every $x : \tau \in \Gamma$ to a v such that $(W, v) \in \mathcal{V}[\tau]$.

With our logical relation in hand, we can now state formal properties about our convertibility judgments.

Lemma 4.0.1 (Convertibility Soundness).

If $\tau \sim \tau$, then $\forall (W, P) \in \mathcal{E}[\tau]. (W, (P, C_{\tau \mapsto \tau})) \in \mathcal{E}[\tau] \wedge \forall (W, P) \in \mathcal{E}[\tau]. (W, (P, C_{\tau \mapsto \tau})) \in \mathcal{E}[\tau]$.

Proof. We sketch the `ref bool` \sim `ref int` case; for the full proofs, which are mostly mechanical unfoldings of the definitions, see Appendix A. For `ref bool` \sim `ref int`, what we need to show is that given any expression in $\mathcal{E}[\text{ref bool}]$, if we apply the conversion (which does nothing), the result will be in $\mathcal{E}[\text{ref int}]$. That requires we show $\mathcal{V}[\text{ref bool}] = \mathcal{V}[\text{ref int}]$.

The value relation at a reference type says that if you look up the location ℓ in the heap typing of the world $(W.\Psi)$, you will get the value interpretation of the type. That means a `ref bool` must be a location ℓ that, in the model, points to the value interpretation of `bool` (i.e., $\mathcal{V}[\![\text{bool}]\!]$). In our model, this must be true for all future worlds, which makes sense for ML-style references. Thus, for this proof to go through, $\mathcal{V}[\![\text{bool}]\!]$ must be the same as $\mathcal{V}[\![\text{int}]\!]$, which it is. \square

Once we have proved Lemma 4.0.1, we can prove semantic type soundness in the standard two-step way for our entire system. First, for each source typing rule, we state and prove a compatibility lemma that is a semantic analog to that rule. We provide all of the lemma statements here; the proofs, which are quite mechanical, are provided in Appendix A.

Lemma 4.0.2 (Compat `()`). $\llbracket \mathbf{T}; \Gamma \vdash () : \text{unit} \rrbracket$

Proof. See A.0.7. \square

Lemma 4.0.3 (Compat `b`). $\mathbf{b} \in \mathbb{B} \implies \llbracket \mathbf{T}; \Gamma \vdash \mathbf{b} : \text{bool} \rrbracket$

Proof. See A.0.8. \square

Lemma 4.0.4 (Compat `x`). $\llbracket \mathbf{T}; \Gamma, \mathbf{x} : \tau \vdash \mathbf{x} : \tau \rrbracket$

Proof. See A.0.9. \square

Lemma 4.0.5 (Compat `inl e`). $\llbracket \mathbf{T}; \Gamma \vdash \mathbf{e} : \tau_1 \rrbracket \implies \llbracket \mathbf{T}; \Gamma \vdash \text{inl } \mathbf{e} : \tau_1 + \tau_2 \rrbracket$

Proof. See A.0.10. \square

Lemma 4.0.6 (Compat `inr e`). $\llbracket \mathbf{T}; \Gamma \vdash \mathbf{e} : \tau_2 \rrbracket \implies \llbracket \mathbf{T}; \Gamma \vdash \text{inr } \mathbf{e} : \tau_1 + \tau_2 \rrbracket$

Proof. See A.0.11. \square

Lemma 4.0.7 (Compat `if`). $\llbracket \mathbf{T}; \Gamma \vdash \mathbf{e} : \text{bool} \rrbracket \wedge \llbracket \mathbf{T}; \Gamma \vdash \mathbf{e}_1 : \tau \rrbracket \wedge \llbracket \mathbf{T}; \Gamma \vdash \mathbf{e}_2 : \tau \rrbracket \implies \llbracket \mathbf{T}; \Gamma \vdash \text{if } \mathbf{e} \ \mathbf{e}_1 \ \mathbf{e}_2 : \tau \rrbracket$

Proof. See A.0.12. \square

Lemma 4.0.8 (Compat `match`).

$$\begin{aligned} & \llbracket \mathbf{T}; \Gamma \vdash \mathbf{e} : \tau_1 + \tau_2 \rrbracket \wedge \llbracket \mathbf{T}; \Gamma, \mathbf{x} : \tau_1 \vdash \mathbf{e}_1 : \tau \rrbracket \wedge \llbracket \mathbf{T}; \Gamma, \mathbf{y} : \tau_2 \vdash \mathbf{e}_2 : \tau \rrbracket \\ & \implies \llbracket \mathbf{T}; \Gamma \vdash \text{match } \mathbf{e} \ \mathbf{x}\{\mathbf{e}_1\} \ \mathbf{y}\{\mathbf{e}_2\} : \tau_1 + \tau_2 \rrbracket \end{aligned}$$

Proof. See A.0.13. \square

Lemma 4.0.9 (Compat (e_1, e_2)). $\llbracket \Gamma; \Gamma \vdash e_1 : \tau_1 \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_2 : \tau_2 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2 \rrbracket$

Proof. See A.0.14. \square

Lemma 4.0.10 (Compat $\text{fst } e$). $\llbracket \Gamma; \Gamma \vdash e : \tau_1 \times \tau_2 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash \text{fst } e : \tau_1 \rrbracket$

Proof. See A.0.15. \square

Lemma 4.0.11 (Compat $\text{snd } e$). $\llbracket \Gamma; \Gamma \vdash e : \tau_1 \times \tau_2 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash \text{snd } e : \tau_2 \rrbracket$

Proof. See A.0.16. \square

Lemma 4.0.12 (Compat $\lambda x : \tau. e$). $\llbracket \Gamma; \Gamma, x : \tau_1 \vdash e : \tau_2 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2 \rrbracket$

Proof. See A.0.17. \square

Lemma 4.0.13 (Compat $e_1 e_2$). $\llbracket \Gamma; \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_2 : \tau_1 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash e_1 e_2 : \tau_2 \rrbracket$

Proof. See A.0.18. \square

Lemma 4.0.14 (Compat $\text{ref } e$). $\llbracket \Gamma; \Gamma \vdash e : \tau \rrbracket \implies \llbracket \Gamma; \Gamma \vdash \text{ref } e : \text{ref } \tau \rrbracket$

Proof. See A.0.19. \square

Lemma 4.0.15 (Compat $!e$). $\llbracket \Gamma; \Gamma \vdash e : \text{ref } \tau \rrbracket \implies \llbracket \Gamma; \Gamma \vdash !e : \tau \rrbracket$

Proof. See A.0.20. \square

Lemma 4.0.16 (Compat $e_1 := e_2$). $\llbracket \Gamma; \Gamma \vdash e_1 : \text{ref } \tau \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_2 : \tau \rrbracket \implies \llbracket \Gamma; \Gamma \vdash e_1 := e_2 : \text{unit} \rrbracket$

Proof. See A.0.21. \square

Lemma 4.0.17 (Compat $(e)_\tau$). $\llbracket \Gamma; \Gamma \vdash e : \tau \rrbracket \wedge \tau \sim \tau \implies \llbracket \Gamma; \Gamma \vdash (e)_\tau : \tau \rrbracket$

Proof. See A.0.22. \square

Lemma 4.0.18 (Compat n). $\llbracket \Gamma; \Gamma \vdash n : \text{int} \rrbracket$

Proof. See A.0.23. \square

Lemma 4.0.19 (Compat x). $\llbracket \Gamma; \Gamma, x : \tau \vdash x : \tau \rrbracket$

Proof. See A.0.24. \square

Lemma 4.0.20 (Compat $[e_1, \dots, e_n]$). $\llbracket \Gamma; \Gamma \vdash e_1 : \tau \rrbracket \wedge \dots \wedge \llbracket \Gamma; \Gamma \vdash e_n : \tau \rrbracket \implies \llbracket \Gamma; \Gamma \vdash [e_1, \dots, e_n] : [\tau] \rrbracket$

Proof. See A.0.25. \square

Lemma 4.0.21 (Compat $e_1[e_2]$). $\llbracket \Gamma; \Gamma \vdash e_1 : [\tau] \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_2 : \text{int} \rrbracket \implies \llbracket \Gamma; \Gamma \vdash e_1[e_2] : \tau \rrbracket$

Proof. See A.0.26. \square

Lemma 4.0.22 (Compat if0). $\llbracket \Gamma; \Gamma \vdash e : \text{int} \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_1 : \tau \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_2 : \tau \rrbracket \implies \llbracket \Gamma; \Gamma \vdash \text{if0 } e \ e_1 \ e_2 : \tau \rrbracket$

Proof. See A.0.27. \square

Lemma 4.0.23 (Compat $\lambda x : \tau. e$). $\llbracket \Gamma; \Gamma, x : \tau_1 \vdash e : \tau_2 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2 \rrbracket$

Proof. See A.0.28. \square

Lemma 4.0.24 (Compat $e_1 e_2$). $\llbracket \Gamma; \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_2 : \tau_1 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash e_1 e_2 : \tau_2 \rrbracket$

Proof. See A.0.29. \square

Lemma 4.0.25 (Compat $e_1 + e_2$). $\llbracket \Gamma; \Gamma \vdash e_1 : \text{int} \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_2 : \text{int} \rrbracket \implies \llbracket \Gamma; \Gamma \vdash e_1 + e_2 : \text{int} \rrbracket$

Proof. See A.0.30. \square

Lemma 4.0.26 (Compat $\text{ref } e$). $\llbracket \Gamma; \Gamma \vdash e : \tau \rrbracket \implies \llbracket \Gamma; \Gamma \vdash \text{ref } e : \text{ref } \tau \rrbracket$

Proof. See A.0.31. \square

Lemma 4.0.27 (Compat $!e$). $\llbracket \Gamma; \Gamma \vdash e : \text{ref } \tau \rrbracket \implies \llbracket \Gamma; \Gamma \vdash !e : \tau \rrbracket$

Proof. See A.0.32. \square

Lemma 4.0.28 (Compat $e_1 := e_2$). $\llbracket \Gamma; \Gamma \vdash e_1 : \text{ref } \tau \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_2 : \tau \rrbracket \implies \llbracket \Gamma; \Gamma \vdash e_1 := e_2 : \text{int} \rrbracket$

Proof. See A.0.33. \square

Lemma 4.0.29 (Compat $(\text{e})_\tau$). $\llbracket \Gamma; \Gamma \vdash e : \tau \rrbracket \wedge \tau \sim \tau \implies \llbracket \Gamma; \Gamma \vdash (\text{e})_\tau : \tau \rrbracket$

Proof. See A.0.34. \square

Once we have all compatibility lemmas we can prove the following theorems:

Theorem 4.0.30 (Fundamental Property).

If $\Gamma; \Gamma \vdash e : \tau$ then $\llbracket \Gamma; \Gamma \vdash e : \tau \rrbracket$ and if $\Gamma; \Gamma \vdash e : \tau$ then $\llbracket \Gamma; \Gamma \vdash e : \tau \rrbracket$.

Proof. By induction on the typing derivations, using the compatibility lemmas. \square

Theorem 4.0.31 (Type Safety for **RefLL**). *If $\cdot; \cdot \vdash e : \tau$ then for any $H : W$, if $\langle H ; \cdot ; e^+ \rangle \xrightarrow{*} \langle H' ; S' ; P' \rangle$, then either $\langle H' ; S' ; P' \rangle \rightarrow \langle H'' ; S'' ; P'' \rangle$, or $P' = \cdot$ and either $S' = \text{Fail } c$ for some $c \in \{\text{CONV}, \text{IDX}\}$ or $S' = v$.*

Proof. Suppose $\langle H ; \cdot ; e^+ \rangle \xrightarrow{n} \langle H' ; S' ; P' \rangle$ for some natural number n . Then, either $\langle H' ; S' ; P' \rangle \rightarrow \langle H'' ; S'' ; P'' \rangle$ or $\langle H' ; S' ; P' \rangle$ is irreducible. If $\langle H' ; S' ; P' \rangle$ is irreducible, then $P' = \cdot$ by Lemma A.0.1.

Next, by the Fundamental Property, since e typechecks under empty environments, $((n+1, \emptyset), e^+) \in \mathcal{E}[\tau]..$ Thus, since $n < n+1$ and $\langle H ; \cdot ; e^+ \rangle \xrightarrow{n} \langle H' ; S' ; \cdot \rangle$, we find that either $S' = \text{Fail } c$ for some $c \in \text{OKERR}$ or $S' = \cdot, v$, as was to be proven. \square

Theorem 4.0.32 (Type Safety for **RefHL**). *If $\cdot; \cdot \vdash e : \tau$ then for any $H : W$, if $\langle H ; \cdot ; e^+ \rangle \xrightarrow{*} \langle H' ; S' ; P' \rangle$, then either $\langle H' ; S' ; P' \rangle \rightarrow \langle H'' ; S'' ; P'' \rangle$, or $P' = \cdot$ and either $S' = \text{Fail } c$ for some $c \in \{\text{CONV}, \text{IDX}\}$ or $S' = v$.*

Proof. This proof is identical to that of **RefLL**. \square

DISCUSSION In addition to directly passing across pointers, there are two alternative conversion strategies, both of which our framework would accommodate: first, we could create a new location and copy and convert the data. This would allow the more flexible convertibility which does not require references to “identical” types, but would not allow aliasing, which may be desirable. Second, rather than converting `ref` τ and `ref` τ , we could instead convert $(\text{unit} \rightarrow \tau) \times (\tau \rightarrow \text{unit})$ and $(\text{unit} \rightarrow \tau) \times (\tau \rightarrow \text{unit})$ (assuming we had pairs)—i.e., read/write proxies to the reference (similar to that used in (Dimoulas et al., 2012)). This allows aliasing, i.e., both languages reading / writing to the same location, and is sound for arbitrary convertibility relations, but it comes at a significant runtime cost, as we introduce overhead at each read / write.

The choice to use the encoding described in this case study, or either of these options, is not, of course, exclusive—we could provide different options for different types in the same system, depending on the performance characteristics we need.

5

CASE STUDY: AFFINE FUNCTIONS

In our second case study, we consider an affine language, **Affi**, interacting with an unrestricted one, **MiniML**. Since **Affi** is affine rather than linear, its substructural features can be enforced dynamically. In particular, we adopt the classic technique, described in (Tov and Pucella, 2010), where affine resources are protected behind thunks with stateful flags that indicate failure on a second force.

This case study includes two variants: first, we develop a simple one, where our affine language **Affi** is compiled to target code that enforces, at runtime, that affine variables are used at most once. This allows for unrestricted and affine code to be safely mixed, as the type invariant of the affine language will be enforced at runtime. However, this calling convention comes at a runtime cost: even **Affi** code that will never interact with **MiniML** is still subject to the overhead of checking variable use, checks that statically we know will always succeed. This motivates the second variant, where we distinguish between affine function types (dynamic) that may interact with **MiniML** code and those (static) ones that will not. Only the dynamic affine variables need be protected at runtime, as only they will possibly be interacting with code that is not subject to the affine static type discipline.

See Figure 5.7 for how affine variables are compiled.

LANGUAGES Both **MiniML** and **Affi** are compiled to the same untyped Scheme-like functional target LCVM. As described above, we have two variants of **Affi**, dynamic and static (which we will label **Affi**_○ and **Affi**_● where needed for clarity), where the latter is an extension of the former. We present the syntax of all three languages in Figure 5.1. Our target LCVM is an untyped lambda calculus with functions, pattern matching, mutable references, and a standard operational semantics defined via steps $\langle H, e \rangle \rightarrow \langle H', e' \rangle$ over heap and expression pairs, presented in Figure 5.2. The operational semantics of our source languages are defined solely via compilation to LCVM.

For the static semantics of **MiniML** and **Affi**, as in the previous case study, we will support open terms across language boundaries, and thus need to carry environments for both languages. In this case study, we can protect affine resources that cross boundaries. That means that our affine environments Ω need to be split, even within **MiniML**, to respect the affine invariant. We present the static semantics of **MiniML** in Figure 5.5, and the static semantics for **Affi** in Figures 5.3 (**Affi**_○) and 5.4 (**Affi**_●).

Affi_○	
Type τ	::= unit bool int $\tau \multimap \tau$! τ $\tau \& \tau$ $\tau \otimes \tau$
Expr. e	::= () true false n x a _○ $\lambda a_{\bullet} : \tau . e$ e e (e) _τ !v let !x = e in e' ⟨e, e'⟩ e.1 e.2 (e, e) let (a _○ , a' _○) = e in e'
Value v	::= () $\lambda a_{\bullet} : \tau . e$!v ⟨e, e'⟩ (v, v')
Mode o	::= ○
Affi_●	
Type τ	::= ... $\tau \multimap \tau$
Mode o	::= ○ ●
MiniML	
Type τ	::= unit int $\tau \times \tau$ $\tau + \tau$ $\tau \rightarrow \tau$ $\forall \alpha . \tau$ α ref τ
Expr. e	::= () n x (e, e) fst e snd e inl e inr e match e x{e} y{e} $\lambda x : \tau . e$ e e $\Lambda \alpha . e$ e[τ] ref e !e e := e (e) _τ
LCVM	
Expr e	::= () n ℓ x (e, e) fst e snd e inl e inr e if e {e} {e} match e x{e} y{e} let x = e in e $\lambda x : e$ e e ref e !e e := e fail c
Evaluation Context K	::= [] (K, e) (v, K) inl K inr K match K x{e} y{e} if K {e} {e} let x = K in e K e v K ref K !K K := e v := K
Values v	::= () n ℓ (v, v) $\lambda x . e$
Err Code c	::= TYPE CONV

Figure 5.1: Syntax for **Affi** (dynamic and static), **MiniML** and LCVM.

$$\begin{array}{c}
\frac{}{\langle H, \text{fst } (v, v') \rangle \rightarrow \langle H, v \rangle} \quad \frac{v \neq (v_1, v_2)}{\langle H, \text{fst } v \rangle \rightarrow \langle H, \text{fail TYPE} \rangle} \\
\\
\frac{}{\langle H, \text{snd } (v', v) \rangle \rightarrow \langle H, v \rangle} \quad \frac{v \neq (v_1, v_2)}{\langle H, \text{snd } v \rangle \rightarrow \langle H, \text{fail TYPE} \rangle} \\
\\
\frac{}{\langle H, \text{if } 0 \{e_1\} \{e_2\} \rangle \rightarrow \langle H, e_1 \rangle} \quad \frac{n \neq 0}{\langle H, \text{if } n \{e_1\} \{e_2\} \rangle \rightarrow \langle H, e_2 \rangle} \\
\\
\frac{v \notin \mathbb{Z}}{\langle H, \text{if } v \{e_1\} \{e_2\} \rangle \rightarrow \langle H, \text{fail TYPE} \rangle} \\
\\
\frac{}{\langle H, \text{match } \text{inl } v \times \{e_1\} y \{e_2\} \rangle \rightarrow \langle H, [x \mapsto v] e_1 \rangle} \\
\\
\frac{}{\langle H, \text{match } \text{inr } v \times \{e_1\} y \{e_2\} \rangle \rightarrow \langle H, [y \mapsto v] e_2 \rangle} \\
\\
\frac{v \notin \{\text{inr } v', \text{inl } v'\}}{\langle H, \text{match } v \times \{e_1\} y \{e_2\} \rangle \rightarrow \langle H, \text{fail TYPE} \rangle} \quad \frac{}{\langle H, \text{let } x = v \text{ in } e \rangle \rightarrow \langle H, [x \mapsto v] e \rangle} \\
\\
\frac{}{\langle H, \lambda x \{e_b\} v \rangle \rightarrow \langle H, [x \mapsto v] e_b \rangle} \quad \frac{v \neq \lambda x \{e\}}{\langle H, v v' \rangle \rightarrow \langle H, \text{fail TYPE} \rangle} \\
\\
\frac{\text{fresh } \ell}{\langle H, \text{ref } v \rangle \rightarrow \langle H[\ell \mapsto v], \ell \rangle} \quad \frac{H[\ell] = v}{\langle H, !\ell \rangle \rightarrow \langle H, v \rangle} \quad \frac{v \neq \ell}{\langle H, !v \rangle \rightarrow \langle H, \text{fail TYPE} \rangle} \\
\\
\frac{}{\langle H, \ell := v \rangle \rightarrow \langle H[\ell \mapsto v], () \rangle} \quad \frac{v \neq \ell}{\langle H, v := v' \rangle \rightarrow \langle H, \text{fail TYPE} \rangle} \\
\\
\hline
\\
\frac{\langle H, e \rangle \rightarrow \langle H', e' \rangle}{\langle H, K[e] \rangle \rightarrow \langle H', K[e'] \rangle} \quad \frac{K \neq [\cdot]}{\langle H, K[\text{fail } c] \rangle \rightarrow \langle H, \text{fail } c \rangle}
\end{array}$$

Figure 5.2: Dynamic semantics for LCVM.

$$\begin{array}{c}
\frac{\mathbf{a} : \tau \in \Omega}{\Delta; \Gamma; \Gamma; \Omega \vdash \mathbf{a} : \tau} \quad \frac{\mathbf{x} : \tau \in \Gamma}{\Delta; \Gamma; \Gamma; \Omega \vdash \mathbf{x} : \tau} \quad \frac{}{\Delta; \Gamma; \Gamma; \Omega \vdash () : \text{unit}} \\
\hline
\frac{}{\Delta; \Gamma; \Gamma; \Omega \vdash \mathbf{n} : \text{int}} \quad \frac{}{\Delta; \Gamma; \Gamma; \Omega \vdash \text{true} : \text{bool}} \quad \frac{}{\Delta; \Gamma; \Gamma; \Omega \vdash \text{false} : \text{bool}} \\
\hline
\frac{\Delta; \Gamma; \Gamma; \Omega[\mathbf{a} := \tau_1] \vdash \mathbf{e} : \tau_2}{\Delta; \Gamma; \Gamma; \Omega \vdash \lambda \mathbf{a} : \tau_1. \mathbf{e} : \tau_1 \multimap \tau_2} \\
\hline
\frac{\Omega = \Omega_1 \uplus \Omega_2 \quad \Delta; \Gamma; \Gamma; \Omega_1 \vdash \mathbf{e}_1 : \tau_1 \multimap \tau_2 \quad \Delta; \Gamma; \Gamma; \Omega_2 \vdash \mathbf{e}_2 : \tau_1}{\Delta; \Gamma; \Gamma; \Omega \vdash \mathbf{e}_1 \ \mathbf{e}_2 : \tau_2} \\
\hline
\frac{\Delta; \Gamma; \Gamma; \cdot \vdash \mathbf{v} : \tau}{\Delta; \Gamma; \Gamma; \cdot \vdash !\mathbf{v} : !\tau} \\
\hline
\frac{\Omega = \Omega_1 \uplus \Omega_2 \quad \Delta; \Gamma; \Gamma; \Omega_1 \vdash \mathbf{e} : !\tau \quad \Delta; \Gamma; \Gamma[\mathbf{x} := \tau]; \Omega_2 \vdash \mathbf{e}' : \tau'}{\Delta; \Gamma; \Gamma; \Omega \vdash \text{let } !\mathbf{x} = \mathbf{e} \text{ in } \mathbf{e}'}
\end{array}$$

$$\begin{array}{c}
\frac{\Delta; \Gamma; \Gamma; \Omega \vdash \mathbf{e}_1 : \tau_1 \quad \Delta; \Gamma; \Gamma; \Omega \vdash \mathbf{e}_2 : \tau_2}{\Delta; \Gamma; \Gamma; \Omega \vdash \langle \mathbf{e}_1, \mathbf{e}_2 \rangle : \tau_1 \& \tau_2} \quad \frac{\Delta; \Gamma; \Gamma; \Omega \vdash \mathbf{e} : \tau_1 \& \tau_2}{\Delta; \Gamma; \Gamma; \Omega \vdash \mathbf{e}.1 : \tau_1} \\
\hline
\frac{\Delta; \Gamma; \Gamma; \Omega \vdash \mathbf{e} : \tau_1 \& \tau_2}{\Delta; \Gamma; \Gamma; \Omega \vdash \mathbf{e}.2 : \tau_2}
\end{array}$$

$$\frac{\Omega = \Omega_1 \uplus \Omega_2 \quad \Delta; \Gamma; \Gamma; \Omega_1 \vdash \mathbf{e}_1 : \tau_1 \quad \Delta; \Gamma; \Gamma; \Omega_2 \vdash \mathbf{e}_2 : \tau_2}{\Delta; \Gamma; \Gamma; \Omega \vdash (\mathbf{e}_1, \mathbf{e}_2) : \tau_1 \otimes \tau_2}$$

$$\frac{\Omega = \Omega_1 \uplus \Omega_2 \quad \Delta; \Gamma; \Gamma; \Omega_1 \vdash \mathbf{e} : \tau_1 \otimes \tau_2 \quad \Delta; \Gamma; \Gamma; \Omega_2[\mathbf{a} := \tau_1, \mathbf{a}' := \tau_1] \vdash \mathbf{e}' : \tau'}{\Delta; \Gamma; \Gamma; \Omega \vdash \text{let } (\mathbf{a}, \mathbf{a}') = \mathbf{e} \text{ in } \mathbf{e}' : \tau'}$$

$$\frac{\Delta; \Gamma; \Gamma; \Omega \vdash \mathbf{e} : \tau \quad _ : \tau \sim \tau}{\Delta; \Gamma; \Gamma; \Omega \vdash (\mathbf{e})_\tau : \tau}$$

Figure 5.3: Static semantics for for **Aff** (dynamic).

$$\begin{array}{c}
\frac{\mathbf{a}_\circ : \tau \in \Omega}{\Delta; \Gamma; \Omega \vdash \mathbf{a}_\circ : \tau} \quad \frac{\mathbf{a}_\bullet : \tau \in \Omega}{\Delta; \Gamma; \Omega \vdash \mathbf{a}_\bullet : \tau} \quad \frac{x : \tau \in \Gamma}{\Delta; \Gamma; \Omega \vdash x : \tau} \\
\\
\frac{}{\Delta; \Gamma; \Omega \vdash () : \text{unit}} \quad \frac{}{\Delta; \Gamma; \Omega \vdash n : \text{int}} \quad \frac{}{\Delta; \Gamma; \Omega \vdash \text{true} : \text{bool}} \\
\\
\frac{\Delta; \Gamma; \Omega[\mathbf{a}_\circ := \tau_1] \vdash e : \tau_2 \quad \text{no}_\bullet(\Omega)}{\Delta; \Gamma; \Omega \vdash \lambda \mathbf{a}_\circ : \tau_1. e : \tau_1 \multimap \tau_2} \\
\\
\frac{\Omega = \Omega_1 \uplus \Omega_2 \quad \Delta; \Gamma; \Omega_1 \vdash e_1 : \tau_1 \multimap \tau_2 \quad \Delta; \Gamma; \Omega_2 \vdash e_2 : \tau_1}{\Delta; \Gamma; \Omega \vdash e_1 e_2 : \tau_2} \\
\\
\frac{\Omega = \Omega_1 \uplus \Omega_2 \quad \Delta; \Gamma; \Omega_1 \vdash e_1 : \tau_1 \multimap \tau_2 \quad \Delta; \Gamma; \Omega_2 \vdash e_2 : \tau_1}{\Delta; \Gamma; \Omega \vdash e_1 e_2 : \tau_2} \\
\\
\frac{\Delta; \Gamma; \cdot \vdash v : \tau}{\Delta; \Gamma; \cdot \vdash !v : !\tau} \\
\\
\frac{\Omega = \Omega_1 \uplus \Omega_2 \quad \Delta; \Gamma; \Omega_1 \vdash e : !\tau \quad \Delta; \Gamma; [\mathbf{x} := \tau]; \Omega_2 \vdash e' : \tau'}{\Delta; \Gamma; \Omega \vdash \text{let } !x = e \text{ in } e'} \\
\\
\frac{\Delta; \Gamma; \Omega \vdash e_1 : \tau_1 \quad \Delta; \Gamma; \Omega \vdash e_2 : \tau_2}{\Delta; \Gamma; \Omega \vdash \langle e_1, e_2 \rangle : \tau_1 \& \tau_2} \quad \frac{\Delta; \Gamma; \Omega \vdash e : \tau_1 \& \tau_2}{\Delta; \Gamma; \Omega \vdash e.1 : \tau_1} \\
\\
\frac{\Delta; \Gamma; \Omega \vdash e : \tau_1 \& \tau_2}{\Delta; \Gamma; \Omega \vdash e.2 : \tau_2} \\
\\
\frac{\Omega = \Omega_1 \uplus \Omega_2 \quad \Delta; \Gamma; \Omega_1 \vdash e_1 : \tau_1 \quad \Delta; \Gamma; \Omega_2 \vdash e_2 : \tau_2}{\Delta; \Gamma; \Omega \vdash (e_1, e_2) : \tau_1 \otimes \tau_2} \\
\\
\frac{\Delta; \Gamma; \Omega \vdash e : \tau_1 \otimes \tau_2 \quad \Omega = \Omega_1 \uplus \Omega_2 \quad \Delta; \Gamma; \Omega_2[a := \tau_1, a' := \tau_1] \vdash e' : \tau'}{\Delta; \Gamma; \Omega \vdash \text{let } (a_\bullet, a'_\bullet) = e \text{ in } e' : \tau'} \\
\\
\frac{\Delta; \Gamma; \Omega \vdash e : \tau \quad _ : \tau \sim \tau}{\Delta; \Gamma; \Omega \vdash (e)_\tau : \tau}
\end{array}$$

Figure 5.4: Static semantics for for **Affi** (static).

$$\begin{array}{c}
\frac{}{\Omega; \Gamma; \Delta; \Gamma \vdash () : \text{unit}} \quad \frac{}{\Omega; \Gamma; \Delta; \Gamma \vdash \mathbb{Z} : \text{int}} \quad \frac{\Delta \vdash \tau \quad x : \tau \in \Gamma}{\Omega; \Gamma; \Delta; \Gamma \vdash x : \tau} \\
\\
\frac{\Omega_1; \Gamma; \Delta; \Gamma \vdash e_1 : \tau_1 \quad \Omega_2; \Gamma; \Delta; \Gamma \vdash e_2 : \tau_2}{\Omega_1 \uplus \Omega_2; \Gamma; \Delta; \Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2} \quad \frac{}{\Omega; \Gamma; \Delta; \Gamma \vdash \text{fst } e : \tau_1} \\
\\
\frac{\Omega; \Gamma; \Delta; \Gamma \vdash e : \tau_1 \times \tau_2}{\Omega; \Gamma; \Delta; \Gamma \vdash \text{snd } e : \tau_2} \quad \frac{\Delta \vdash \tau_2 \quad \Omega; \Gamma; \Delta; \Gamma \vdash e : \tau_1}{\Omega; \Gamma; \Delta; \Gamma \vdash \text{inl } e : \tau_1 + \tau_2} \\
\\
\frac{\Delta \vdash \tau_1 \quad \Omega; \Gamma; \Delta; \Gamma \vdash e : \tau_2}{\Omega; \Gamma; \Delta; \Gamma \vdash \text{inr } e : \tau_1 + \tau_2} \\
\\
\frac{\Omega_1; \Gamma; \Delta; \Gamma \vdash e : \tau_1 + \tau_2 \quad \Omega_2; \Gamma; \Delta; \Gamma[x : \tau_1] \vdash e_1 : \tau \quad \Omega_2; \Gamma; \Delta; \Gamma[y : \tau_2] \vdash e_2 : \tau}{\Omega_1 \uplus \Omega_2; \Gamma; \Delta; \Gamma \vdash \text{match } e \text{ x } e_1 \text{ y } e_2 : \tau} \\
\\
\frac{\Omega; \Gamma; \Delta; \Gamma[x : \tau_1] \vdash e : \tau_2}{\Omega; \Gamma; \Delta; \Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2} \\
\\
\frac{\Omega_1; \Gamma; \Delta; \Gamma \vdash e : \tau' \rightarrow \tau \quad \Omega_2; \Gamma; \Delta; \Gamma \vdash e' : \tau}{\Omega_1 \uplus \Omega_2; \Gamma; \Delta; \Gamma \vdash e e' : \tau} \quad \frac{}{\Omega; \Gamma; \Delta; \Gamma \vdash \Lambda \alpha. e : \forall \alpha. \tau} \\
\\
\frac{\Omega; \Gamma; \Delta \vdash \tau \quad \Omega; \Gamma; \Delta; \Gamma \vdash e : \forall \alpha. \tau}{\Omega; \Gamma; \Delta; \Gamma \vdash e[\tau'] : \tau[\tau'/\alpha]} \quad \frac{\Omega; \Gamma; \Delta; \Gamma \vdash e : \tau}{\Omega; \Gamma; \Delta; \Gamma \vdash \text{ref } e : \text{ref } \tau} \\
\\
\frac{\Omega; \Gamma; \Delta; \Gamma \vdash e : \text{ref } \tau}{\Omega; \Gamma; \Delta; \Gamma \vdash !e : \tau} \quad \frac{\Omega_1; \Gamma; \Delta; \Gamma \vdash e_1 : \text{ref } \tau \quad \Omega_2; \Gamma; \Delta; \Gamma \vdash e_2 : \tau}{\Omega_1 \uplus \Omega_2; \Gamma; \Delta; \Gamma \vdash e_1 := e_2 : \text{unit}} \\
\\
\frac{\Omega; \Gamma; \Delta; \Gamma \vdash e : \tau \quad _ : \tau \sim \tau}{\Omega; \Gamma; \Delta; \Gamma \vdash (e)_\tau : \tau}
\end{array}$$

Figure 5.5: Static semantics for for MiniML.

Affi. exists in order to avoid unnecessary dynamic enforcement of at-most-once variable use. To facilitate that, it has two kinds of affine function types: --o and --\bullet . This creates a distinction between **Affi.** functions (and thus bindings) that may be passed across the boundary (our “dynamic” affine arrows --o , written with a hollow circle and bind dynamic affine variables \mathbf{a}_{o}), and ones that will only ever be used within **Affi** (our “static” affine arrows --\bullet , written with a solid circle and bind static affine variables \mathbf{a}_{\bullet}).

We can see in both Figure 5.3 and Figure 5.4 how **Affi**’s affine-variable environment Ω is maintained: variables are introduced by lambda and tensor-destructuring let, and environments are split across subterms, but all bindings are not required to be used, as we can see, in the variable rule. Since affine resources can exist within unrestricted **Miniml** terms, our affine environments Ω need to be split, even in **Miniml** typing rules, shown in Figure 5.5.

Our compilers, presented in Figures 5.6 and 5.7, are primarily interesting in the cases that address affine bindings; otherwise, they do standard type erasure for polymorphic types, etc. We use a compiler macro, $\text{once}(\cdot)$, which expands to a nullary function closing over a freshly-allocated reference—called a *flag*—initialized to 1. When called, this function fails if the flag is 0. Otherwise, it sets the flag to 0 and returns the macro’s argument. Throughout this chapter, we will use the constants **UNUSED** and **USED** for 1 and 0. We use $\text{once}(\cdot)$ when introducing affine bindings, and then we compile uses of affine variable to expressions that force the thunk. Unrestricted **Affi** variables \mathbf{x} and variables from **Miniml** are unaffected by this strategy. And, of course, static bindings in **Affi.** do not introduce the runtime check, since it is unnecessary.

CONVERTIBILITY We define convertibility relations and conversions for **Affi** and **Miniml** in Figure 5.8. Note that since the additions to **Affi.** to make **Affi.** involve unconvertible types, the two variants share these definitions. In the figure we define base type conversions between **unit** and **unit**, **bool** and **int**, tensors and pairs, and between \rightarrow and --o . The last is most interesting and challenging. Our compiler is designed to support affine code being mixed directly with unrestricted code. Intuitively, an affine function should be able to behave as an unrestricted one, but the other direction is harder to accomplish, and higher-order functions mean both must be addressed at once. In order to account for this, we convert $\tau_1 \text{--o } \tau_2$ not to $\tau_1 \rightarrow \tau_2$ (not even for some convertible argument/return types), but rather to $(\mathbf{unit} \rightarrow \tau_1) \rightarrow \tau_2$. That is, to a **Miniml** function that expects its argument to be a thunk containing a τ_1 rather than a τ_1 directly. Provided that the thunk fails if invoked more than once, we can ensure, dynamically, that a **Miniml** function with that type behaves as an **Affi** function of a

$\lambda x : \tau. e$	\rightsquigarrow	$\lambda x. \{e^+\}$
$e_1 \ e_2$	\rightsquigarrow	$e_1^+ \ e_2^+$
(e_1, e_2)	\rightsquigarrow	(e_1^+, e_2^+)
$\text{fst } e$	\rightsquigarrow	$\text{fst } e^+$
$\text{snd } e$	\rightsquigarrow	$\text{snd } e^+$
$\text{inl } e$	\rightsquigarrow	$\text{inl } e^+$
$\text{inr } e$	\rightsquigarrow	$\text{inr } e^+$
$\text{match } e$	\rightsquigarrow	$\text{match } e^+$
$x\{e_1\} \ y\{e_2\}$	\rightsquigarrow	$x\{e_1^+\} \ y\{e_2^+\}$
$\Lambda \alpha. e$	\rightsquigarrow	$\lambda_- \{e^+\}$
$e[\tau]$	\rightsquigarrow	$e^+ ()$
$\text{ref } e$	\rightsquigarrow	$\text{ref } e^+$
$!e$	\rightsquigarrow	$!e^+$
$e_1 := e_2$	\rightsquigarrow	$e_1^+ := e_2^+$
$(e)_\tau$	\rightsquigarrow	$C_{\tau \mapsto \tau}(e^+)$

Figure 5.6: Compiler for [MiniML](#).

related type. These invariants are ensured by appropriate wrapping and use of the compiler macro `once(·)`.

EXAMPLES. Using our conversions, we investigate several small examples, presented (with their compilations) in Figure 5.9. Program *P1* converts a **MiniML** function that projects the first element of a pair of integers to **Affi** and applies it to **(true, false)**, producing **true** successfully. By contrast, $P1^\dagger$ tries to use the pair twice (sites of errors are highlighted), which once converted to **Affi**, is a violation of the type invariant, and thus this produces a runtime error, which we can see in the compiled code will occur at the second invocation of `x()`, which contains the contents of a `once()`.

Program $P2$ defines an affine function (and immediately applies it) that binds a variable a in Affi , then uses it (inside a closure) in MiniML , returning a pair made up of that variable and a value from MiniML . This works fine, evaluating to $(\text{true}, \text{true})$. Program $P2^\dagger$ attempts to do an analogous thing, but uses the variable twice, which is a violation of the affine type and thus results in a runtime failure. We can see that in the invocations of $a()$, which contain $\text{once}(0)$.

Note that we do not allow a dynamic function `λa0:...e` to close over static resources, as it may be duplicated if passed to `MinиML`, and thus the static resources would be unprotected. However, we do allow a dynamic function

$()$	$\rightsquigarrow ()$
n	$\rightsquigarrow n$
$true$	$\rightsquigarrow 0$
$false$	$\rightsquigarrow 1$
x	$\rightsquigarrow x$
a	$\rightsquigarrow a()$
$\lambda a : \tau.e$	$\rightsquigarrow \lambda a.\{e^+\}$
$e_1 e_2$	$\rightsquigarrow e_1^+ \text{ (let } x = e_2^+ \text{ in once}(x))$
(e_1, e_2)	$\rightsquigarrow (e_1^+, e_2^+)$
$e.1$	$\rightsquigarrow (\text{fst } e^+)()$
$e.2$	$\rightsquigarrow (\text{snd } e^+)()$
$\langle e_1, e_2 \rangle$	$\rightsquigarrow (\lambda_{-}\{e_1^+\}, \lambda_{-}\{e_2^+\})$
$\text{let } (a_1, a_2) = e_1$	$\rightsquigarrow \text{let } x_{\text{fresh}} = e_1^+, x'_{\text{fresh}} = \text{fst } x_{\text{fresh}},$
$\text{in } e_2$	$x''_{\text{fresh}} = \text{snd } x_{\text{fresh}}, a_1 = \text{once}(x'_{\text{fresh}}),$
	$a_2 = \text{once}(x''_{\text{fresh}}) \text{ in } e_2^+$
$!e$	$\rightsquigarrow e^+$
$\text{let } !x = e_1$	$\rightsquigarrow \text{let } x = e_1^+ \text{ in } e_2^+$
$\text{in } e_2$	
$\langle e \rangle_\tau$	$\rightsquigarrow C_{\tau \mapsto \tau}(e^+)$

$()$	$\rightsquigarrow ()$
$true$	$\rightsquigarrow 0$
$false$	$\rightsquigarrow 1$
x	$\rightsquigarrow x$
a_\circ	$\rightsquigarrow a()$
a_\bullet	$\rightsquigarrow a_\bullet$
$\lambda a_\circ : \tau.e$	$\rightsquigarrow \lambda a_\circ.\{e^+\}$
$\lambda a_\bullet : \tau.e$	$\rightsquigarrow \lambda a_\bullet.\{e^+\}$
$(e_1 : \tau_1 \multimap \tau_2) e_2$	$\rightsquigarrow e_1^+ \text{ (let } x = e_2^+ \text{ in once}(x))$
$(e_1 : \tau_1 \bullet \tau_2) e_2$	$\rightsquigarrow e_1^+ e_2^+$
$!v$	$\rightsquigarrow v^+$
$\text{let } !x = e \text{ in } e'$	$\rightsquigarrow \text{let } x = e^+ \text{ in } e'^+$
$\langle e, e' \rangle$	$\rightsquigarrow (\lambda_{-}\{e^+\}, \lambda_{-}\{e'^+\})$
$e.1$	$\rightsquigarrow (\text{fst } e^+)()$
$e.2$	$\rightsquigarrow (\text{snd } e^+)()$
$\langle e, e' \rangle$	$\rightsquigarrow (e^+, e'^+)$
$\text{let } (a_\bullet, a'_\bullet) = e \text{ in } e'$	$\rightsquigarrow \text{let } x_{\text{fresh}} = e^+ \text{ in let } a_\bullet = \text{fst } x_{\text{fresh}} \text{ in let } a'_\bullet = \text{snd } x_{\text{fresh}} \text{ in } e'^+$
$\langle e : \tau \rangle_\tau$	$\rightsquigarrow C_{\tau \mapsto \tau}(e^+)$

$\text{once}(e) \triangleq \text{let } r_{\text{fresh}} = \text{ref UNUSED} \text{ in } \lambda_{-}\{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := \text{USED}; e\}\}$
 where USED = 0 and UNUSED = 1

Figure 5.7: Compilers for **Affi** (static and dynamic).

$C_{\text{unit} \leftrightarrow \text{unit}}, C_{\text{unit} \leftrightarrow \text{unit}} : \text{unit} \sim \text{unit}$	
	$\overline{C_{\text{int} \leftrightarrow \text{bool}}, C_{\text{bool} \leftrightarrow \text{int}} : \text{int} \sim \text{bool}}$
$C_{\tau_1 \leftrightarrow \tau_1}, C_{\tau_1 \leftrightarrow \tau_1} : \tau_1 \sim \tau_1$	$C_{\tau_2 \leftrightarrow \tau_2}, C_{\tau_2 \leftrightarrow \tau_2} : \tau_2 \sim \tau_2$
$C_{\tau_1 \otimes \tau_2 \leftrightarrow \tau_1 \times \tau_2}, C_{\tau_1 \times \tau_2 \leftrightarrow \tau_1 \otimes \tau_2} : \tau_1 \otimes \tau_2 \sim \tau_1 \times \tau_2$	
$C_{\tau_1 \leftrightarrow \tau_1}, C_{\tau_1 \leftrightarrow \tau_1} : \tau_1 \sim \tau_1$	$C_{\tau_2 \leftrightarrow \tau_2}, C_{\tau_2 \leftrightarrow \tau_2} : \tau_2 \sim \tau_2$
$C_{\tau_1 \multimap \tau_2 \leftrightarrow (\text{unit} \rightarrow \tau_1) \rightarrow \tau_2}, C_{(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2 \leftrightarrow \tau_1 \multimap \tau_2} : \tau_1 \multimap \tau_2 \sim (\text{unit} \rightarrow \tau_1) \rightarrow \tau_2$	
$C_{\text{unit} \leftrightarrow \text{unit}}(e) \triangleq C_{\text{int} \leftrightarrow \text{bool}}(e) \triangleq C_{\text{unit} \leftrightarrow \text{unit}}(e) \triangleq e$	$C_{\text{bool} \leftrightarrow \text{int}}(e) \triangleq \text{if } e \text{ 0 1}$
$C_{\tau_1 \otimes \tau_2 \leftrightarrow \tau_1 \times \tau_2}(e) \triangleq \text{let } x = e \text{ in } (C_{\tau_1 \leftrightarrow \tau_1}(\text{fst } x), C_{\tau_2 \leftrightarrow \tau_2}(\text{snd } x))$	
$C_{\tau_1 \times \tau_2 \leftrightarrow \tau_1 \otimes \tau_2}(e) \triangleq \text{let } x = e \text{ in } (C_{\tau_1 \leftrightarrow \tau_1}(\text{fst } x), C_{\tau_2 \leftrightarrow \tau_2}(\text{snd } x))$	
$C_{\tau_1 \multimap \tau_2 \leftrightarrow (\text{unit} \rightarrow \tau_1) \rightarrow \tau_2}(e) \triangleq \text{let } x = e \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{conv}} = C_{\tau_1 \leftrightarrow \tau_1}(x_{\text{thnk}}())$	$\text{in let } x_{\text{acc}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \leftrightarrow \tau_2}(x x_{\text{acc}})$
$C_{(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2 \leftrightarrow \tau_1 \multimap \tau_2}(e) \triangleq \text{let } x = e \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{acc}} = \text{once}(C_{\tau_1 \leftrightarrow \tau_1}(x_{\text{thnk}}()))$	$\text{in } C_{\tau_2 \leftrightarrow \tau_2}(x x_{\text{acc}})$

Figure 5.8: Convertibility for **MiniML** and **Affi**.

$$\begin{aligned}
 P1 &= ((\lambda x : (\text{unit} \rightarrow \text{int} \times \text{int}).\text{fst } (x \text{ })) \text{)} \text{bool} \otimes \text{bool} \multimap \text{bool} \quad (\text{true}, \text{false}) \\
 P1^\dagger &= ((\lambda x : (\text{unit} \rightarrow \text{int} \times \text{int}).(\text{fst } (x \text{ }), \text{snd } (x \text{ })) \text{)} \text{bool} \otimes \text{bool} \multimap \text{bool} \otimes \text{bool} \\
 &\quad (\text{true}, \text{false}) \\
 P2 &= (\lambda a : \text{bool}.((\lambda y : \text{int}.(a \text{ int}, y)) \text{ 0}) \text{bool} \otimes \text{bool}) \text{ true} \\
 P2^\dagger &= (\lambda a : \text{bool}.((\lambda y : \text{int}.(a \text{ int}, (a \text{ }) \text{ int})) \text{ 0}) \text{bool} \otimes \text{bool}) \text{ true}
 \end{aligned}$$

```

compile*(P1)    =
  ( $\lambda x_{\text{thnk}}^1.$ ( $\lambda x.$ {fst (x ())}))  

    once(let x2 = (x1thnk ()) in (if (fst x2) 0 1, if (snd x2) 0 1))  

  (once((0,1)))  

compile*(P1†)  =
  ( $\lambda x_{\text{thnk}}^1.$ (let x3 = ( $\lambda x.$ {(fst (x ())), snd( x () )})  

    (once(let x2 = (x1thnk ()) in (if (fst x2) 0 1, if (snd x2) 0 1)))  

    in (fst x3, snd x3))  

  (once((0,1)))  

compile*(P2)    =
  ( $\lambda a.$ (let x2 = (( $\lambda y.$ (if (a ()) 0 1, y)) 0) in (fst x2, snd x2)))  

  (let x1 = 0 in once(x1))  

compile*(P2†) =
  ( $\lambda a.$ (let x2 = (( $\lambda y.$ (if (a ()) 0 1, if ( a () ) 0 1)) 0) in (fst x2, snd x2)))  

  (let x1 = 0 in once(x1))

```

Note: `compile*(·)` performs basic simplifications after compiling.

Figure 5.9: Examples of interoperability for MiniML and **Affi**, with compilations.

$$\begin{aligned}
Atom_n &\triangleq \{(W, e_1, e_2) \mid W \in World_n\} & AtomVal_n &\triangleq \{(W, v_1, v_2) \in Atom_n\} \\
AtomVal &\triangleq \bigcup_n AtomVal_n \\
World_n &\triangleq \{(k, \Psi, \Theta) \mid k < n \wedge \Psi \subset HeapTy_k \wedge \text{dom}(\Psi) \# \text{dom}(\Theta)\} \\
HeapTy_n &\triangleq \{(\ell_1, \ell_2) \mapsto Typ_n, \dots\} \\
Typ_n &\triangleq \{R \in 2^{AtomVal_n} \mid \forall (W, v_1, v_2) \in R. \forall W'. W \sqsubseteq W' \implies (W', v_1, v_2) \in R\} \\
Typ &\triangleq \{R \in 2^{AtomVal} \mid \forall k. [R]_k \in Typ_k\} \\
\Theta &\triangleq \{(\ell_1, \ell_2) \mapsto \{\text{USED}, \text{UNUSED}\}, \dots\} \text{ where USED} = 0 \text{ and UNUSED} = 1 \\
W \sqsubseteq W' &\triangleq W'.k \leq W.k \wedge \forall (\ell_1, \ell_2) \in \text{dom}(W.\Psi). [W.\Psi(\ell_1, \ell_2)]_{W'.k} = W'.\Psi(\ell_1, \ell_2) \\
&\wedge \forall (\ell_1, \ell_2) \in \text{dom}(W.\Theta). (\ell_1, \ell_2) \in \text{dom}(W'.\Theta) \\
&\wedge (W.\Theta(\ell_1, \ell_2) = \text{USED} \implies W'.\Theta(\ell_1, \ell_2) = \text{USED}) \\
H_1, H_2 : W &\triangleq (\forall (\ell_1, \ell_2) \mapsto R \in W.\Psi. (\triangleright W, H_1(\ell_1), H_2(\ell_2)) \in R) \\
&\wedge (\forall (\ell_1, \ell_2) \mapsto b \in W.\Theta. H_1(\ell_1) = H_2(\ell_2) = b)
\end{aligned}$$

Figure 5.10: Supporting definitions for logical relation for **MiniML** and **Affi**.

to accept a static closure as argument. This is safe because the dynamic guards will ensure that the static closure is called at most once. Once called, any static resources in its body will be used safely because the static closure typechecked.

SEMANTIC MODEL FOR **Affi (DYNAMIC)** Each variant has a separate model, as what we need to track in our semantics varies significantly. We present the simpler case of **Affi** first. To reason about this system, we create a step indexed binary logical relation, shown in Figures 5.10 and 5.11. Although there is some overlap with the previous case study and prior work, the treatment of affine resources is novel. In our worlds W , we keep the step index, a standard heap typing Ψ , and a novel affine flag store Θ . Following prior work, Ψ maintains a simple bijection between locations of the two programs (this doesn't support sophisticated reasoning about equivalence in the presence of "local state" (Ahmed et al., 2009) but suffices for soundness), whereas Θ maintains a bijection on flags (locations) that track whether affine variables have been used. The latter is a subset of the heap, disjoint from Ψ , and restricted by the model to only contain either 0 or 1, which for convenience we write using the constants **USED** and **UNUSED**. Our world extension relation, $W \sqsubseteq W'$, shows that flags cannot be removed from Θ , and once a flag is marked as **USED**, it cannot be marked **UNUSED**. With this setup, our expression relation $\mathcal{E}[\tau]_\rho$ is quite ordinary, as the described structure is entirely about characterizing the heaps that programs will run in, not about how they will run. Note that $\mathcal{E}[\tau]_\rho$ allows for the possibility of e_1 raising a conversion error (fail CONV) at runtime.

Most cases of the value relation are standard, though of the realizability flavor. Unlike the previous case study, this is a binary relation, which means that it is composed of triples of worlds W and pairs of values or

$$\begin{aligned}
\mathcal{V}[\![\text{unit}]\!]_\rho &= \{(W, (), ())\} \\
\mathcal{V}[\![\text{int}]\!]_\rho &= \{(W, n, n) \mid n \in \mathbb{Z}\} \\
\mathcal{V}[\![\tau_1 \times \tau_2]\!]_\rho &= \{(W, (v_{1a}, v_{2a}), (v_{1b}, v_{2b})) \mid (W, v_{1a}, v_{1b}) \in \mathcal{V}[\![\tau_1]\!]_\rho \\
&\quad \wedge (W, v_{2a}, v_{2b}) \in \mathcal{V}[\![\tau_2]\!]_\rho\} \\
\mathcal{V}[\![\tau_1 + \tau_2]\!]_\rho &= \{(W, \text{inl } v_1, \text{inl } v_2) \mid (W, v_1, v_2) \in \mathcal{V}[\![\tau_1]\!]_\rho\} \\
&\quad \cup \{(W, \text{inr } v_1, \text{inr } v_2) \mid (W, v_1, v_2) \in \mathcal{V}[\![\tau_2]\!]_\rho\} \\
\mathcal{V}[\![\tau_1 \rightarrow \tau_2]\!]_\rho &= \{(W, \lambda x. \{e_1\}, \lambda x. \{e_2\}) \mid \forall v_1 v_2. W \sqsubset W' \\
&\quad \wedge (W', v_1, v_2) \in \mathcal{V}[\![\tau_1]\!]_\rho \\
&\quad \implies (W', [x \mapsto v_1]e_1, [x \mapsto v_2]e_2) \in \mathcal{E}[\![\tau_2]\!]_\rho\} \\
\mathcal{V}[\![\text{ref } \tau]\!]_\rho &= \{(W, \ell_1, \ell_2) \mid W.\Psi(\ell_1, \ell_2) = [\mathcal{V}[\![\tau]\!]_\rho]_{W.k}\} \\
\mathcal{V}[\![\forall \alpha. \tau]\!]_\rho &= \{(W, \lambda_- e_1, \lambda_- e_2) \mid \forall R \in \text{Typ}, W'. W \sqsubset W' \\
&\quad \implies (W', e_1, e_2) \in \mathcal{E}[\![\tau]\!]_{\rho[\alpha \mapsto R]}\} \\
\mathcal{V}[\![\alpha]\!]_\rho &= \rho(\alpha) \\
\mathcal{V}[\![\text{unit}]\!]. &= \{(W, (), ())\} \\
\mathcal{V}[\![\text{bool}]\!]_\rho &= \{(W, 0, 0)\} \cup \{(W, n_1, n_2) \mid n_1 \neq 0 \wedge n_2 \neq 0\} \\
\mathcal{V}[\![\text{int}]\!]. &= \{(W, n, n) \mid n \in \mathbb{Z}\} \\
\mathcal{V}[\![\tau_1 \multimap \tau_2]\!]. &= \{(W, \lambda a.e_1, \lambda a.e_2) \mid \forall v_1 v_2. W' \ell_1 \ell_2. \\
&\quad W \sqsubset W' \wedge (W', v_1, v_2) \in \mathcal{V}[\![\tau_1]\!] \\
&\quad \wedge (\ell_1, \ell_2) \notin \text{dom}(W'.\Psi) \cup \text{dom}(W'.\Theta) \\
&\quad \implies ((W'.k, W'.\Psi, W'.\Theta \uplus (\ell_1, \ell_2) \mapsto \text{UNUSED}), \\
&\quad [a \mapsto \text{guard}(v_1, \ell_1)]e_1, [a \mapsto \text{guard}(v_2, \ell_2)]e_2) \in \mathcal{E}[\![\tau_2]\!]\} \\
\mathcal{V}[\![\text{!}\tau]\!]. &= \{(W, v_1, v_2) \mid (W, v_1, v_2) \in \mathcal{V}[\![\tau]\!]\} \\
\mathcal{V}[\![\tau_1 \otimes \tau_2]\!]. &= \{(W, (v_{1a}, v_{2a}), (v_{1b}, v_{2b})) \mid (W, v_{1a}, v_{1b}) \in \mathcal{V}[\![\tau_1]\!] \\
&\quad \wedge (W, v_{2a}, v_{2b}) \in \mathcal{V}[\![\tau_2]\!]\} \\
\mathcal{V}[\![\tau_1 \& \tau_2]\!]. &= \{(W, (\lambda_- \{e_{1a}\}, \lambda_- \{e_{2a}\}), (\lambda_- \{e_{1b}\}, \lambda_- \{e_{2b}\})) \\
&\quad \mid (W, e_{1a}, e_{1b}) \in \mathcal{E}[\![\tau_1]\!]. \wedge (W, e_{2a}, e_{2b}) \in \mathcal{E}[\![\tau_2]\!]\} \\
\\
\text{guard}(e, \ell) &\triangleq \lambda_- \{\text{if } !\ell \{ \text{fail CONV} \} \{ \ell := \text{USED}; e \}\} \\
\mathcal{E}[\![\tau]\!]_\rho &= \{(W, e_1, e_2) \mid \text{freevars}(e_1) = \text{freevars}(e_2) = \emptyset \wedge \\
&\quad \forall H_1, H_2 : W, e'_1, H'_1, j < W.k. \langle H_1, e_1 \rangle \xrightarrow{j} \langle H'_1, e'_1 \rangle \rightsquigarrow \\
&\quad \implies e'_1 = \text{fail CONV} \vee (\exists v_2 H'_2 W'. \langle H_2, e_2 \rangle \xrightarrow{*} \langle H'_2, v_2 \rangle \\
&\quad \wedge W \sqsubseteq W' \wedge H'_1, H'_2 : W' \wedge (W', e'_1, v_2) \in \mathcal{V}[\![\tau]\!]_\rho)\} \\
\\
\mathcal{G}[\![\cdot]\!]_\rho &= \{(W, \cdot) \mid W \in \text{World}\} \\
\mathcal{G}[\![\Gamma, x : \tau]\!]_\rho &= \{(W, \gamma; x \mapsto (v_1, v_2)) \mid (W, v_1, v_2) \in \mathcal{V}[\![\tau]\!]_\rho \wedge (W, \gamma) \in \mathcal{G}[\![\Gamma]\!]_\rho\} \\
\mathcal{G}[\![\Gamma, x : \tau]\!]_\rho &= \{(W, \gamma; x \mapsto (v_1, v_2)) \mid (W, v_1, v_2) \in \mathcal{V}[\![\tau]\!]_\rho \wedge (W, \gamma) \in \mathcal{G}[\![\Gamma]\!]_\rho\} \\
\mathcal{G}[\![\Omega, a : \tau]\!]_\rho &= \{(W, \gamma; a \mapsto (\text{guard}(v_1, \ell_1), \text{guard}(v_2, \ell_2))) \mid \\
&\quad (\ell_1, \ell_2) \in W.\Theta \wedge (W, v_1, v_2) \in \mathcal{V}[\![\tau]\!]_\rho \wedge (W, \gamma) \in \mathcal{G}[\![\Omega]\!]_\rho \\
&\quad \wedge \ell_1 \notin FL(v_1) \cup FL(\text{cod}(\gamma^1)) \wedge \ell_2 \notin FL(v_2) \cup FL(\text{cod}(\gamma^2))\} \\
\\
\gamma^1 &\triangleq \{\ell \rightarrow v_1 \mid \ell \rightarrow (v_1, v_2) \in \gamma\} \quad \gamma^2 \triangleq \{\ell \rightarrow v_2 \mid \ell \rightarrow (v_1, v_2) \in \gamma\} \\
\mathcal{D}[\![\cdot]\!] &= \{\cdot\} \quad \mathcal{D}[\![\Delta, \alpha]\!] = \{\rho[\alpha \mapsto R] \mid R \in \text{Typ} \wedge \rho \in \mathcal{D}[\![\Delta]\!]\} \\
\Gamma; \Omega; \Delta; \Gamma \vdash e_1 \preceq e_2 : \tau &\equiv \forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Delta \\
\rho \in \mathcal{D}[\![\Delta]\!] \wedge (W, \gamma_\Gamma) &\in \mathcal{G}[\![\Gamma]\!]_\rho \wedge (W, \gamma_\Omega) \in \mathcal{G}[\![\Omega]\!]. \wedge (W, \gamma_\Delta) \in \mathcal{G}[\![\Delta]\!] \\
\implies (W, \text{close}_1(\gamma_\Gamma, \text{close}_1(\gamma_\Omega, \text{close}_1(\gamma_\Delta, e_1^+))), \\
&\quad \text{close}_2(\gamma_\Gamma, \text{close}_2(\gamma_\Omega, \text{close}_2(\gamma_\Delta, e_2^+)))) \in \mathcal{E}[\![\tau]\!]_\rho \\
\\
\Gamma; \Omega; \Delta; \Gamma \vdash e_1 \preceq e_2 : \tau &\equiv \forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Delta \\
\rho \in \mathcal{D}[\![\Delta]\!] \wedge (W, \gamma_\Gamma) &\in \mathcal{G}[\![\Gamma]\!]_\rho \wedge (W, \gamma_\Omega) \in \mathcal{G}[\![\Omega]\!]. \wedge (W, \gamma_\Delta) \in \mathcal{G}[\![\Delta]\!] \\
\implies (W, \text{close}_1(\gamma_\Gamma, \text{close}_1(\gamma_\Omega, \text{close}_1(\gamma_\Delta, e_1^+))), \\
&\quad \text{close}_2(\gamma_\Gamma, \text{close}_2(\gamma_\Omega, \text{close}_2(\gamma_\Delta, e_2^+)))) \in \mathcal{E}[\![\tau]\!]_\rho
\end{aligned}$$

Figure 5.11: Logical Relation for **MiniML** and **Affi**.

terms. Note the step indexing stratification, with $World_n$ built out of heap typings for smaller step index, and $HeapTy_n$ composed of relations on world, value, value triples for world at the given index. As is standard, Typ_n is made up of relations that are closed under world extension. The value relations themselves mostly follow the same pattern as the previous case study, though **MinimL** has polymorphism, which we interpret in the standard way using relational substitutions: $\forall \alpha.\tau$ is inhabited by thunks which are in the relation extended with an arbitrary relation R , and α is exactly that relation.

The only true novelty in the relation is the affine arrow \multimap case. A pair of functions $\lambda\alpha.e_1$ and $\lambda\alpha.e_2$ are related if, given a pair of arguments v_1 and v_2 related at a future world W' , we get related results in W' extended with a new entry in the flag store $W'.\Theta$ for some fresh locations ℓ_1, ℓ_2 . Importantly, what we substitute into the body is *not* v_1 and v_2 , but rather wrapped forms, $\text{guard}(v_1, \ell_1)$ and $\text{guard}(v_2, \ell_2)$, each of which closes over the fresh location in the flag store and thus ensures that the argument is not used more than once. This makes sense, since in the target, our calling convention is that affine variables should be thunks, and will be forced upon use.

With the logical relation in hand, we prove semantic soundness in the standard way, first establishing convertibility soundness and compatibility lemmas for all of our typing rules. We provide the lemma statements here; the proofs, which are quite mechanical, are provided in Appendix B.

Theorem 5.0.1 (Convertibility Soundness).

$$\begin{aligned} \text{If } \tau \sim \sigma \text{ then } \forall (W, e_1, e_2) \in \mathcal{E}[\tau]. &\implies (W, C_{\tau \mapsto \sigma}(e_1), C_{\tau \mapsto \sigma}(e_2)) \in \mathcal{E}[\sigma]. \\ \forall (W, e_1, e_2) \in \mathcal{E}[\sigma]. &\implies (W, C_{\sigma \mapsto \tau}(e_1), C_{\sigma \mapsto \tau}(e_2)) \in \mathcal{E}[\tau]. \end{aligned}$$

Proof. See B.1.13. □

Lemma 5.0.2 (Compat **unit**).

$$\Gamma; \Omega; \Delta; \Gamma \vdash () \preceq () : \text{unit}$$

Proof. See B.1.14. □

Lemma 5.0.3 (Compat **int**).

$$\Gamma; \Omega; \Delta; \Gamma \vdash \mathbb{Z} \preceq \mathbb{Z} : \text{int}$$

Proof. See B.1.15. □

Lemma 5.0.4 (Compat **x**).

$$\Delta \vdash \tau \wedge x : \tau \in \Gamma \implies \Gamma; \Omega; \Delta; \Gamma \vdash x \preceq x : \tau$$

Proof. See B.1.16. □

Lemma 5.0.5 (Compat \times).

$$\begin{aligned} \Gamma_1; \Omega_1; \Delta; \Gamma \vdash e_1 \preceq e_1 : \tau_1 \wedge \Gamma_2; \Omega_2; \Delta; \Gamma \vdash e_2 \preceq e_2 : \tau_2 \\ \implies \Gamma_1; \Omega_1 \uplus \Omega_2; \Delta; \Gamma \vdash (e_1, e_2) \preceq (e_1, e_2) : \tau_1 \times \tau_2 \end{aligned}$$

Proof. See B.1.17. \square

Lemma 5.0.6 (Compat fst).

$$\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau_1 \times \tau_2 \implies \Gamma; \Omega; \Delta; \Gamma \vdash \text{fst } e \preceq \text{fst } e : \tau_1$$

Proof. See B.1.18. \square

Lemma 5.0.7 (Compat snd).

$$\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau_1 \times \tau_2 \implies \Gamma; \Omega; \Delta; \Gamma \vdash \text{snd } e \preceq \text{snd } e : \tau_2$$

Proof. See B.1.19. \square

Lemma 5.0.8 (Compat inl).

$$\Delta \vdash \tau_2 \wedge \Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau_1 \implies \Gamma; \Omega; \Delta; \Gamma \vdash \text{inl } e \preceq \text{inl } e : \tau_1 + \tau_2$$

Proof. See B.1.20. \square

Lemma 5.0.9 (Compat inr).

$$\Delta \vdash \tau_1 \wedge \Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau_2 \implies \Gamma; \Omega; \Delta; \Gamma \vdash \text{inr } e \preceq \text{inr } e : \tau_1 + \tau_2$$

Proof. See B.1.21. \square

Lemma 5.0.10 (Compat match).

$$\begin{aligned} & \Gamma_1; \Omega_1; \Delta; \Gamma \vdash e \preceq e : \tau_1 + \tau_2 \\ & \wedge \Gamma_2; \Omega_2; \Delta; \Gamma[x : \tau_1] \vdash e_1 \preceq e_1 : \tau \\ & \wedge \Gamma_2; \Omega_2; \Delta; \Gamma[y : \tau_2] \vdash e_2 \preceq e_2 : \tau \\ \implies & \Gamma_1; \Omega_1 \uplus \Omega_2; \Delta; \Gamma \vdash \text{match } e \text{ x}\{e_1\} \text{ y}\{e_2\} \preceq \text{match } e \text{ x}\{e_1\} \text{ y}\{e_2\} : \tau \end{aligned}$$

Proof. See B.1.22. \square

Lemma 5.0.11 (Compat \rightarrow).

$$\Gamma; \Omega; \Delta; \Gamma[x : \tau_1] \vdash e \preceq e : \tau_2 \implies \Gamma; \Omega; \Delta; \Gamma \vdash \lambda x : \tau_1. e \preceq \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2$$

Proof. See B.1.23. \square

Lemma 5.0.12 (Compat `app`).

$$\begin{aligned} \Gamma_1; \Omega_1; \Delta; \Gamma \vdash e_1 \preceq e_1 : \tau_1 \rightarrow \tau_2 \wedge \Gamma_2; \Omega_2; \Delta; \Gamma \vdash e_2 \preceq e_2 : \tau_1 \implies \\ \Gamma_1; \Omega_1 \uplus \Omega_2; \Delta; \Gamma \vdash e_1 e_2 \preceq e_1 e_2 : \tau_2 \end{aligned}$$

Proof. See B.1.24. \square

Lemma 5.0.13 (Compat `V`).

$$\Gamma; \Omega; \Delta, \alpha; \Gamma \vdash e \preceq e : \tau \implies \Gamma; \Omega; \Delta; \Gamma \vdash \Lambda \alpha. e \preceq \Lambda \alpha. e : \forall \alpha. \tau$$

Proof. See B.1.25. \square

Lemma 5.0.14 (Compat $[\tau/\alpha]$).

$$\Delta \vdash \tau' \wedge \Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \forall \alpha. \tau \implies \Gamma; \Omega; \Delta; \Gamma \vdash e[\tau'] \preceq e[\tau'] : \tau[\tau'/\alpha]$$

Proof. See B.1.26. \square

Lemma 5.0.15 (Compat `ref`).

$$\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau \implies \Gamma; \Omega; \Delta; \Gamma \vdash \text{ref } e \preceq \text{ref } e : \text{ref } \tau$$

Proof. See B.1.27. \square

Lemma 5.0.16 (Compat `!`).

$$\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \text{ref } \tau \implies \Gamma; \Omega; \Delta; \Gamma \vdash !e \preceq !e : \tau$$

Proof. See B.1.28. \square

Lemma 5.0.17 (Compat `:=`).

$$\begin{aligned} \Gamma_1; \Omega_1; \Delta; \Gamma \vdash e_1 \preceq e_1 : \text{ref } \tau \wedge \Gamma_2; \Omega_2; \Delta; \Gamma \vdash e_2 \preceq e_2 : \tau \\ \implies \Gamma_1; \Omega_1 \uplus \Omega_2; \Delta; \Gamma \vdash e_1 := e_2 \preceq e_1 := e_2 : \text{unit} \end{aligned}$$

Proof. See B.1.29. \square

Lemma 5.0.18 (Compat $(\text{e})_\tau$).

$$\Delta; \Gamma; \Omega \vdash e \preceq e : \tau \wedge \tau \sim \tau \implies \Gamma; \Omega; \Delta; \Gamma \vdash (\text{e})_\tau \preceq (\text{e})_\tau : \tau \wedge _ : \tau \sim \tau$$

Proof. See B.1.30. \square

Lemma 5.0.19 (Compat `unit`).

$$\Delta; \Gamma; \Omega \vdash () \preceq () : \text{unit}$$

Proof. See B.1.31. \square

Lemma 5.0.20 (Compat **true**).

$$\Delta; \Gamma; \Gamma; \Omega \vdash \text{true} \preceq \text{true} : \text{bool}$$

Proof. See B.1.32. □

Lemma 5.0.21 (Compat **false**).

$$\Delta; \Gamma; \Gamma; \Omega \vdash \text{false} \preceq \text{false} : \text{bool}$$

Proof. See B.1.33. □

Lemma 5.0.22 (Compat **int**).

$$\Delta; \Gamma; \Gamma; \Omega \vdash n \preceq n : \text{int}$$

Proof. See B.1.34. □

Lemma 5.0.23 (Compat **a**).

$$a : \tau \in \Omega \implies \Delta; \Gamma; \Gamma; \Omega \vdash a \preceq a : \tau$$

Proof. See B.1.35. □

Lemma 5.0.24 (Compat **x**).

$$x : \tau \in \Gamma \implies \Delta; \Gamma; \Gamma; \Omega \vdash x \preceq x : \tau$$

Proof. See B.1.36. □

Lemma 5.0.25 (Compat **→**).

$$\Delta; \Gamma; \Gamma; \Omega, a : \tau_1 \vdash e \preceq e : \tau_2 \implies \Delta; \Gamma; \Gamma; \Omega \vdash \lambda a : \tau_1. e \preceq \lambda a : \tau_1. e : \tau_1 \multimap \tau_2$$

Proof. See B.1.37. □

Lemma 5.0.26 (Compat **app**).

$$\begin{aligned} & \Delta_1; \Gamma_1; \Gamma; \Omega_1 \vdash e_1 \preceq e_1 : \tau_1 \multimap \tau_2 \wedge \Delta_2; \Gamma_2; \Gamma; \Omega_2 \vdash e_2 \preceq e_2 : \tau_1 \\ & \implies \Delta_1; \Gamma_1; \Gamma; \Omega_1 \uplus \Omega_2 \vdash e_1 e_2 \preceq e_1 e_2 : \tau_2 \end{aligned}$$

Proof. See B.1.38. □

Lemma 5.0.27 (Compat **!**).

$$\Delta; \Gamma; \Gamma; \cdot \vdash v \preceq v : \tau \implies \Delta; \Gamma; \Gamma; \cdot \vdash !v \preceq !v : !\tau$$

Proof. See B.1.39. □

Lemma 5.0.28 (Compat let!).

$$\begin{aligned} & \Delta; \Gamma; \Omega; \Omega_1 \vdash e_1 \preceq e_1 : !\tau \wedge \Delta; \Gamma; \Omega; \Omega_2 \vdash e_2 \preceq e_2 : \tau' \\ \implies & \Delta; \Gamma; \Omega; \Omega_1 \uplus \Omega_2 \vdash \text{let } !x = e_1 \text{ in } e_2 \preceq \text{let } !x = e_1 \text{ in } e_2 : \tau' \end{aligned}$$

Proof. See B.1.40. \square

Lemma 5.0.29 (Compat $\&$).

$$\begin{aligned} & \Delta; \Gamma; \Omega; \Omega \vdash e_1 \preceq e_1 : \tau_1 \wedge \Delta; \Gamma; \Omega; \Omega \vdash e_2 \preceq e_2 : \tau_2 \\ \implies & \Delta; \Gamma; \Omega; \Omega \vdash \langle e_1, e_2 \rangle \preceq \langle e_1, e_2 \rangle : \tau_1 \& \tau_2 \end{aligned}$$

Proof. See B.1.41. \square

Lemma 5.0.30 (Compat $.1$).

$$\Delta; \Gamma; \Omega; \Omega \vdash e \preceq e : \tau_1 \& \tau_2 \implies \Delta; \Gamma; \Omega; \Omega \vdash e.1 \preceq e.1 : \tau_1$$

Proof. See B.1.42. \square

Lemma 5.0.31 (Compat $.2$).

$$\Delta; \Gamma; \Omega; \Omega \vdash e \preceq e : \tau_1 \& \tau_2 \implies \Delta; \Gamma; \Omega; \Omega \vdash e.2 \preceq e.2 : \tau_2$$

Proof. See B.1.43. \square

Lemma 5.0.32 (Compat \otimes).

$$\begin{aligned} & \Delta; \Gamma; \Omega; \Omega_1 \vdash e_1 \preceq e_1 : \tau_1 \wedge \Delta; \Gamma; \Omega; \Omega_2 \vdash e_2 \preceq e_2 : \tau_2 \\ \implies & \Delta; \Gamma; \Omega; \Omega_1 \uplus \Omega_2 \vdash (e_1, e_2) \preceq (e_1, e_2) : \tau_1 \otimes \tau_2 \end{aligned}$$

Proof. See B.1.44. \square

Lemma 5.0.33 (Compat let).

$$\begin{aligned} & \Delta; \Gamma; \Omega; \Omega_1 \vdash e_1 \preceq e_1 : \tau_1 \otimes \tau_2 \wedge \Delta; \Gamma; \Omega_2, a : \tau_1, a' : \tau_2 \vdash e_2 \preceq e_2 : \tau \\ \implies & \Delta; \Gamma; \Omega; \Omega_1 \uplus \Omega_2 \vdash \text{let } (a, a') = e_1 \text{ in } e_2 \preceq \text{let } (a, a') = e_1 \text{ in } e_2 : \tau \end{aligned}$$

Proof. See B.1.45. \square

Lemma 5.0.34 (Compat $(e)_\tau$).

$$\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau \wedge \tau \sim \tau \implies \Delta; \Gamma; \Omega \vdash (e)_\tau \preceq (e)_\tau : \tau$$

Proof. See B.1.46. \square

Theorem 5.0.35 (Fundamental Property).

If $\Gamma; \Omega; \Delta; \Gamma \vdash e : \tau$ then $\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau$ and if $\Delta; \Gamma; \Omega \vdash e : \tau$ then $\Delta; \Gamma; \Omega \vdash e \preceq e : \tau$.

Proof. By induction on typing derivation, relying on the compatibility lemmas, which exist for every typing rule in both source languages. \square

Theorem 5.0.36 (Type Safety for **MinimL**).

For any **MinimL** term e where $\cdot; \cdot; \cdot; \cdot \vdash e : \tau$ and for any heap H , if $\langle H, e^+ \rangle \xrightarrow{*} \langle H', e' \rangle$, then either $e' = \text{fail CONV}$, e' is a value, or there exist H'', e'' such that $\langle H', e' \rangle \rightarrow \langle H'', e'' \rangle$.

Proof. This follows as a consequence of the fundamental property and the definition of the logical relation, as follows: if $\langle H, e^+ \rangle \xrightarrow{n} \langle H', e' \rangle$, then consider a trivial world W with $k > n$, an empty heap typing and empty affine store. Then, since the term is closed, the fundamental property says that $(W, e^+, e^+) \in \mathcal{E}[\tau]$. This means that it runs to a stuck state, which is either at n or greater than n . If it's greater than n , then we have a further step that can be taken. If it gets stuck at n , then we know that is either fail CONV or a value. \square

Theorem 5.0.37 (Type Safety for **Affi**).

For any **Affi** term e where $\cdot; \cdot; \cdot; \cdot \vdash e : \tau$ and for any heap H , if $\langle H, e^+ \rangle \xrightarrow{*} \langle H', e' \rangle$, then we know from the logical relation that either $e' = \text{fail CONV}$, e' is a value, or there exist H'', e'' such that $\langle H', e' \rangle \rightarrow \langle H'', e'' \rangle$.

Proof. This proof is identical to that of **MinimL**. \square

SEMANTIC MODEL FOR **Affi (STATIC)** For our variant with purely static affine functions, **Affi_s**, we have to do significantly more in our model. In particular, we have to show that both the dynamic and static affine bindings within **Affi** are used at most once. For a dynamic binding, this, as in the dynamic variant, is tracked in target code by the dynamic reference flag created by the macro thunk. For a static binding, we use a similar strategy of tracking use via a flag, but rather than a target-level dynamic runtime flag, we create a *phantom* flag that exists only within our model.

Specifically, we define an augmented target operational semantics that exists solely for the model, and any program that runs without getting stuck under the augmented semantics has a trivial erasure to a program that runs under the standard semantics. This means we are using the model to identify a subset of target programs (the erasures of well-behaved augmented programs) that behave sensibly and do not violate source type constraints (i.e., do not use static variables more than once), even if there is nothing in the target programs that actually witnesses those constraints (i.e., dynamic checks or static types).

We build the model as follows. First, we extend our machine configurations to keep track of *phantom flags* f — i.e., in addition to a heap H and term e , we have a *phantom flag set* Φ . Second, the augmented semantics uses

one additional term, `protect`, which consumes one of the aforementioned phantom flags when it reduces:

$$\begin{aligned} \text{Expressions } e ::= \dots & \text{protect}(e, f) \\ \langle \Phi \uplus \{f\}, H, \text{protect}(e, f) \rangle & \dashrightarrow \langle \Phi, H, e \rangle \end{aligned}$$

And finally, we modify the two rules that introduce bindings such that whenever a binding in the syntactic category \bullet is introduced, we create a new phantom flag (where “ f fresh” means f is disjoint from all the flags generated thus far during this execution):

$$\frac{\begin{array}{c} f \text{ fresh} \\ \hline \langle \Phi, H, \text{let } a_\bullet = v \text{ in } e \rangle \dashrightarrow \langle \Phi \uplus \{f\}, H, [a_\bullet \mapsto \text{protect}(v, f)]e \rangle \end{array}}{\begin{array}{c} f \text{ fresh} \\ \hline \langle \Phi, H, \lambda a_\bullet. e v \rangle \dashrightarrow \langle \Phi \uplus \{f\}, H, [a_\bullet \mapsto \text{protect}(v, f)]e \rangle \end{array}}$$

Note that we write \dashrightarrow for a step in this augmented semantics, to distinguish it from the true operational step \rightarrow . While phantom flags in the augmented operational semantics play a similar role in protecting static affine resources as dynamic reference flags in the dynamic case, the critical difference is that in the augmented semantics, a $\text{protect}(\cdot)$ ed resource for which there is no phantom flag will get stuck, and thus be excluded from the logical relation by construction. This is very different from the dynamic case, where we want — and, in fact, need — to include terms that can fail in order to mix **MinimL** and **Affi** without imposing an affine type system on **MinimL** itself. What this means for the model is that dynamic reference flags are a *shared resource* that can be accessed from many parts of the program and therefore tracked in the world, while phantom flags are an *unique resource* which our type system ensures is owned/used by at most one part of the program, which is what allows us to prove that the augmented semantics will not get stuck.

The definitions are shown in Figures 5.12 and 5.13, where much of the structures is similar to the previous variant.

Our expression relation, $\mathcal{E}[\tau]_\rho$, is made up of tuples of worlds W and phantom flag stores / term pairs (Φ_i, e_i) , where each flag store represents the phantom variables owned by the expression. Our worlds W keep the step index, a standard heap typing Ψ (as in Chapter 4), but also an affine flag store Θ , which maps dynamic flags ℓ to either a marker that indicates a dynamic affine variable has been used (0, written USED), or the phantom flags Φ that it closes over if it has not been used (a set that can be empty, of course). These dynamic flags ℓ are a subset of the heap, disjoint from Ψ (which tracks the rest of the heap, i.e., all the normal/non-dynamic-flag references).

The expression relation then says that, given a heap that satisfies the world and arbitrary “rest” of phantom flag store Φ_r (disjoint from that closed over by the world and the owned portion), the term e will either: (i) run longer than the step index accounts for, (ii) fail CONV (error while converting a value), or (iii) terminate at some value e' , where the flag store Φ has been modified to $\Phi_f \uplus \Phi_g$, the heap has changed to H' , and the new world W' is an extension of W . World extension (\sqsubseteq_{Φ_r}) is defined over worlds that do not contain phantom flags from Φ_r , since phantom flags are a local resource and the world contains what is global. It allows the step index to decrease, the heap typing to gain (but not overwrite or remove) entries, and the affine store to mark (but not unmark) dynamic bindings as USED.

At that future world, we know that the resulting value, along with their Φ_f , will be in the value relation $\mathcal{V}[\tau]_\rho$. The phantom flag store Φ_g is “garbage” that is no longer needed, and the “rest” is unchanged. Note that, while running, some phantom flags may have moved into the world, which has changed, but the world cannot have absorbed what was in the “rest”.

Our value relation cases are now mostly standard, so we will focus only on the interesting ones: \multimap and \multimap . $\mathcal{V}[\tau_1 \multimap \tau_2]$. is defined to take an arbitrary argument from $\mathcal{V}[\tau_1]$., which may own static phantom flags in Φ , and add a new location ℓ that will be used in the thunk that prevents multiple uses, but also store the phantom flags in the affine store. The idea is that a function $\lambda a_\circ : _.e$ can be applied to an expression that closes over static phantom flags, like $\text{let } (b_\bullet, c_\bullet) = (1, 2) \text{ in } \lambda a_\bullet.b_\bullet$ —the latter will have phantom flags for both b_\bullet and c_\bullet . The body is then run with the argument substituted with a guarded expression.

Now, consider what happens when the variable is used: the guard(\cdot) wrapper will update the location to USED, which means that in the world, the phantom flags that were put at that location are no longer there — i.e., they are no longer returned by $\text{flags}(W')$, which returns all phantom flags closed over by dynamic flags. That means, for the reduction to be well-formed, the phantom flags have to move somewhere else—either back to being owned by the term (in Φ_f) or in the discarded “garbage” Φ_g . Once the phantom flag set has been moved back out of the world, the flags can again be used by $\text{protect}(\cdot)$ expressions.

The static function, $\mathcal{V}[\tau_1 \multimap \tau_2]$., has a similar flavor, but it may itself own static phantom flags. That means that the phantom flag set for the argument must be disjoint, and when we run the body, we combine the set along with a fresh phantom flags f for the argument, which are then put inside the $\text{protect}(\cdot)$ expressions.

With the logical relation in hand, we take the same approach as before to prove type soundness. Our lemma statements are analogous to the previous section, as the significant details are in the model, not in the static type

$$\begin{aligned}
Atom_n &= \{(W, (\Phi_1, e_1), (\Phi_2, e_2)) \mid W \in World_n \wedge \Phi_1, \Phi_2 : W\} \\
\Phi_1, \Phi_2 : W &\triangleq \forall i \in \{1, 2\}. \Phi_i \# \text{flags}(W, i) \\
AtomVal_n &= \{(W, (\Phi_1, v_1), (\Phi_2, v_2)) \in Atom_n\} \\
AtomVal &= \bigcup_n AtomVal_n \\
World_n &= \{(k, \Psi, \Theta) \mid k < n \wedge \Psi \subset \text{HeapTy}_k \wedge \text{dom}(\Psi) \# \text{dom}(\Theta) \\
&\quad \wedge (\forall (\ell_1, \ell_2) \mapsto (\Phi_1, \Phi_2), (\ell'_1, \ell'_2) \mapsto (\Phi'_1, \Phi'_2) \in \Theta. \\
&\quad (\ell_1, \ell_2) \neq (\ell'_1, \ell'_2) \implies \Phi_1 \cap \Phi'_1 = \Phi_2 \cap \Phi'_2 = \emptyset)\}\} \\
\Theta &= \{(\ell_1, \ell_2) \mapsto \text{USED}\} \cup \{(\ell_1, \ell_2) \mapsto (\Phi_1, \Phi_2)\} \\
\Phi &= \{f\} \quad \text{flags}(W, i) = \bigcup_{(\ell_1, \ell_2) \mapsto (\Phi_1, \Phi_2) \in W. \Theta} \Phi_i \\
Typ_n &= \{R \in 2^{AtomVal_n} \mid \forall (W, (\Phi_1, v_1), (\Phi_2, v_2)) \in R. \forall W'. \\
&\quad W \sqsubseteq_{\Phi_1, \Phi_2} W' \implies (W', (\Phi_1, v_1), (\Phi_2, v_2)) \in R\} \\
Typ &= \{R \in 2^{AtomVal} \mid \forall k. [R]_k \in Typ_k\} \\
UnrTyp &= \{R \in Typ \mid \forall (W, (\Phi_1, v_1), (\Phi_2, v_2)) \in R. \Phi_1 = \Phi_2 = \emptyset\} \\
H_1, H_2 : W &\triangleq (\forall (\ell_1, \ell_2) \mapsto R \in W. \Psi. (\triangleright W, H_1(\ell_1), H_2(\ell_2)) \in R) \\
&\quad \wedge (\forall (\ell_1, \ell_2) \mapsto \text{USED} \in W. \Theta. \forall i \in \{1, 2\}. H_i(\ell_i) = \text{USED}) \\
&\quad \wedge (\forall (\ell_1, \ell_2) \mapsto (\Phi_1, \Phi_2) \in W. \Theta. \forall i \in \{1, 2\}. H_i(\ell_i) = \text{UNUSED}) \\
\text{guard}(e, \ell) &\triangleq \lambda_. \{\text{if } !\ell \{ \text{fail CONV} \} \{ \ell := \text{USED}; e \}\}
\end{aligned}$$

Figure 5.12: **MiniML** & **Aff** Logical Relation Supporting Definitions (static).

system, so we defer both the lemma statements and proofs to Appendix B. With the compatibility lemmas in place, as before we prove Type Safety in the standard two-step way as follows:

Theorem 5.0.38 (Fundamental Property).

If $\Gamma; \Omega; \Delta; \Gamma \vdash e : \tau$ then $\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau$ and if $\Delta; \Gamma; \Omega \vdash e : \tau$ then $\Delta; \Gamma; \Omega \vdash e \preceq e : \tau$.

Proof. By induction on typing derivation, relying on the compatibility lemmas, which exist for every typing rule in both source languages. \square

Theorem 5.0.39 (Type Safety for **MiniML**).

For any **MiniML** term e where $\cdot ; \cdot ; \cdot ; \cdot \vdash e : \tau$ and for any heap H , if $\langle H, e^+ \rangle \xrightarrow{*} \langle H', e' \rangle$, then either $e' = \text{fail CONV}$, e' is a value, or there exist H'', e'' such that $\langle H', e' \rangle \rightarrow \langle H'', e'' \rangle$.

Proof. Suppose $\langle H, e^+ \rangle \xrightarrow{n} \langle H', e' \rangle$ for some natural number n . Either $\langle H', e' \rangle \rightarrow \langle H'', e'' \rangle$, in which case we are done, or $\langle H', e' \rangle$ is irreducible.

Consider a trivial world W that has an arbitrary $k > 2n$, an empty heap typing and an empty affine store. Then, since the term is closed, by the Fundamental Property, $(W, (\emptyset, e^+), (\emptyset, e^+)) \in \mathcal{E}[\![\tau]\!]$. Now by Lemma B.2.20, we know that for any Φ_{r1}, Φ_{r2} , $\langle \Phi_{r1} \uplus \text{flags}(W, 1), H_1, e_1^+ \rangle \xrightarrow{-j} \langle \Phi'_1, H'_1, e'_1 \rangle$ where $j \leq 2n$ and if e'_1 is a value, then $Z(e'_1) = e'$.

$$\begin{aligned}
\mathcal{V}[\![\text{unit}]\!]_\rho &= \{(W, (\emptyset, ()), (\emptyset, ()))\} \\
\mathcal{V}[\![\text{int}]\!]_\rho &= \{(W, (\emptyset, n), (\emptyset, n)) \mid n \in \mathbb{Z}\} \\
\mathcal{V}[\![\tau_1 \times \tau_2]\!]_\rho &= \{(W, (\emptyset, (v_{1a}, v_{2a})), (\emptyset, (v_{1b}, v_{2b}))) \\
&\quad \mid (W, (\emptyset, v_{1a}), (\emptyset, v_{1b})) \in \mathcal{V}[\![\tau_1]\!]_\rho \\
&\quad \wedge (W, (\emptyset, v_{2a}), (\emptyset, v_{2b})) \in \mathcal{V}[\![\tau_2]\!]_\rho\} \\
\mathcal{V}[\![\tau_1 + \tau_2]\!]_\rho &= \{(W, (\emptyset, \text{inl } v_1), (\emptyset, \text{inl } v_2)) \\
&\quad \mid (W, (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\tau_1]\!]_\rho\} \\
\cup \{(W, (\emptyset, \text{inr } v_1), (\emptyset, \text{inr } v_2)) \mid (W, (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\tau_2]\!]_\rho\} \\
\mathcal{V}[\![\tau_1 \rightarrow \tau_2]\!]_\rho &= \{(W, (\emptyset, \lambda x. \{e\})) \mid \forall v W'. \\
&\quad W \sqsubset_\emptyset W' \wedge (W', (\emptyset, v)) \in \mathcal{V}[\![\tau_1]\!]_\rho \\
&\quad \implies (W', (\emptyset, [x \mapsto v] e)) \in \mathcal{E}[\![\tau_2]\!]_\rho\} \\
\mathcal{V}[\![\text{ref } \tau]\!]_\rho &= \{(W, (\emptyset, \ell_1), (\emptyset, \ell_2)) \mid W. \Psi(\ell_1, \ell_2) = |\mathcal{V}[\![\tau]\!]_{\rho| W.k}\} \\
\mathcal{V}[\![\forall \alpha. \tau]\!]_\rho &= \{(W, (\emptyset, \lambda.e_1), (\emptyset, \lambda.e_2)) \mid \forall R \in \text{UnrTyp}, W'. \\
&\quad W \sqsubset_{\emptyset, \emptyset} W' \implies (W', (\emptyset, e_1), (\emptyset, e_2)) \in \mathcal{E}[\![\tau]\!]_{\rho[\alpha \mapsto R]}\} \\
\mathcal{V}[\![\alpha]\!]_\rho &= \rho(\alpha) \\
\mathcal{V}[\![\text{unit}]\!]. &= \{(W, (\emptyset, ()), (\emptyset, ()))\} \\
\mathcal{V}[\![\text{bool}]\!]_\rho &= \{(W, (\emptyset, 0), (\emptyset, 0))\} \\
&\quad \cup \{(W, (\emptyset, n_1), (\emptyset, n_2)) \mid n_1 \neq 0 \wedge n_2 \neq 0\} \\
\mathcal{V}[\![\text{int}]\!]. &= \{(W, (\emptyset, n), (\emptyset, n)) \mid n \in \mathbb{Z}\} \\
\mathcal{V}[\![\tau_1 \multimap \tau_2]\!]. &= \{(W, (\emptyset, \lambda x. \{e\})) \mid \forall \Phi \vee W'. \\
&\quad W \sqsubset_\emptyset W' \wedge (W', (\Phi, v)) \in \mathcal{V}[\![\tau_1]\!]. \\
&\quad \implies ((W'.k, W'.\Psi, W'.\Theta \uplus \ell \mapsto \Phi), \\
&\quad (\emptyset, [x \mapsto \text{guard}(v, \ell)] e)) \in \mathcal{E}[\![\tau_2]\!].\} \\
\mathcal{V}[\![\tau_1 \multimap \bullet \tau_2]\!]. &= \{(W, (\Phi, \lambda a_\bullet. \{e\})) \mid \\
&\quad \forall \Phi' f_1 \vee W'. W \sqsubset_\Phi W' \wedge (W', (\Phi', v)) \in \mathcal{V}[\![\tau_1]\!]. \\
&\quad \wedge \Phi \cap \Phi' = \emptyset \wedge f \notin \Phi \uplus \Phi' \uplus \text{flags}(W') \\
&\quad \implies (W', (\Phi \uplus \Phi' \uplus \{f\}, [a_\bullet \mapsto \text{protect}(v, f)] e)) \\
&\quad \in \mathcal{E}[\![\tau_2]\!].\} \\
\mathcal{V}[\![\text{!}\tau]\!]. &= \{(W, (\emptyset, v_1), (\emptyset, v_2)) \mid (W, (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\tau]\!].\} \\
\mathcal{V}[\![\tau_1 \otimes \tau_2]\!]. &= \{(W, (\Phi_1 \uplus \Phi'_1, (v_{1a}, v_{2a})), (\Phi_2 \uplus \Phi'_2, (v_{1b}, v_{2b}))) \\
&\quad \mid (W, (\Phi_1, v_{1a}), (\Phi_2, v_{1b})) \in \mathcal{V}[\![\tau_1]\!]. \\
&\quad \wedge (W, (\Phi'_1, v_{2a}), (\Phi'_2, v_{2b})) \in \mathcal{V}[\![\tau_2]\!].\} \\
\mathcal{V}[\![\tau_1 \& \tau_2]\!]. &= \{(W, (\Phi_1, (\lambda_. \{e_{1a}\}), \lambda_. \{e_{2a}\})), \\
&\quad (\Phi_2, (\lambda_. \{e_{1b}\}, \lambda_. \{e_{2b}\}))) \\
&\quad \mid (W, (\Phi_1, e_{1a}), (\Phi_2, e_{1b})) \in \mathcal{E}[\![\tau_1]\!]. \\
&\quad \wedge (W, (\Phi_1, e_{2a}), (\Phi_2, e_{2b})) \in \mathcal{E}[\![\tau_2]\!].\} \\
\mathcal{E}[\![\tau]\!]_\rho &= \{(W, (\Phi, e)) \mid \text{freevars}(e) = \emptyset \wedge \\
&\quad \forall \Phi_r, H: W, e', H', j < W.k. \Phi_r \# \Phi \wedge \Phi_r \uplus \Phi : W \wedge \\
&\quad \langle \Phi_r \uplus \text{flags}(W) \uplus \Phi, H, e \rangle \xrightarrow{j} \langle \Phi', H', e' \rangle \rightsquigarrow \\
&\quad \implies e' = \text{fail CONV} \vee (\exists \Phi_f \Phi_g W'. \\
&\quad \Phi' = \Phi_r \uplus \text{flags}(W') \uplus \Phi_f \uplus \Phi_g \\
&\quad \wedge W \sqsubseteq_{\Phi_r} W' \wedge H' : W' \wedge (W', (\Phi_f, e')) \in \mathcal{V}[\![\tau]\!]_\rho)\} \\
(k, \Psi, \Theta) \sqsubseteq_\Phi (j, \Psi', \Theta') &\triangleq (j, \Psi', \Theta') \in \text{World}_j \wedge \\
j \leq k \wedge \Phi \# \text{flags}(k, \Psi, \Theta) \wedge \Phi \# \text{flags}(j, \Psi', \Theta') \\
\wedge \forall \ell \in \text{dom}(\Psi). \lfloor \Psi(\ell) \rfloor_j = \Psi'(\ell) \wedge \\
\forall \ell \in \text{dom}(\Theta). (\ell) \in \text{dom}(\Theta') \wedge \\
(\Theta(\ell) = \text{USED} \implies \Theta'(\ell) = \text{USED}) \\
\wedge (\Theta(\ell) = \Phi \implies \Theta'(\ell) = (\text{USED} \vee \Phi))
\end{aligned}$$

Figure 5.13: **MiniML** & **Affi** Value and Expression Relation (static).

Then, by applying $(W, (\emptyset, \mathbf{e}^+), (\emptyset, \mathbf{e}^+)) \in \mathcal{E}[\tau]_.$, we find that either $\mathbf{e}'_1 = \text{fail CONV}$ or there exist Φ'', H''_2, v_2 such that $\langle \Phi'_2 \uplus \text{flags}(W, 2), H'_2, e'_2 \rangle \xrightarrow{*} \langle \Phi'', H''_2, v_2 \rangle$ and e'_1 and v_2 are in the value relation with some world and sets of static flags. Ergo, since expressions in the value relation are values, e'_1 is a value. Finally, since e'_1 being a value implies $e' = Z(e'_1)$, we find that e' is a value. \square

Theorem 5.0.40 (Type Safety for **Affi**). *For any **Affi** term \mathbf{e} where $\cdot; \cdot; \cdot; \cdot \vdash \mathbf{e} : \tau$ and for any heap H , if $\langle H, \mathbf{e}^+ \rangle \xrightarrow{*} \langle H', \mathbf{e}' \rangle$, then we know from the logical relation that either $\mathbf{e}' = \text{fail CONV}$, \mathbf{e}' is a value, or there exist H'', \mathbf{e}'' such that $\langle H', \mathbf{e}' \rangle \rightarrow \langle H'', \mathbf{e}'' \rangle$.*

Proof. This proof is identical to that of [MiniML](#). \square

Note that to prove our type safety theorems, Theorems 5.0.39 and 5.0.40 we used Lemma B.2.20 which states that, if $\langle H, \mathbf{e} \rangle \xrightarrow{*} \langle H', \mathbf{e}' \rangle \not\rightarrow$, then for any Φ , $\langle \Phi, H, \mathbf{e} \rangle \xrightarrow{*} \langle \Phi'_1, H'_1, e'_1 \rangle \not\rightarrow$. This lemma is necessary because the given assumption of the type safety theorem is that the configuration $\langle H, \mathbf{e} \rangle$ steps under the normal operational semantics, but to apply the expression relation, we need that a corresponding configuration steps to an irreducible configuration under the phantom operational semantics.

Although our phantom flag realizability model was largely motivated by efficiency concerns with the dynamic enforcement of affinity, more broadly, it demonstrates how one can build complex static reasoning into the model even if such reasoning is absent from the target. Indeed, the actual target language, which source programs are compiled to and run in, has not changed; the augmentations exist *only in the model*. In this way, the preservation of source invariants is subtle: it is not that the types actually exist in the target (via runtime invariants or actual target types), but rather that the operational behavior of the target is exactly what the type interpretations characterize.

6

CASE STUDY: MEMORY MANAGEMENT AND POLYMORPHISM

Our final case study for value interoperability considers how [Miniml](#), whose references are garbage collected, can interoperate with core [L³](#), a language with safe strong updates despite memory aliasing, supported via linear capabilities ([Morrisett et al., 2005](#); [Ahmed et al., 2007](#)). This case study primarily highlights how different memory management strategies can interoperate safely, in particular, that manually managed linear references can be converted to garbage-collected references without copying. This is of particular interest as more low-level code is written in Rust, a language with an ownership discipline on memory that similarly could allow safe transfer of memory to garbage-collected languages.

We also use this case study to explore how polymorphism/generics in one language can be used, via a form of interoperability, from the other. This is interesting because significant effort has gone into adding generics to languages that did not originally support them, in order to more easily build certain re-usable libraries.¹ While we are not claiming that interoperability could entirely replace built-in polymorphism, sound support for cross-language type instantiation and polymorphic libraries presents a possible alternative, especially for smaller, perhaps more special-purpose, languages. This would allow us to write something like:

```
map((λx : int.x + 1)⟨int⟩→⟨int⟩[[1, 2, 3]]list ⟨int⟩)
```

where the [blue](#) language supports polymorphism, and has a generic `map` function, while the [pink](#) language does not. Of course, since convertibility is still driving this, in addition to using a concrete `intlist`, `[1, 2, 3]`, as above, the language without polymorphism could convert entirely different (non-list) concrete representations into similar polymorphic ones — i.e., implementing a sort of polymorphic interface at the boundary. For example, rather than an `intlist` (or a `stringlist`), in the example above, one could start with an `intarray` or `intbtree`, or any number of other traversable data structures that could be converted to `list int` (or any `list α`).

LANGUAGES We present the syntax of [L³](#) and [Miniml](#), augmented with forms for interoperability, in Figure 6.1. Their static semantics are in

¹ e.g., Java 1.5/5, C# 2.0 ([Kennedy and Syme, 2001](#)) and more recently, in the Go programming language

L³	
Type τ	::= unit bool $\tau \otimes \tau$ $\tau \multimap \tau$! τ ptr ζ cap $\zeta \tau$ $\forall \zeta. \tau$ $\exists \zeta. \tau$
Value v	::= $\lambda x : \tau. e$ () \mathbb{B} (v, v) !v $\Lambda \zeta. e$ $\lceil \zeta, v \rceil$
MinimL	
Expr. e	::= v x (e, e) e e let () = e in e if e e e let (x, x) = e in e let !x = e in e dupl e drop e new e free e swap e e e e [ζ] $\lceil \zeta, e \rceil$ let $\lceil \zeta, x \rceil$ = e in e $(\langle e \rangle_\tau)$ $\langle e \rangle_\tau$
DUPPLICABLE	= {unit, bool, ptr ζ , ! τ }
Type τ	::= α unit $\tau \rightarrow \tau$ $\forall \alpha. \tau$ ref τ $\langle \tau \rangle$
Expression e	::= x () $\lambda x : \tau. e$ $\Lambda \alpha. e$ e e e [τ] ref e !e e := e $(\langle e \rangle_\tau)$

Figure 6.1: Syntax for L³ and MinimL.

Figures 6.2 and 6.3 respectively. L³ has linear capability types cap $\zeta \tau$ (capability for abstract location ζ storing data of type τ), unrestricted pointer types ptr ζ to support aliasing, and location abstraction ($\Lambda \zeta. e : \forall \zeta. \tau$ and $\lceil \zeta, v \rceil : \exists \zeta. \tau$). The key design idea behind L³ is that the pointer can be separated from the capability and passed around in the program separately. At runtime, the capabilities will be erased, but the static discipline only allows pointers to be used with their capabilities (tied together with the type variables ζ), and requires capabilities to be used linearly. This enables safe in-place updates and low-level manual memory management while still supporting some flexibility in terms of pointer manipulation.

We highlight the key instructions here. **new** allocates memory and returns an existential package containing a capability and pointer ($\exists \zeta. \text{cap } \zeta \tau \otimes \text{ptr } \zeta$). **swap** takes a matching capability (cap $\zeta \tau_1$) and pointer ptr ζ and a value (of a possibly different type τ_2) and replaces what is stored, returning the capability and old value cap $\zeta \tau_2 \otimes \tau_1$. Note that since capabilities record the type of what is in the heap and are unique, strong updates are safe. Finally, **free** takes a package of a capability and pointer ($\exists \zeta. \text{cap } \zeta \tau \otimes \text{ptr } \zeta$) and frees the memory, consuming both in the process and returning what was stored there—any lingering pointers are harmless, as the necessary capability is now gone.

We compile both L³ and MinimL to an extension of the Scheme-like target LCVM that we used in the previous case study (see Figure 6.6 for L³ and Figure 6.7 for MinimL). The syntax of LCVM is shown in Figure 6.4, and it adds manual memory allocation to the version used in the previous case study. Its operational semantics is given in Figure 6.5. Whereas previously, we only had alloc, we now have free (which will error on a garbage-collected

$$\begin{array}{c}
\frac{}{\Delta; \Gamma; \Delta; x : \tau \vdash x : \tau} \quad \frac{\Delta; \Gamma; \Delta; x : \tau_1 \vdash e : \tau_2}{\Delta; \Gamma; \Delta; \Gamma \vdash \lambda x : \tau_1 e : \tau_1 \multimap \tau_2} \\[10pt]
\frac{\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 : \tau_1 \multimap \tau_2 \quad \Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 : \tau_1}{\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash e_1 e_2 : \tau_2} \quad \frac{}{\Delta; \Gamma; \Delta; \emptyset \vdash () : \text{unit}} \\[10pt]
\frac{\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 : \text{unit} \quad \Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 : \tau}{\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{let } () = e_1 \text{ in } e_1 : \tau} \quad \frac{}{\Delta; \Gamma; \Delta; \emptyset \vdash \text{B} : \text{bool}} \\[10pt]
\frac{\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 : \text{bool} \quad \Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 : \tau \quad \Delta; \Gamma; \Delta; \Gamma_2 \vdash e_3 : \tau}{\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{if } e_1 e_2 e_3 : \tau} \\[10pt]
\frac{\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 : \tau_1 \quad \Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 : \tau_2}{\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash (e_1, e_2) : \tau_1 \otimes \tau_2} \\[10pt]
\frac{\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 : \tau_1 \otimes \tau_2 \quad \Delta; \Gamma; \Delta; \Gamma_2, x_1 : \tau_1, x_2 : \tau_2 \vdash e_2 : \tau}{\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{let } (x_1, x_2) = e_1 \text{ in } e_2 : \tau} \\[10pt]
\frac{\Delta; \Gamma; \Delta; !\Gamma \vdash v : \tau \quad \Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 : !\tau_1 \quad \Delta; \Gamma; \Delta; \Gamma_2, x : \tau_1 \vdash e_2 : \tau_2}{\Delta; \Gamma; \Delta; !\Gamma \vdash !v : !\tau} \quad \frac{\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{let } !x = e_1 \text{ in } e_2 : \tau_2}{\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{let } !x = e_1 \text{ in } e_2 : \tau_2} \\[10pt]
\frac{\Delta; \Gamma; \Delta; \Gamma \vdash e : !\tau}{\Delta; \Gamma; \Delta; \Gamma \vdash \text{dupl } e : !\tau \otimes !\tau} \quad \frac{\Delta; \Gamma; \Delta; \Gamma \vdash e : !\tau}{\Delta; \Gamma; \Delta; \Gamma \vdash \text{drop } e : \text{unit}} \\[10pt]
\frac{\Delta; \Gamma; \Delta; \Gamma \vdash e : \tau}{\Delta; \Gamma; \Delta; \Gamma \vdash \text{new } e : \exists \zeta. \text{cap } \zeta \tau \otimes \text{!ptr } \zeta} \\[10pt]
\frac{\Delta; \Gamma; \Delta; \Gamma \vdash e : \exists \zeta. \text{cap } \zeta \tau \otimes \text{!ptr } \zeta}{\Delta; \Gamma; \Delta; \Gamma \vdash \text{free } e : \exists \zeta. \tau} \\[10pt]
\frac{\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 : \text{cap } \zeta \tau_1 \quad \Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 : \text{ptr } \zeta \quad \Delta; \Gamma; \Delta; \Gamma_3 \vdash e_3 : \tau_3}{\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3 \vdash \text{swap } e_1 e_2 e_3 : \text{cap } \zeta \tau_3 \otimes \tau_1} \\[10pt]
\frac{\Delta; \Gamma; \Delta; \zeta; \Gamma \vdash e : \tau}{\Delta; \Gamma; \Delta; \Gamma \vdash \Lambda \zeta. e : \forall \zeta. \tau} \quad \frac{\Delta; \Gamma; \Delta; \Gamma \vdash e : \forall \zeta. \tau \quad \zeta' \in \Delta}{\Delta; \Gamma; \Delta; \Gamma \vdash e [\zeta'] : [\zeta \mapsto \zeta'] \tau} \\[10pt]
\frac{\Delta; \Gamma; \Delta; \Gamma \vdash e : [\zeta \mapsto \zeta'] \tau \quad \zeta' \in \Delta}{\Delta; \Gamma; \Delta; \Gamma \vdash \text{!}\zeta', e^\top : \exists \zeta. \tau} \\[10pt]
\frac{\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 : \exists \zeta. \tau_1 \quad \Delta; \Gamma; \Delta; \zeta; \Gamma_2, x : \tau_1 \vdash e_2 : \tau_2 \quad FLV(\tau_2) \subseteq \Delta}{\Delta; \Gamma; \Delta; \Gamma_1 \vdash \text{let } \text{!}\zeta, x^\top = e_1 \text{ in } e_2} \\[10pt]
\frac{\Delta; !\Gamma; \Delta; \Gamma \vdash e : \tau \quad \tau \sim \tau}{\Delta; \Gamma; \Delta; !\Gamma \vdash (e)_\tau : \tau}
\end{array}$$

Figure 6.2: Static semantics for L^3 .

$$\begin{array}{c}
\frac{x : \tau \in \Gamma}{\Delta; !\Gamma; \Delta; \Gamma \vdash x : \tau} \quad \frac{}{\Delta; !\Gamma; \Delta; \Gamma \vdash () : \text{unit}} \quad \frac{\Delta; !\Gamma; \Delta; \Gamma, x : \tau_1 \vdash e : \tau_2}{\Delta; !\Gamma; \Delta; \Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2} \\
\\
\frac{\Delta; !\Gamma; \Delta; \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Delta; !\Gamma; \Delta; \Gamma \vdash e_2 : \tau_1}{\Delta; !\Gamma; \Delta; \Gamma \vdash e_1 \ e_2 : \tau_2} \quad \frac{\Delta; !\Gamma; \Delta; \alpha; \Gamma \vdash e : \tau}{\Delta; !\Gamma; \Delta; \Gamma \vdash \Lambda \alpha. e : \forall \alpha. \tau} \\
\\
\frac{\Delta; !\Gamma; \Delta; \Gamma \vdash e : \forall \alpha. \tau_2}{\Delta; !\Gamma; \Delta; \Gamma \vdash e [\tau_1] : [\alpha \mapsto \tau_1] \tau_2} \quad \frac{\Delta; !\Gamma; \Delta; \Gamma \vdash e : \tau}{\Delta; !\Gamma; \Delta; \Gamma \vdash \text{ref } e : \text{ref } \tau} \\
\\
\frac{\Delta; !\Gamma; \Delta; \Gamma \vdash e : \text{ref } \tau}{\Delta; !\Gamma; \Delta; \Gamma \vdash !e : \tau} \quad \frac{\Delta; !\Gamma; \Delta; \Gamma \vdash e_1 : \text{ref } \tau \quad \Delta; !\Gamma; \Delta; \Gamma \vdash e_2 : \tau}{\Delta; !\Gamma; \Delta; \Gamma \vdash e_1 := e_2 : \text{unit}} \\
\\
\frac{\Delta; \Gamma; \Delta; !\Gamma \vdash e : \tau \quad \tau \ \tau}{\Delta; !\Gamma; \Delta; \Gamma \vdash (e)_{\tau} : \tau}
\end{array}$$

Figure 6.3: Static semantics for MiniML.

location), an instruction (gcmov) to convert a manually managed location to garbage collected, and an instruction (callgc) to explicitly invoke the garbage collector. The last allows the compiler to decide where the GC can intercede (before allocation, in compiled code), and in doing so simplifies our model slightly. The memory management itself is captured in our heap definition, which allows the same location names to be used as either GC'd (\xrightarrow{gc}) or manually managed (\xrightarrow{m}), and re-used after garbage collection or manual free. Dereference (!e) and assignment (e := e) work on both types of reference (failing, of course, if it is manually managed and has been freed). This strategy of explicitly invoking the garbage collector and using a single pool of locations retains significant challenging aspects about garbage collectors while remaining simple enough to expose the interesting aspects of interoperation.

A few more detailed notes on the operational semantics of the target, given in Figure 6.5. Consider the third to last evaluation rule: Let $H : MHeap$ denote that H contains only mappings of the form $\ell \xrightarrow{m} v$ and let $H : GCHep$ denote that H contains only mappings of the form $\ell \xrightarrow{gc} v$.

Next, let $FL(e)$ and $FL(K[\cdot])$ be the set of locations that appear free in e and K , respectively. Then, we say that a location ℓ is directly reachable from a location ℓ' in the heap H if $\ell' \in \text{dom}(H)$ and $\ell \in FL(H(\ell'))$. We say that ℓ is reachable from ℓ' in H if one can construct a sequence of locations $\ell_0 = \ell', \ell_1, \ell_2, \dots, \ell_n = \ell$ where ℓ_i is directly reachable from ℓ_{i-1} in H for all $1 \leq i \leq n$. (Note that, for any location ℓ and heap H , ℓ is reachable from ℓ in H because we can construct the singleton sequence $\ell_0 = \ell$.)

Expressions e	$\coloneqq () \mid \mathbb{Z} \mid \ell \mid x \mid (e, e) \mid \text{fst } e \mid \text{snd } e \mid \text{inl } e \mid \text{inr } e \mid \text{if } e \{e\} \{e\}$ $\mid \text{match } e \times \{e\} y \{e\} \mid \text{let } x = e \text{ in } e \mid \lambda x \{e\} \mid e \ e \mid \text{ref } e$ $\mid \text{alloc } e \mid \text{free } e \mid \text{callgc} \mid \text{gcmov } e \mid !e \mid e := e \mid \text{fail } c$
Values v	$\coloneqq () \mid \mathbb{Z} \mid \ell \mid (v, v) \mid \lambda x. e$
Error Code c	$\coloneqq \text{TYPE} \mid \text{CONV} \mid \text{PTR}$
Heap H	$\coloneqq \ell \xrightarrow{gc} v, \dots \mid \ell \xrightarrow{m} v, \dots$
Evaluation Context K	$\coloneqq [] \mid (K, e) \mid (v, K) \mid \text{inl } K \mid \text{inr } K \mid \text{match } K \times \{e\} y \{e\}$ $\mid \text{if } K \{e\} \{e\} \mid \text{let } x = K \text{ in } e \mid K \ e \mid v \ K \mid \text{ref } K \mid \text{alloc } K$ $\mid \text{free } K \mid \text{gcmov } K \mid !K \mid K := e \mid v := K$

Figure 6.4: Syntax for LCV.

Finally, let $\text{reachablelocs}(H, L)$ be the set of all locations in $\text{dom}(H)$ reachable from L in H . (Note that $L \subseteq \text{reachablelocs}(H, L)$ by the previous parenthetical observation.)

Using the above definitions, we can define a step on whole programs that performs garbage collection. This step is indexed by a set of locations L denoting the locations that must be preserved and can not be garbage collected. The step shrinks the heap non-deterministically, ensuring that garbage-collectable locations which are reachable from either the program or L are not removed from the heap. Finally, we also allow whole programs to take steps according to \Rightarrow and to lift fail c errors out of evaluation contexts.

Returning to our source languages, as in the previous case study, we have boundary terms, $(\langle e \rangle)_\tau$ and $(\langle e \rangle)_\tau$, for *converting* a term and using it in the other language. Now, we also add new types $\langle \tau \rangle$, pronounced “foreign type”, and allow conversions from τ to $\langle \tau \rangle$ for *opaquely embedding*² types for use in polymorphic functions.

If a language supports polymorphism, then its type abstractions should be agnostic to the types that instantiate them, allowing them to range over not only host types, but indeed any foreign types as well. Doing so should not violate parametricity. However, the non-polymorphic language may need to make restrictions on how this power can be used, so as to not allow the polymorphic language to violate its invariants. To make this challenge material, our non-polymorphic language in this case study has linear resources (heap capabilities) that cannot, if we are to maintain soundness, be duplicated. This means, in particular, that whatever interoperability strategy we come up with cannot allow a linear capability from \mathbf{L}^3 to flow over to a **MinimL** function that duplicates it, even if such function is well-typed (and parametric) in **MinimL**.

² Similar to “lumps” in Matthews-Findler (Matthews and Findler, 2007), though they give a *single* lump type for all foreign types, i.e., they would have only $\langle \rangle$, rather than $\langle \tau \rangle$.

$$\begin{array}{c}
\frac{}{\langle H, \text{fst } (v, v') \rangle \Rightarrow \langle H, v \rangle} \quad \frac{v \neq (v_1, v_2)}{\langle H, \text{fst } v \rangle \Rightarrow \langle H, \text{fail TYPE} \rangle} \\
\\
\frac{}{\langle H, \text{snd } (v', v) \rangle \Rightarrow \langle H, v \rangle} \quad \frac{v \neq (v_1, v_2)}{\langle H, \text{snd } v \rangle \Rightarrow \langle H, \text{fail TYPE} \rangle} \\
\\
\frac{}{\langle H, \text{if } 0 \{e_1\} \{e_2\} \rangle \Rightarrow \langle H, e_1 \rangle} \quad \frac{n \neq 0}{\langle H, \text{if } n \{e_1\} \{e_2\} \rangle \Rightarrow \langle H, e_2 \rangle} \\
\\
\frac{v \notin \mathbb{Z}}{\langle H, \text{if } v \{e_1\} \{e_2\} \rangle \Rightarrow \langle H, \text{fail TYPE} \rangle} \\
\\
\frac{}{\langle H, \text{match } \text{inl } v \times \{e_1\} y \{e_2\} \rangle \Rightarrow \langle H, [x \mapsto v] e_1 \rangle} \\
\\
\frac{}{\langle H, \text{match } \text{inr } v \times \{e_1\} y \{e_2\} \rangle \Rightarrow \langle H, [y \mapsto v] e_2 \rangle} \\
\\
\frac{v \notin \{\text{inr } v', \text{inl } v'\}}{\langle H, \text{match } v \times \{e_1\} y \{e_2\} \rangle \Rightarrow \langle H, \text{fail TYPE} \rangle} \quad \frac{}{\langle H, \text{let } x = v \text{ in } e \rangle \Rightarrow \langle H, [x \mapsto v] e \rangle} \\
\\
\frac{}{\langle H, \lambda x \{e_b\} v \rangle \Rightarrow \langle H, [x \mapsto v] e_b \rangle} \quad \frac{v \neq \lambda x \{e\}}{\langle H, v v' \rangle \Rightarrow \langle H, \text{fail TYPE} \rangle} \\
\\
\frac{\ell \notin \text{dom}(H)}{\langle H, \text{ref } v \rangle \Rightarrow \langle H[\ell \xrightarrow{gc} v], \ell \rangle} \quad \frac{\ell \notin \text{dom}(H)}{\langle H, \text{alloc } v \rangle \Rightarrow \langle H[\ell \xrightarrow{m} v], \ell \rangle} \\
\\
\frac{\ell \xrightarrow{m} v \in H}{\langle H, \text{free } \ell \rangle \Rightarrow \langle H \setminus \ell, () \rangle} \quad \frac{\ell \xrightarrow{gc} v \in H}{\langle H, \text{free } \ell \rangle \Rightarrow \langle H, \text{fail PTR} \rangle} \\
\\
\frac{\ell \notin \text{dom}(H)}{\langle H, \text{free } \ell \rangle \Rightarrow \langle H, \text{fail PTR} \rangle} \quad \frac{\ell \xrightarrow{m} v \in H}{\langle H, \text{gcmov } \ell \rangle \Rightarrow \langle H[\ell \xrightarrow{gc} v], \ell \rangle} \\
\\
\frac{H[\ell] = v}{\langle H, !\ell \rangle \Rightarrow \langle H, v \rangle} \quad \frac{\ell \notin \text{dom}(H)}{\langle H, !\ell \rangle \Rightarrow \langle H, \text{fail PTR} \rangle} \quad \frac{v \neq \ell}{\langle H, !v \rangle \Rightarrow \langle H, \text{fail TYPE} \rangle} \\
\\
\frac{\ell \xrightarrow{m} v \in H}{\langle H, \ell := v' \rangle \Rightarrow \langle H[\ell \xrightarrow{m} v'], () \rangle} \quad \frac{\ell \xrightarrow{gc} v \in H}{\langle H, \ell := v' \rangle \Rightarrow \langle H[\ell \xrightarrow{gc} v'], () \rangle} \\
\\
\frac{\ell \notin \text{dom}(H)}{\langle H, \ell := v' \rangle \Rightarrow \langle H, \text{fail PTR} \rangle} \quad \frac{v \neq \ell}{\langle H, v := v' \rangle \Rightarrow \langle H, \text{fail TYPE} \rangle} \\
\\
\begin{array}{c}
H_{gc} : GCH \text{Heap} \quad H_m : M \text{Heap} \\
\text{reachablelocs}(H_{gc} \uplus H_m, \text{dom}(H_m) \cup FL(K[\cdot]) \cup L) \cap \text{dom}(H_{gc}) \subseteq \text{dom}(H'_{gc}) \\
H'_{gc} \subseteq H_{gc}
\end{array} \\
\frac{}{\langle H_{gc} \uplus H_m, K[\text{callgc}] \rangle \rightarrow_L \langle H'_{gc} \uplus H_m, K[()] \rangle} \\
\\
\frac{\langle H, e \rangle \Rightarrow \langle H', e' \rangle}{\langle H, K[e] \rangle \rightarrow_L \langle H', K[e'] \rangle} \quad \frac{K \neq [\cdot]}{\langle H, K[\text{fail } c] \rangle \rightarrow_L \langle H, \text{fail } c \rangle}
\end{array}$$

Figure 6.5: Operational semantics for LCVM.

x	\rightsquigarrow	x
$\lambda x : \tau.e$	\rightsquigarrow	$\lambda x.e^+$
$e_1 e_2$	\rightsquigarrow	$e_1^+ e_2^+$
()	\rightsquigarrow	()
let () = e_1 in e_2	\rightsquigarrow	let _ = e_1^+ in e_2^+
true	\rightsquigarrow	0
false	\rightsquigarrow	1
if $e_1 e_2 e_3$	\rightsquigarrow	if $e_1^+ e_2^+ e_3^+$
(e_1 , e_2)	\rightsquigarrow	(e_1^+ , e_2^+)
let (x_1 , x_2) = e_1 in e_2	\rightsquigarrow	let p = e_1^+ in let x_1 = fst p in let x_2 = snd p in e_2^+
!v	\rightsquigarrow	v^+
let !x = e_1 in e_2	\rightsquigarrow	let x = e_1^+ in e_2^+
dupl e	\rightsquigarrow	let x = e^+ in (x, x)
drop e	\rightsquigarrow	let _ = e^+ in ()
new e	\rightsquigarrow	let _ = callgc in let x_ℓ = alloc e^+ in (((), x_ℓ)
free e	\rightsquigarrow	let x = e^+ in let x_r = !(snd x) in let _ = free (snd x) in x_r
swap $e_c e_p e_v$	\rightsquigarrow	let x_p = e_p^+ in let _ = e_c in let $x_{v'}$ = ! x_p in let _ = (x_p := e_v^+) in (((), $x_{v'}$))
$\Lambda\zeta.e$	\rightsquigarrow	$\lambda_.e^+$
$e[\zeta]$	\rightsquigarrow	$e^+()$
$\ulcorner\zeta, e\urcorner$	\rightsquigarrow	e^+
let $\ulcorner\zeta, x\urcorner$ = e_1 in e_2	\rightsquigarrow	let x = e_1^+ in e_2^+
(e) $_\tau$	\rightsquigarrow	$C_{\tau\mapsto\tau}(e^+)$

Figure 6.6: Compiler for L^3 .

x	\rightsquigarrow	x
()	\rightsquigarrow	()
$\lambda x : \tau.e$	\rightsquigarrow	$\lambda x.e^+$
$e_1 e_2$	\rightsquigarrow	$e_1^+ e_2^+$
$\Lambda\alpha.e$	\rightsquigarrow	$\lambda_.e^+$
$e[\tau]$	\rightsquigarrow	$e^+()$
ref e	\rightsquigarrow	let _ = callgc in ref e^+
!e	\rightsquigarrow	$!e^+$
$e_1 := e_2$	\rightsquigarrow	$e_1^+ := e_2^+$
(e) $_\tau$	\rightsquigarrow	$C_{\tau\mapsto\tau}(e^+)$

Figure 6.7: Compiler for MiniML.

CONVERTIBILITY The first conversion that we want to highlight is between references. In \mathbf{L}^3 , pointers have capabilities that convey ownership, and thus to convert a pointer we also need the corresponding capability. For brevity, we may use $\mathbf{REF } \tau$ to abbreviate a capability+pointer package type.

$$\frac{C_{\tau \mapsto \tau}, C_{\tau \mapsto \tau} : \tau \sim \tau}{C_{\mathbf{REF } \tau \mapsto \mathbf{ref } \tau}, C_{\mathbf{ref } \tau \mapsto \mathbf{REF } \tau} : \mathbf{ref } \tau \sim \exists \zeta. \mathbf{cap } \zeta \tau \otimes !\mathbf{ptr } \zeta}$$

$$\begin{aligned} C_{\mathbf{REF } \tau \mapsto \mathbf{ref } \tau}(e) &\triangleq \text{let } x = \text{snd } e \text{ in} \\ &\quad \text{let } _ = (x := C_{\tau \mapsto \tau}(!x)) \text{ in gcmov } x \\ C_{\mathbf{ref } \tau \mapsto \mathbf{REF } \tau}(e) &\triangleq \text{let } x = \text{alloc } C_{\tau \mapsto \tau}(!e) \text{ in } ((,), x) \end{aligned}$$

The glue code itself is quite interesting: going from \mathbf{L}^3 to \mathbf{Miniml} , since the \mathbf{L}^3 type system guarantees that the capability in the capability+pointer package being converted is the only capability to this pointer, we can safely directly convert the pointer into a \mathbf{Miniml} pointer with gcmov after in-place replacing the contents with the result of converting.³ Going the other direction, from \mathbf{Miniml} to \mathbf{L}^3 , there is no way for us to know if there are other aliases to the reference, so we can't re-use the pointer. While we could simply disallow this conversion, and error if it were attempted, instead we copy and convert data into a freshly allocated manually managed location (note how, in the target, capabilities are erased to unit). In this case, as in many, there are multiple sound ways of converting, and it may be that a particular one makes more sense for your use case: we took the position that it was useful to get a copy of the data, unaliased, but perhaps a language designer would rather force the pointer to be dereferenced on the \mathbf{Miniml} side and the underlying data converted.

We account for interoperability of polymorphism in two parts. First, we have a *foreign type*, $\langle \tau \rangle$, which embeds an \mathbf{L}^3 type into the type grammar of \mathbf{Miniml} . This foreign type, like any \mathbf{Miniml} type, can be used to instantiate type abstractions, define functions, etc, but \mathbf{Miniml} has no introduction or elimination rules for it—terms of foreign type must come across from, and then be sent back to, \mathbf{L}^3 . These come by way of the conversion rule $\langle \tau \rangle \sim \tau$, which allow terms of the form $\langle e \rangle_{\langle \tau \rangle}$ (to bring an \mathbf{L}^3 term to \mathbf{Miniml}) and $\langle e \rangle_{\tau}$ (the reverse). Moreover, the conversion rule for foreign types restricts τ to a safe DUPLICABLE subset of types, but has no runtime consequences:

$$\frac{\tau \in \text{DUPLICABLE}}{C_{\langle \tau \rangle \mapsto \tau}, C_{\tau \mapsto \langle \tau \rangle} : \langle \tau \rangle \sim \tau} \quad \begin{aligned} C_{\langle \tau \rangle \mapsto \tau}(e) &\triangleq e \\ C_{\tau \mapsto \langle \tau \rangle}(e) &\triangleq e \end{aligned}$$

³ a less general rule that had a different premise might not need to convert, e.g., if the data was already compatible—see the case study in Chapter 4 for more details

To prove soundness we need to show that DUPLICABLE types are indeed safe to embed. The soundness condition depends on the expressive power of the two languages when viewed through the lens of polymorphism. In our case, since the non-polymorphic language is linear but the polymorphic one is not, we need to show that a DUPLICABLE type can be copied (i.e., none of its values own linear capabilities)—this includes **unit** and **bool**, but also **ptr** ζ and any type of the form $!\tau$. Now, consider examples using foreign types:

$$(\Lambda\alpha.\lambda x:\alpha.\lambda y:\alpha.y)[\langle \text{bool} \rangle] \langle \text{true} \rangle_{\langle \text{bool} \rangle} \langle \text{false} \rangle_{\langle \text{bool} \rangle} \quad (1)$$

$$(\lambda x : \text{BOOL}.x) \langle \text{true} \rangle_{\text{BOOL}} \text{ where } \text{BOOL} \triangleq \forall\alpha.\alpha \rightarrow \alpha \rightarrow \alpha \quad (2)$$

In (1), the leftmost expression is a polymorphic **Miniml** function that returns the second of its two arguments. It is instantiated with a foreign type, $\langle \text{bool} \rangle$. Next, two terms of type **bool** in **L**³ are embedded via the foreign conversion, $\langle \cdot \rangle_{\langle \text{bool} \rangle}$, which requires that **bool** ∈ DUPLICABLE. Not only does this mechanism allow **L**³ programmers to use polymorphic functions, but also **Miniml** programmers to use new base types. Of course, we could also convert the actual values, as in (2). To do so, we can define conversions between Church booleans in **Miniml** (which has no booleans) and ordinary booleans in **L**³:

$$\frac{}{\forall\alpha.\alpha \rightarrow \alpha \rightarrow \alpha \sim \text{bool}} \quad \begin{array}{c} C_{\text{BOOL} \rightarrow \text{bool}}(e) \triangleq e () 0 1 \\ C_{\text{bool} \rightarrow \text{BOOL}}(e) \triangleq \text{if}0 e \{ \Lambda\alpha.\lambda x:\alpha.\lambda y:\alpha.x \} \\ \quad \quad \quad \{ \Lambda\alpha.\lambda x:\alpha.\lambda y:\alpha.y \} \end{array}$$

SEMANTIC MODEL In Figure 6.8, we present the logical relation that we use to prove our conversions and entire languages sound. Supporting definitions relating to worlds and heaps are given in Figure 6.9.

Our model is inspired by that of core **L**³ (Ahmed et al., 2007), though ours is significantly more complex to account for garbage collection and interoperation with **Miniml**. The key is a careful distinction between owned (linear) manual memory, which is *local* and described by heap fragments associated with terms, and garbage-collected memory, which is *global* and described by the world W . Since memory can be freed (via garbage collection or manual free), reused, and moved from manual memory to garbage-collected memory, there are several constraints on how heap fragments and worlds may evolve so we can ensure safe memory usage.

With that in mind, our value interpretation of source types $\mathcal{V}[\tau]_\rho$ are sets of worlds and related heap-fragments-and-values (H, v) , where the heap

$$\begin{aligned}
\mathcal{V}[\alpha]_\rho &= \rho.F(\alpha) \\
\mathcal{V}[\text{unit}]_\rho &= \{(W, (\emptyset, ()), (\emptyset, ()))\} \\
\mathcal{V}[\tau_1 \rightarrow \tau_2]_\rho &= \{(W, (\emptyset, \lambda x_1.e_1), (\emptyset, \lambda x_2.e_2)) \mid \\
&\quad \forall W', v_1, v_2. W \sqsubseteq_{\emptyset, \emptyset, e_1, e_2} W' \wedge (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau_1]_\rho \implies \\
&\quad (W', (\emptyset, [x_1 \mapsto v_1]e_1), (\emptyset, [x_2 \mapsto v_2]e_2)) \in \mathcal{E}[\tau_2]_\rho\} \\
\mathcal{V}[\forall \alpha. \tau]_\rho &= \{(W, (\emptyset, \lambda_-e_1), (\emptyset, \lambda_-e_2)) \mid \\
&\quad \forall R \in \text{RelT}, W'. W \sqsubseteq_{\emptyset, \emptyset, e_1, e_2} W' \implies (W', (\emptyset, e_1), (\emptyset, e_2)) \in \mathcal{E}[\tau]_{\rho[F(\alpha) \mapsto R]}\} \\
\mathcal{V}[\text{ref } \tau]_\rho &= \{(W, (\emptyset, \ell_1), (\emptyset, \ell_2)) \mid W.\Psi(\ell_1, \ell_2) = [\mathcal{V}[\tau]_\rho]_{W.k}\} \\
\mathcal{V}[(\tau)]_\rho &= \mathcal{V}[\tau]_\rho \\
\mathcal{V}[\text{unit}]_\rho &= \{(W, (\emptyset, ()), (\emptyset, ()))\} \\
\mathcal{V}[\text{bool}]_\rho &= \{(W, (\emptyset, b), (\emptyset, b)) \mid b \in \{0, 1\}\} \\
\mathcal{V}[\tau_1 \otimes \tau_2]_\rho &= \{(W, (H_{1l} \uplus H_{1r}, (v_{1l}, v_{1r})), (H_{2l} \uplus H_{2r}, (v_{2l}, v_{2r}))) \mid \\
&\quad (W, (H_{1l}, v_{1l}), (H_{2l}, v_{2l})) \in \mathcal{V}[\tau_1]_\rho \wedge \\
&\quad (W, (H_{1r}, v_{1r}), (H_{2r}, v_{2r})) \in \mathcal{V}[\tau_2]_\rho\} \\
\mathcal{V}[\tau_1 \multimap \tau_2]_\rho &= \{(W, (H_1, \lambda x_1.e_1), (H_2, \lambda x_2.e_2)) \mid \forall W', H_{1v}, v_1, H_{2v}, v_2. \\
&\quad W \sqsubseteq_{H_1, H_2, e_1, e_2} W' \wedge (W', (H_{1v}, v_1), (H_{2v}, v_2)) \in \mathcal{V}[\tau_1]_\rho \\
&\quad \implies (W', (H_1 \uplus H_{1v}, [x_1 \mapsto v_1]e_1), (H_2 \uplus H_{2v}, [x_2 \mapsto v_2]e_2)) \in \mathcal{E}[\tau_2]_\rho\} \\
\mathcal{V}[\text{!}\tau]_\rho &= \{(W, (\emptyset, v_1), (\emptyset, v_2)) \mid (W, (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau]_\rho\} \\
\mathcal{V}[\text{ptr } \zeta]_\rho &= \{(W, (\emptyset, \ell_1), (\emptyset, \ell_2)) \mid \rho.\text{L3}(\zeta) = (\ell_1, \ell_2)\} \\
\mathcal{V}[\text{cap } \zeta \tau]_\rho &= \{(W, (H_1 \uplus \{\ell_1 \mapsto v_1\}, ()), (H_2 \uplus \{\ell_2 \mapsto v_2\}, ())) \mid \\
&\quad \rho.\text{L3}(\zeta) = (\ell_1, \ell_2) \wedge (W, (H_1, v_1), (H_2, v_2)) \in \mathcal{V}[\tau]_\rho\} \\
\mathcal{V}[\forall \zeta. \tau]_\rho &= \{(W, (H_1, \lambda_-e_1), (H_2, \lambda_-e_2)) \mid \forall \ell_1 \ell_2. (W, (H_1, e_1), (H_2, e_2)) \in \mathcal{E}[\tau]_{\rho[\text{L3}(\zeta) \mapsto (\ell_1, \ell_2)]}\} \\
\mathcal{V}[\exists \zeta. \tau]_\rho &= \{(W, (H_1, v_1), (H_2, v_2)) \mid \exists \ell_1 \ell_2. (W, (H_1, v_1), (H_2, v_2)) \in \mathcal{V}[\tau]_{\rho[\text{L3}(\zeta) \mapsto (\ell_1, \ell_2)]}\} \\
\mathcal{E}[\tau]_\rho &= \{(W, (H_1, e_1), (H_2, e_2)) \mid \\
&\quad \forall L_1, L_2, v_1, H_{1g+}, H_{2g+} : W, H_{1+} : MHeap, H_{1*}. \\
&\quad (H_{1g+} \uplus H_1 \uplus H_{1+}, e_1) \xrightarrow{*}_{L_1} (H_{1*}, v_1) \not\rightarrow_{L_1} \\
&\quad \implies \exists H'_1, H'_{1g}, \forall H_{2+} : MHeap. \exists H'_2, W', H'_{2g}, v_2. \\
&\quad H_{1*} = H'_{1g} \uplus H'_1 \uplus H_{1+} \wedge H'_{1g}, H'_{2g} : W' \wedge \\
&\quad W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))} W' \wedge \\
&\quad (W', (H'_1, v_1), (H'_2, v_2)) \in \mathcal{V}[\tau]_\rho \wedge \\
&\quad (H_{2g+} \uplus H_2 \uplus H_{2+}, e_2) \xrightarrow{*}_{L_2} (H'_{2g} \uplus H'_2 \uplus H_{2+}, v_2) \not\rightarrow_{L_2} \\
&\quad \wedge H_{1'} = H_{2'} = \emptyset\}
\end{aligned}$$

Note that the parts highlighted in **MiniML** colors only apply to types τ from **MiniML**, not types τ from **L3**.

$$\gamma_{\text{locs}}(\rho) \equiv \{x_\zeta \mapsto (\ell_1, \ell_2) \mid \zeta \mapsto (\ell_1, \ell_2) \in \rho\}$$

$$\begin{aligned}
\mathcal{G}[\cdot]_\rho &= \{(W, \cdot)\} \\
\mathcal{G}[\Gamma, x : \tau]_\rho &= \{(W, \gamma[x \mapsto (v_1, v_2)]) \mid (W, \gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau]_\rho\} \\
\mathcal{G}[\cdot]_\rho &= \{(W, \emptyset, \emptyset, \cdot)\} \\
\mathcal{G}[\Gamma, x : \tau]_\rho &= \{(W, H_1 \uplus H_{1x}, H_2 \uplus H_{2x}, \gamma[x \mapsto (v_1, v_2)]) \mid \\
&\quad (W, H_1, H_2, \gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, (H_{1x}, v_1), (H_{2x}, v_2)) \in \mathcal{V}[\tau]_\rho\} \\
\mathcal{D}[\cdot] &= \{\cdot\} \\
\mathcal{D}[\Delta, \alpha] &= \{\rho[\alpha \mapsto R] \mid \rho \in \mathcal{D}[\Delta] \wedge R \in \text{RelT}\} \\
\mathcal{D}[\Delta, \zeta] &= \{\rho[\zeta \mapsto (\ell_1, \ell_2)] \mid \rho \in \mathcal{D}[\Delta]\}
\end{aligned}$$

$$\begin{aligned}
\Delta; \Gamma; \Delta; \Gamma \vdash e_1 \preceq e_2 : \tau &\equiv \\
\forall \rho, \gamma_L, \gamma_\Gamma, \rho.\text{L3} \in \mathcal{D}[\Delta] \wedge \rho.F \in \mathcal{D}[\Delta] \wedge (\emptyset, \emptyset, \gamma_L.\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge \gamma_\Gamma \in \mathcal{G}[\Gamma]_\rho \wedge \gamma_L.\Delta = \gamma_{\text{locs}}(\rho.\text{L3}) \\
\implies (\emptyset, \gamma_L^1(\gamma_\Gamma^1(e_1^+)), \emptyset, \gamma_L^2(\gamma_\Gamma^2(e_2^+))) &\in \mathcal{E}[\tau]_\rho \\
\Delta; \Gamma; \Delta; \Gamma \vdash e_1 \preceq e_2 : \tau &\equiv \\
\forall \rho, \gamma_\Gamma, H_1, H_2, \rho.F \in \mathcal{D}[\Delta] \wedge \rho.\text{L3} \in \mathcal{D}[\Delta] \wedge \gamma_\Gamma \in \mathcal{G}[\Gamma]_\rho \wedge (H_1, H_2, \gamma_L.\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge \gamma_L.\Delta = \gamma_{\text{locs}}(\rho.\text{L3}) \\
\implies (H_1, \gamma_\Gamma^1(\gamma_L^1(e_1^+)), H_2, \gamma_\Gamma^2(\gamma_L^2(e_2^+))) &\in \mathcal{E}[\tau]_\rho
\end{aligned}$$

Figure 6.8: Logical Relation for **MiniML** and **L3**.

$$\begin{aligned}
World_n &= \{(k, \Psi) \mid k < n \wedge \Psi \subset HeapTy_k \wedge \text{dom}(\Psi) \text{ is a bijection}\} \\
World &= \bigcup_n World_n \\
HeapTy_n &= \{(\ell_1, \ell_2) \mapsto Typ_n, \dots\} \\
Atom_n &= \{(W, (\mathsf{H}_1, \mathsf{e}_1), (\mathsf{H}_2, \mathsf{e}_2)) \mid W \in World_n \wedge \mathsf{H}_1 : MHeap \wedge \mathsf{H}_2 : MHeap \wedge \\
&\quad \wedge \mathsf{H}_1 \# \text{dom}(W.\Psi)^1 \wedge \mathsf{H}_2 \# \text{dom}(W.\Psi)^2\} \\
AtomVal_n &= \{(W, (\mathsf{H}_1, \mathsf{v}_1), (\mathsf{H}_2, \mathsf{v}_2)) \in Atom_n\} \\
Atom &= \bigcup_n Atom_n \\
AtomVal &= \bigcup_n AtomVal_n \\
[R]_j &= \{(W, (\mathsf{H}_1, \mathsf{e}_1), (\mathsf{H}_2, \mathsf{e}_2)) \mid (W, (\mathsf{H}_1, \mathsf{e}_1), (\mathsf{H}_2, \mathsf{e}_2)) \in R \wedge W.k < j\} \\
[\Psi]_j &= \{(\ell_1, \ell_2) \mapsto [R]_j \mid (\ell_1, \ell_2) \mapsto R \in \Psi\} \\
\mathsf{H} &= \{\ell \xrightarrow{m} \mathsf{v}, \dots\} \uplus \{\ell \xrightarrow{gc} \mathsf{v}, \dots\} \\
\mathsf{H}_1 : GCH heap \wedge \mathsf{H}_2 : GCH heap \wedge \\
\forall (\ell_1, \ell_2) \mapsto R \in W.\Psi. \exists \mathsf{v}_1, \mathsf{v}_2. \ell_1 \xrightarrow{gc} \mathsf{v}_1 \in \mathsf{H}_1 \wedge \ell_2 \xrightarrow{gc} \mathsf{v}_2 \in \mathsf{H}_2 \wedge (\triangleright W, (\emptyset, \mathsf{v}_1), (\emptyset, \mathsf{v}_2)) \in R \\
&\wedge \mathbb{L}.1 \# \text{dom}((\Psi')^1) \wedge \mathbb{L}.2 \# \text{dom}((\Psi')^2) \\
&\wedge \forall (\ell_1, \ell_2) \in \eta. \Psi'(\ell_1, \ell_2) = [\Psi(\ell_1, \ell_2)]_j \\
W_1 \sqsubset_{\mathbb{L}, \eta} W_2 &\triangleq W_1.k > W_2.k \wedge W_1 \sqsubseteq_{\mathbb{L}, \eta} W_2 \\
W_1 \sqsubseteq_{\mathsf{H}_1, \mathsf{H}_2, \mathsf{e}_1, \mathsf{e}_2} W_2 &\triangleq W_1 \sqsubseteq_{(\text{dom}(\mathsf{H}_1), \text{dom}(\mathsf{H}_2)), \eta} W_2 \\
\eta &= \text{rchgclocs}(W_1, FL(\text{cod}(\mathsf{H}_1)) \cup FL(\mathsf{e}_1), FL(\text{cod}(\mathsf{H}_2)) \cup FL(\mathsf{e}_2)) \\
Typ_n &= \{R \in 2^{AtomVal_n} \mid \forall (W, (\mathsf{H}_1, \mathsf{v}_1), (\mathsf{H}_2, \mathsf{v}_2)) \in R. \forall W'. W \sqsubseteq_{\mathsf{H}_1, \mathsf{H}_2, \mathsf{v}_1, \mathsf{v}_2} W' \\
&\quad \implies (W', (\mathsf{H}_1, \mathsf{v}_1), (\mathsf{H}_2, \mathsf{v}_2)) \in R\} \\
Typ &= \{R \in 2^{AtomVal} \mid \forall k. [R]_k \in Typ_k\}
\end{aligned}$$

Figure 6.9: Supporting definitions for Logical Relation for **Miniml** and **L³**.

fragment H paired with value v is the portion of the manually managed heap that v *owns*.

The relational substitution ρ maps type variables α to arbitrary type interpretations R and location variables ζ to concrete locations ℓ . Since **MiniML** cannot own manual (linear) memory, all cases of $\mathcal{V}[\tau]_\rho$ have empty \emptyset heap fragments. However, during evaluation, memory could be allocated and subsequently freed so the expression relation does not have that restriction. In \mathbf{L}^3 , pointer types $\text{ptr } \zeta$ do not own locations, so they can be freely copied. Rather, linear capabilities $\text{cap } \zeta \tau$ convey ownership of the location ℓ that ζ maps to and the heap fragment H pointed to by ℓ .

In the expression relation $\mathcal{E}[\tau]_\rho$, we run the expression with a set of pinned locations (L) that the garbage collector should not touch (which may come from an outer context if we are evaluating a subterm), a garbage-collected heap fragment that satisfies the world (H_{g+}), an arbitrary disjoint manually allocated ($M\text{Heap}$) “rest” of the heap (H_r), composed with the owned fragment (H). Then, assuming e terminates at v , we expect the following four facts: (1) the “rest” heap H_r is unchanged, (2) the garbage-collected portion H_{g+} has been transformed to H'_g , (3) the owned portion H has been transformed into H' , and (4) $(W', (H', v)) \in \mathcal{V}[\tau]_\rho$, where W' is a future world the transformed GC'd portion of the heap H'_g must satisfy.s

Critical to the relation is world extension, written $\sqsubseteq_{\mathbb{L}, \eta}$, which indicates how our logical worlds can evolve over time. In typical logical relations for state, the heap grows monotonically and no location ever has its type change, which world extension captures. But, in our setting, the future heap might have deallocated, overwritten, re-used memory (and re-used it between the GC and manual allocation). We can't just allow arbitrary future states, however, as the semantics of types do dictate restrictions on what has to happen in the heap. In particular, there are two sets of locations that we need to keep careful track of; the rest can change freely. The first are manually managed locations that the GC can't disturb, which index \mathbb{L} captures. Those are generally just the owned locations of term that we are currently running. The second are the garbage collected locations that we must preserve in the heap, at the same type (but we can change the value of), captured by η . We also have a syntactic shorthand, denoted by \sqsubseteq , that is indexed by the heap H and the expressions e . This syntactic shorthand is defined so that \mathbb{L} takes its manually managed locations from the domain of H while η takes its garbage collected locations as the locations in the original world that are present in either some value in the heap H or the expression e . Finally, we often use rchgclocs in order to compute η when using world extension; $\text{rchgclocs}(W, S)$ is the set of locations in the world W that are actually mentioned in the set S ; i.e., $\text{rchgclocs}(W, S) = \text{dom}(W) \cap S$.

While our target supports dynamic failure (in the form of the `fail` term), our logical relation rules out that possibility, ensuring that there are no

errors from the source nor from the conversion. This is, of course, a choice we made, which may be stronger than desired for some languages (and, indeed, for our previous two case studies), but given our choice of conversions, it is possible.

With the logical relation in hand, we can prove type soundness in the same manner as for the previous two case studies. As before, we include the lemma and theorem statements below, but defer the proofs, which are quite mechanical, to Appendix C.

Our convertibility soundness result proves that our conversions above between garbage-collected and manual references, as well as L^3 booleans and `MiniML` Church booleans (described above) are sound. We also show that $\tau_1 \rightarrow \tau_2 \sim !(\neg \tau_1 \rightarrow \neg \tau_2)$ assuming $\tau_1 \sim \tau_1$ and $\tau_2 \sim \tau_2$.

Theorem 6.0.1 (Convertibility Soundness). *If $\tau_A \sim \tau_B$ then for all ρ ,*

1. $\forall (W, (\mathsf{H}_1, e_1), (\mathsf{H}_2, e_2)) \in \mathcal{E}[\![\tau_A]\!]_\rho. (W, (\mathsf{H}_1, C_{\tau_A \mapsto \tau_B} e_1), (\mathsf{H}_2, C_{\tau_A \mapsto \tau_B} e_2)) \in \mathcal{E}[\![\tau_B]\!]_\rho; \text{ and}$
2. $\forall (W, (\mathsf{H}_1, e_1), (\mathsf{H}_2, e_2)) \in \mathcal{E}[\![\tau_B]\!]_\rho. (W, (\mathsf{H}_1, C_{\tau_B \mapsto \tau_A} e_1), (\mathsf{H}_2, C_{\tau_B \mapsto \tau_A} e_2)) \in \mathcal{E}[\![\tau_A]\!]_\rho$

Proof. See C.0.15. □

Lemma 6.0.2 (Compat `x`).

$$\Delta; !\Gamma; \Delta; \Gamma, x : \tau \vdash x \preceq x : \tau$$

Proof. See C.0.16. □

Lemma 6.0.3 (Compat `()`).

$$\Delta; !\Gamma; \Delta; \Gamma \vdash () \preceq () : \text{unit}$$

Proof. See C.0.17. □

Lemma 6.0.4 (Compat `λx : τ.e`). *If $\Delta; !\Gamma; \Delta; \Gamma, x : \tau_1 \vdash e \preceq e : \tau_2$, then*

$$\Delta; !\Gamma; \Delta; \Gamma, x : \tau_1 \vdash \lambda x : \tau_1. e \preceq \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2$$

Proof. See C.0.18. □

Lemma 6.0.5 (Compat `e1 e2`). *If $\Delta; !\Gamma; \Delta; \Gamma \vdash e_1 \preceq e_1 : \tau_1 \rightarrow \tau_2$ and $\Delta; !\Gamma; \Delta; \Gamma \vdash e_2 \preceq e_2 : \tau_1$, then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash e_1 e_2 \preceq e_1 e_2 : \tau_2$$

Proof. See C.0.19. □

Lemma 6.0.6 (Compat $\Lambda\alpha.e$). *If $\Delta; !\Gamma; \Delta, \alpha; \Gamma \vdash e \preceq e : \tau$, then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash \Lambda\alpha.e \preceq \Lambda\alpha.e : \forall\alpha.\tau$$

Proof. See C.0.20. \square

Lemma 6.0.7 (Compat $e[\tau]$). *If $\Delta; !\Gamma; \Delta; \Gamma \vdash e \preceq e : \forall\alpha.\tau_2$, then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash e[\tau_1] \preceq e[\tau_1] : [\alpha \mapsto \tau_1]\tau_2$$

Proof. See C.0.21. \square

Lemma 6.0.8 (Compat $\text{ref } e$). *If $\Delta; !\Gamma; \Delta; \Gamma \vdash e \preceq e : \text{ref } \tau$, then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash \text{ref } e \preceq \text{ref } e : \text{ref } \tau$$

Proof. See C.0.22. \square

Lemma 6.0.9 (Compat $!e$). *If $\Delta; !\Gamma; \Delta; \Gamma \vdash e \preceq e : \text{ref } \tau$ then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash !e \preceq !e : \tau$$

Proof. See C.0.23. \square

Lemma 6.0.10 (Compat $e := e$). *If $\Delta; !\Gamma; \Delta; \Gamma \vdash e_1 \preceq e_1 : \text{ref } \tau$ and $\Delta; !\Gamma; \Delta; \Gamma \vdash e_2 \preceq e_2 : \tau$ then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash e_1 := e_2 \preceq e_1 := e_2 : \text{unit}$$

Proof. See C.0.24. \square

Lemma 6.0.11 (Compat $(e)_\tau$). *If $\Delta; \Gamma; \Delta; !\Gamma \vdash e \preceq e : \tau$ and $\tau \sim \tau$, then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash (e)_\tau \preceq (e)_\tau : \tau$$

Proof. See C.0.25. \square

Lemma 6.0.12 (Compat x).

$$\Delta; \Gamma; \Delta; x : \tau \vdash x \preceq x : \tau$$

Proof. See C.0.26. \square

Lemma 6.0.13 (Compat $\lambda x : \tau.e$). *If $\Delta; \Gamma; \Delta; \Gamma, x : \tau_1 \vdash e \preceq e : \tau_2$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \lambda x : \tau.e \preceq \lambda x : \tau.e : \tau_1 \multimap \tau_2$$

Proof. See C.0.27. \square

Lemma 6.0.14 (Compat $e_1 e_2$). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : \tau_1 \multimap \tau_2$ and $\Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 \preceq e_2 : \tau_1$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash e_1 e_2 \preceq e_1 e_2 : \tau_2$$

Proof. See C.0.28. □

Lemma 6.0.15 (Compat $()$).

$$\Delta; \Gamma; \Delta; \emptyset \vdash () \preceq () : \text{Unit}$$

Proof. See C.0.29. □

Lemma 6.0.16 (Compat \mathbb{B}). *If $b \in \mathbb{B}$, then*

$$\Delta; \Gamma; \Delta; \emptyset \vdash b \preceq b : \text{Bool}$$

Proof. See C.0.30. □

Lemma 6.0.17 (Compat $\text{let } ()$). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : \text{Unit}$ and $\Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 \preceq e_2 : \tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{let } () = e_1 \text{ in } e_2 \preceq \text{let } () = e_1 \text{ in } e_2 : \tau$$

Proof. See C.0.31. □

Lemma 6.0.18 (Compat if). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : \text{Bool}$ and $\Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 \preceq e_2 : \tau$ and $\Delta; \Gamma; \Delta; \Gamma_3 \vdash e_3 \preceq e_3 : \tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{if } e_1 e_2 e_3 \preceq \text{if } e_1 e_2 e_3 : \tau$$

Proof. See C.0.32. □

Lemma 6.0.19 (Compat (e_1, e_2)). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : \tau_1$ and $\Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 \preceq e_2 : \tau_2$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash (e_1, e_2) \preceq (e_1, e_2) : \tau_1 \otimes \tau_2$$

Proof. See C.0.33. □

Lemma 6.0.20 (Compat $\text{let } (x_1, x_2)$). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : \tau_1 \otimes \tau_2$ and $\Delta; \Gamma; \Delta; \Gamma_2, x_1 : \tau_1, x_2 : \tau_2 \vdash e_2 \preceq e_2 : \tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{let } (x_1, x_2) = e_1 \text{ in } e_2 \preceq \text{let } (x_1, x_2) = e_1 \text{ in } e_2 : \tau$$

Proof. See C.0.34. □

Lemma 6.0.21 (Compat $\mathbf{!v}$). *If $\Delta; \Gamma; \Delta; !\Gamma \vdash v \preceq v : \tau$, then*

$$\Delta; \Gamma; \Delta; !\Gamma \vdash !v \preceq !v : !\tau$$

Proof. See C.0.35. \square

Lemma 6.0.22 (Compat $\mathbf{let} \mathbf{!x}$). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : !\tau_1$ and $\Delta; \Gamma; \Delta; \Gamma_2, x : \tau_1 \vdash e_2 \preceq e_2 : \tau_2$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \mathbf{let} \mathbf{!x} = e_1 \mathbf{in} e_2 \preceq \mathbf{let} \mathbf{!x} = e_1 \mathbf{in} e_2 : \tau_2$$

Proof. See C.0.36. \square

Lemma 6.0.23 (Compat $\mathbf{dupl} \mathbf{e}$). *If $\Delta; \Gamma; \Delta; \Gamma \vdash e \preceq e : !\tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{dupl} \mathbf{e} \preceq \mathbf{dupl} \mathbf{e} : !\tau \otimes !\tau$$

Proof. See C.0.37. \square

Lemma 6.0.24 (Compat $\mathbf{drop} \mathbf{e}$). *If $\Delta; \Gamma; \Delta; \Gamma \vdash e \preceq e : !\tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{drop} \mathbf{e} \preceq \mathbf{drop} \mathbf{e} : \mathbf{Unit}$$

Proof. See C.0.38. \square

Lemma 6.0.25 (Compat $\mathbf{new} \mathbf{e}$). *If $\Delta; \Gamma; \Delta; \Gamma \vdash e \preceq e : \tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{new} \mathbf{e} \preceq \mathbf{new} \mathbf{e} : \exists \zeta. \mathbf{cap} \zeta \tau \otimes !\mathbf{ptr} \zeta$$

Proof. See C.0.39. \square

Lemma 6.0.26 (Compat $\mathbf{free} \mathbf{e}$). *If $\Delta; \Gamma; \Delta; \Gamma \vdash e \preceq e : \exists \zeta. \mathbf{cap} \zeta \tau \otimes !\mathbf{ptr} \zeta$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{free} \mathbf{e} \preceq \mathbf{free} \mathbf{e} : \exists \zeta. \tau$$

Proof. See C.0.40. \square

Lemma 6.0.27 (Compat $\mathbf{swap} \mathbf{e}$). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : \mathbf{cap} \zeta \tau_1$, $\Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 \preceq e_2 : \mathbf{ptr} \zeta$, and $\Delta; \Gamma; \Delta; \Gamma_3 \vdash e_3 \preceq e_3 : \tau_3$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{swap} \mathbf{e}_1 \mathbf{e}_2 \mathbf{e}_3 \preceq \mathbf{swap} \mathbf{e}_1 \mathbf{e}_2 \mathbf{e}_3 : \mathbf{cap} \zeta \tau_3 \otimes \tau_1$$

Proof. See C.0.41. \square

Lemma 6.0.28 (Compat $\Lambda \zeta. \mathbf{e}$). *If $\Delta; \Gamma; \Delta, \zeta; \Gamma \vdash e \preceq e : \tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \Lambda \zeta. \mathbf{e} \preceq \Lambda \zeta. \mathbf{e} : \forall \zeta. \tau$$

Proof. See C.0.42. \square

Lemma 6.0.29 (Compat $e[\zeta']$). *If $\Delta; \Gamma; \Delta; \Gamma \vdash e \preceq e : \forall \zeta. \tau$ and $\zeta' \in \Delta$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash e[\zeta'] \preceq e[\zeta'] : [\zeta \mapsto \zeta']\tau$$

Proof. See C.0.43. \square

Lemma 6.0.30 (Compat $\lceil \zeta, e \rceil$). *If $\Delta; \Gamma; \Delta; \Gamma \vdash e \preceq e : [\zeta \mapsto \zeta']\tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \lceil \zeta', e \rceil \preceq \lceil \zeta', e \rceil : \exists \zeta. \tau$$

Proof. See C.0.44. \square

Lemma 6.0.31 (Compat $\text{let } \lceil \zeta, x \rceil$). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : \exists \zeta. \tau_1$, $\Delta; \Gamma; \Delta, \zeta; \Gamma_2, x : \tau_1 \vdash e_2 \preceq e_2 : \tau_2$ and $FLV(\tau_2) \subseteq \Delta$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{let } \lceil \zeta, x \rceil = e_1 \text{ in } e_2 \preceq \text{let } \lceil \zeta, x \rceil = e_1 \text{ in } e_2 : \tau_2$$

Proof. See C.0.45. \square

Lemma 6.0.32 (Compat $(\langle e \rangle_\tau)$). *If $\Delta; !\Gamma; \Delta; \Gamma \vdash e \preceq e : \tau$ and $\tau \sim \tau$, then*

$$\Delta; \Gamma; \Delta; !\Gamma \vdash (\langle e \rangle_\tau) \preceq (\langle e \rangle_\tau) : \tau$$

Proof. See C.0.46. \square

Lemma 6.0.33 (Fundamental Property). *If $\Delta; \Gamma; \Delta; \Gamma \vdash e : \tau$, then $\Delta; \Gamma; \Delta; \Gamma \vdash e \preceq e : \tau$ and if $\Delta; \Gamma; \Delta; \Gamma \vdash e : \tau$, then $\Delta; \Gamma; \Delta; \Gamma \vdash e \preceq e : \tau$.*

Proof. By induction on typing derivation, relying on the following compatibility lemmas, which have to exist for every typing rule in both source languages. \square

Theorem 6.0.34 (Type Safety for [MiniML](#)). *If $\cdot; \cdot; \cdot; \cdot \vdash e : \tau$, then for any heap H , if $(H, e^+) \xrightarrow{*} (H', e')$, either there exist H'', e'' such that $(H', e') \rightarrow (H'', e'')$ or e' is a value.*

Proof. By the fundamental property, since the environments under which e is typechecked are empty, $(\cdot, (\emptyset, e^+), (\emptyset, e^+)) \in \mathcal{E}[\tau]$.

Then, either $(H', e') \rightarrow (H'', e'')$ or (H', e') is irreducible. If (H', e') is irreducible, we can apply the expression relation and find that there exists a world W and expression v_2 such that $(W, (\emptyset, e'), (\emptyset, v_2)) \in \mathcal{V}[\tau]$. Since expressions in the value relation are target values, this suffices to show that e' is a value. \square

Theorem 6.0.35 (Type Safety for [L³](#)). *If $\cdot; \cdot; \cdot; \cdot \vdash e : \tau$, then for any heap H , if $(H, e^+) \xrightarrow{*} (H', e')$, either there exist H'', e'' such that $(H', e') \rightarrow (H'', e'')$ or e' is a value.*

Proof. By the fundamental property, since the environments under which \mathbf{e} is typechecked are empty, $(\cdot, (\emptyset, \mathbf{e}^+), (\emptyset, \mathbf{e}^+)) \in \mathcal{E}[\tau]$.

Then, either $(H', e') \rightarrow (H'', e'')$ or (H', e') is irreducible. If (H', e') is irreducible, we can apply the expression relation and find that there exists a world W , heaps H'_1, H'_2 , and an expression v_2 such that $(W, (H'_1, e'), (H'_2, v_2)) \in \mathcal{V}[\tau]$. Since expressions in the value relation are target values, this suffices to show that e' is a value. \square

7

DISCUSSION: VALUE INTEROPERABILITY

7.1 ASYMMETRIC CONVERTIBILITY

Thus, we have “partial type equivalences” as described in (Dagand et al., 2016).

Throughout this dissertation, we have presented convertibility as a symmetric relation $\tau_A \sim \tau_B$, to mean that τ_A and τ_B are interconvertible via some target-level glue code. We have shown how this setup can apply to many different situations: even if two types are not isomorphic, the conversion can fail in either direction. However, there is a downside to this approach: if truly only one direction should ever be possible, we defer errors when converting in the other direction to runtime. We could instead implement the system with a directed convertibility relation, $\tau_A \succ \tau_B$. In that case, each conversion would still admit the possibility of dynamic failure, and thus $\tau_A \sim \tau_B$ can be recovered by $\tau_A \succ \tau_B \wedge \tau_B \succ \tau_A$. But the benefit is that if we never wanted a particular direction to succeed (for example, converting garbage-collected pointers to linear ones, as in Chapter 6), we could only provide one direction and yield a static error in the other direction. Now, there is a slight usability downside to the asymmetric approach: if higher-order values can cross the boundary, then the direction of conversion that is needed switches with the polarity of the type: i.e., to convert a function $\tau_1 \rightarrow \tau_2$ to $\tau'_1 \rightarrow \tau'_2$, we would need $\tau'_1 \succ \tau_1$ and $\tau_2 \succ \tau'_2$. For a type like $(\tau_1 \rightarrow \tau_2) \rightarrow \tau_3$, it becomes more confusing, and perhaps the runtime errors would be more descriptive. But an asymmetric presentation of our framework could account for both, and all the underlying development would be identical.

PRECISION There is another way that we could make the relations asymmetric: have $\tau_A \succ \tau_B$ mean not that there is no conversion from τ_B to τ_A , but that errors can only occur in that conversion, and that τ_A to τ_B will always succeed. In some ways, this is a reasonable proposal, because only in artificially created scenarios have we found we want errors in both directions. Rather, usually one type is more precise than the other, and thus converting to the more precise type may fail, but converting from it never will. This notion of type precision is of critical importance to gradual typing, as it underlies the notion of the gradual guarantee (Siek et al., 2015) or graduality (New and Ahmed, 2018; New et al., 2020). In our more general interoperability setting, we haven’t yet discovered analogous theorems, so making the convertibility relation capture precision doesn’t seem of value. It’s possible that will change in the future!

7.2 RUST AND UNSAFE

One of the most important features of the Rust programming language, and the subject of the RustBelt research project (Jung et al., 2018), is the `unsafe` language feature. `unsafe` allows a few things, but primarily, dereferencing raw pointers, which clearly can violate invariants of Rust’s type system if done incorrectly. This functionality is used to implement data structures and code that would not otherwise typecheck, and the claim is that it is better to have Rust and `unsafe` than to do the alternative: link against C. But there are issues that make us question this. First, the semantics of `unsafe` are still up for debate at the writing of this dissertation. Indeed, the semantics of C are much better understood, as there have been significant efforts to explain precise pointer provenance behavior. And, more broadly, we wonder if a better approach than building the escape hatch into the language, as with Rust `unsafe`, would be to have a separate proper low-level language that had precise semantics and could be linked against. One candidate is the Clight language defined within the CompCert project (Leroy, 2009), which is a significant subset of the C programming language, fully specified. A Rust compiler linking against Clight could rely on exactly what behavior could occur, which would seem to be a benefit over as-yet unclear semantics of `unsafe`. Of course, the downside is that programmers would have to write in Clight rather than in (almost) Rust, but the benefit is that data structures defined this way could be re-used in other languages that adopted the same FFI-with-Clight strategy.

7.3 EXISTENTIAL TYPES

In Chapter 6, we showed how to build conversions involving universal types by providing foreign opaque types, but handling existential types likely requires additional work. With the same “foreign type” mechanism, we could support defining data structures and operations over them and passing both across. For example, we could pass an expression of type $\langle \text{int} \rangle \times (\langle \text{int} \rangle \rightarrow \langle \text{int} \rangle) \times (\langle \text{int} \rangle \rightarrow \text{int})$, for a counter defined as an integer. That provides some degree of abstraction, but doesn’t, for example, disallow passing the $\langle \text{int} \rangle$ back to some other code that expects that type. We could, however, in the language with existential types, pack that to $\exists \alpha. \alpha \times (\alpha \rightarrow \alpha) \times (\alpha \rightarrow \text{int})$.

More interesting is when both languages have polymorphism. In that case, if we wanted to convert abstract types, we would need to generalize our convertibility rules to handle open types, i.e., $\Delta \vdash \tau \sim \tau'$. If the interpretation of type variables were the same in both languages (i.e., in our model this would mean that both were drawn from the same relation), this would be sufficient. If, however, the interpretation of type variables

were different in the two languages (e.g., we do this in the case study in Chapter 5, where you can see the use of *UnrTyp* in $\mathcal{V}[\![\forall \alpha.\tau]\!]_\rho$), we would need, in our source type systems, some form of bounded polymorphism in order to restrict the judgment to variables that were equivalent. Otherwise, it would be impossible to prove convertibility rules sound, since a proof of convertibility would allow too many values in the interpretation of type variables.

Part III

MOVING BEHAVIOR ACROSS LANGUAGE BOUNDARIES

8

A RECIPE FOR SOUNDLY INCORPORATING FOREIGN BEHAVIOR

In this chapter, we present a recipe for proving soundness of languages that import inexpressible behavior via an FFI, and in particular, prove that interaction sound by giving novel and precise types to that imported code. This chapter demonstrates the framework by extending the simple language used in the tutorial in Chapter 2. By seeing the full framework in the small, the reader should be prepared for the more substantive case studies in Chapters 9 and 10.

Throughout Part II, we considered how to exchange values across language boundaries. What this required was that there be a type in the core language that could accurately describe the foreign code. But what if there isn't such a type? Consider a pure language that wishes to use state to implement a particular algorithm more efficiently. We take, as assumption, that the pure language should remain pure, since if it were stateful, it would have types to capture the stateful behavior and the approach of Part II would apply. What can we do? The simplest approach is to implement the entire stateful algorithm in the stateful language, and only return the result back to the pure language. In an appropriate model, we should be able to argue that such computation is extensionally pure and, thus, does not threaten the soundness of our pure language. This is similar to the proof mechanism used by Timany et al. (2017); Jacobs et al. (2022) to show the purity of Haskell's ST monad. Further, since the code is extensionally pure, it can be given a type from our pure language, and thus our value interoperability strategy would work.

However, there is a downside to this approach: all of the code that uses the stateful behavior must be written in the stateful language, and only once it has been essentially encapsulated can the result be brought across, as an opaque entity, to the pure language. In this part of the dissertation, we present a novel design for interoperability that maintains soundness while eliminating this weakness. The core idea is to enrich, in a controlled way, the types of our core language such that we can import the foreign constructs that we wish to use. We call these novel types "linking types". The foreign behavior, with its linking types, will then be encapsulated from the rest of the program so that the original invariants of the core language are not disrupted by these novel types. But, unlike in the alternative proposed solution, this approach allows the core-language programmer to still use

most of their abstractions, albeit with some novel behavior to account for, when writing their code.

In the concrete example of a pure language programmer wanting to use state to implement a particular algorithm more efficiently, they may import state primitives for reading and writing from the heap, but the core of the algorithm would all be written in their core language. Encapsulation would be responsible for taking the resulting stateful computation (which is given linking types that capture that it is stateful) and ensuring that it was extensionally pure; i.e., that it could have been implemented purely, if less efficiently or in a more verbose way.

One important restriction to our approach is that we require that the target have some way of performing that encapsulation, which for some effects may be difficult or maybe even impossible. For state, we accomplish this by means of a region-like mechanism: the stateful code is encapsulated by ensuring that any heap it used is discarded. Encapsulating exceptions is easier: all escaping exceptions must be caught. But, other effects may be harder to encapsulate: e.g., non-termination could be accounted for, but only by turning all computations into computations that might time out and thus return no value. It also should be stated that doing almost any of this encapsulation requires that the target have features that the source does not have, as the type of reflective computation necessary for encapsulation is often what is ruled out by safe type systems. Thus, while our choice of untyped and lower-level targets is not a requirement, we do require aspects of expressivity that are more common in those types of languages.

The remainder of this chapter demonstrates the idea in recipe form, using a small example, along the same lines as Chapter 3. Then, in Chapters 9 and 10 we will explore in detail two more complex instantiations of the linking types approach, showing how to encapsulate foreign behavior while maintaining soundness.

THE FRAMEWORK

The inputs to the framework are a core language, [SimpleFunLang](#) (reused from Chapter 2); a target language *Lambda*; and a compiler, $e^+ = e$. Implicitly, there is a (or many) foreign language(s), but the framework only interacts with them *after compilation*, and thus the only thing we will need to know is the behavior, in terms of *Lambda* code, that we wish to link with cannot be captured by a type in [SimpleFunLang](#). This can, in general, be both code that captures behavior more *positively expressive* (can distinguish programs that are equivalent in [SimpleFunLang](#), e.g., stateful code from the perspective of a pure language) or *negatively expressive* (has equivalences that can be violated by [SimpleFunLang](#), e.g., types that capture finer distinctions than possible in your core language). For this section, we

We take positive and negative expressivity from Patterson and Ahmed (2017), where positive expressivity is expressive power defined by Felleisen (1990).

will use a *negative expressivity* example: a type β that includes *only* the target value *true*. The example in this section serves both as a roadmap of what is to come and a reference to refer back to. Also, the two subsequent case studies are both examples of *positive* expressivity, so this demonstrates the other possibility.

The first three steps of the recipe, extending the type system, defining type function relating core types to extended types, and building encapsulation wrappers (§8.0.1, §8.0.2, and §8.0.3) must be performed by the **designer** of the interoperability system, whereas the last six, which develop realizability models that prove soundness and encapsulation (§8.0.4, §8.0.5, and §8.0.6, §8.0.7, §8.0.8, and §8.0.9) should be performed by the **verifier** of the system. As we did for Part II, we make this distinction to highlight that we believe there is value in attempting an approximation of soundness; and in this case, we think the linking types design is an interesting contribution to the interoperability design space, even outside of the question of how such an interoperability system is proven sound.

SimpleFunLang

$$\begin{array}{lll} \text{Type } \tau & ::= & B \mid \tau \rightarrow \tau \\ \text{Expression } e & ::= & b_1 \mid b_2 \mid b_{\text{op}} \mid x \mid \text{fun}(x : \tau)\{e\} \mid e(e) \\ \text{Value } v & ::= & b_1 \mid b_2 \mid \text{fun}(x : \tau)\{e\} \end{array}$$

Lambda

$$\begin{array}{lll} \text{Expression } e & ::= & \text{true} \mid \text{false} \mid \text{if } e \ e \ e \mid x \mid \lambda x : \tau. e \mid e \ e \mid \text{fail} \\ \text{Value } v & ::= & \text{true} \mid \text{false} \mid \lambda x : \tau. e \\ \text{Evaluation context } E & ::= & [] \mid \text{if } E \ e \ e \mid E \ e \mid (\lambda x. e) \ E \end{array}$$

We reproduce both the static semantics and compiler from Chapter 2 for easy reference.

$$\frac{}{\Gamma \vdash b_1 : B} \quad \frac{}{\Gamma \vdash b_2 : B} \quad \frac{\Gamma \vdash e : B}{\Gamma \vdash b_{\text{op}}(e) : B} \quad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash \text{fun}(x : \tau)\{e\} : \tau \rightarrow \tau'} \quad \frac{\Gamma \vdash e : \tau \rightarrow \tau' \quad \Gamma \vdash e' : \tau}{\Gamma \vdash e(e') : \tau'}$$

$$e \rightsquigarrow e^+$$

$$\begin{array}{ll} b_1 & \rightsquigarrow \text{true} \\ b_2 & \rightsquigarrow \text{false} \\ b_{\text{op}} & \rightsquigarrow \lambda x. \text{if } x \text{ true fail} \\ \text{fun}(x : \tau)\{e\} & \rightsquigarrow \lambda x. e^+ \\ e(e') & \rightsquigarrow e^+ e'^+ \end{array}$$

8.0.1 Extend the type system (\vdash_+)

Our goal is to accommodate additional behavior: in this case, we want to have a type β that includes only *true*, as distinct from B , which includes *true* and *false*. This may seem a silly example, but such subsets are relevant and non-trivial in reality—consider tracking the sign of integers, or functions that are linear—in both cases, the type (a positive/negative integer, or a linear $\tau \rightarrow \tau'$) is a smaller set of target values than the type that our signless/unrestricted language had (integer and unrestricted $\tau \rightarrow \tau'$).

Our first task, then, is to give ourselves a way to reason statically about code that includes this novel type β . The first step for that is to give ourselves syntax for separating code that involves this novel type, which we will usually refer to as “linking code”, from the rest of our [SimpleFunLang](#) program. We call it “linking code”, as it is the code that uses the imported functionality, and will be eventually encapsulated. Programmers who write this code will need to have some understanding of the semantics of the imported functionality they are using, whereas programmers not writing linking code need not concern themselves with anything beyond [SimpleFunLang](#) semantics. In the examples in this dissertation, we will use a “wrap” expression (or boundary) to delineate this boundary, written $\{e\}_\tau^+$, where e is the aforementioned linking code, and τ is the type that the wrapped expression should be treated as in the rest of the [SimpleFunLang](#) program, and $+$ identifies the particular linking types extension (as multiple extensions can coexist). We do not provide wrapping in the other direction, but will allow bindings defined outside the wrapper to be used inside.

The code inside this expression will be typed according to our extended type system \vdash_+ . In general, the idea behind linking types extensions is that they give the minimal static reasoning that allows the programmer to use the feature. In particular, we will not give introduction rules for β , since values of this type will arise from linked code, not in code written in [SimpleFunLang](#). Whether we give elimination rules depends on the use: in this case, we could avoid it, judging that all β values should be consumed by linked functions (i.e., with signature $\beta \rightarrow \tau$), but doing so makes the example slightly too small. Instead, we will allow B_{op} on β values, and have it return a B value. This means the changes to the type system are to add one new base type in the grammar and add one new rule. We inherit the rest of the rules and syntax from [SimpleFunLang](#).

$$\text{Type } \tau ::= \beta \mid B \mid \tau \rightarrow \tau$$

$$\frac{\Gamma \vdash_+ e : \beta}{\Gamma \vdash_+ b_{op}(e) : B}$$

While we call this an “extension”, and in some cases it may be, in our presentation it is a proper new type system, formally related to the old one by the next step in the recipe. In more realistic implementations, we would want the extended type system to be able to significantly re-use the existing type infrastructure, perhaps via some plugin mechanism. We’ll discuss this more in Chapter 12.

TECHNICAL NOTE In the examples in Chapters 9 and 10, we have our changed type system distinguish between code that has the novel effect and does not. While this is in some sense clearer, and makes the proof obligations a bit more obvious, in the positive expressivity cases we need only have our types be able to express the more expressive behavior: i.e., it would suffice to have a single type for possibly stateful arrows, so long as our model was sufficiently powerful to distinguish between pure and stateful computations. In that case, the original language would have a pure function type and the extended type system would have a possibly stateful function type, but since we only typecheck one type of code at a time, we avoid the complexity of a modality. In the negative expressivity cases, we cannot simplify by avoiding modalities, since we need to be able to re-use core code (and thus our extended types must be able to capture these types) and the smaller types that form the extension. For example, if we had an extension for affine functions, we would need to be able to express both affine and unrestricted functions in the extended type system.

8.0.2 Define lift ($\uparrow\tau$) and lower ($\downarrow\tau$)

We relate our original types, τ , with our new types, which we write $\textcolor{pink}{\tau}$, via two type functions: lift (written \uparrow) converts a blue τ to a pink $\textcolor{pink}{\tau}$ and lower (written \downarrow) converts a pink $\textcolor{pink}{\tau}$ to a blue τ .

Intuitively, lift exists so that any value that has type τ should have type $\uparrow\tau$: this allows us to write code in `SimpleFunLang` and then within linking code reuse, at type $\uparrow\tau$, bindings that were defined outside the wrapping boundary. In the case presented in this section, all of the types from `SimpleFunLang` are available in our extended language, so lift behaves as an identity. In the case studies examined in Chapters 9 and 10, this isn’t the case.

Lower, on the other hand, should faithfully summarize a linking type $\textcolor{pink}{\tau}$ with a core type τ . Clearly, since $\textcolor{pink}{\tau}$ includes more behavior than τ , we cannot get the same implication as for lift, but the intention (later proved) is that the type is what results when the $\textcolor{pink}{\tau}$ computation is encapsulated. Indeed, the type $\downarrow\tau$ is the only interface through which `SimpleFunLang` can access the results of the linking code computation.

$$\begin{array}{rcl}
 \uparrow \mathbf{B} & = & \mathbf{B} \\
 \uparrow \tau_1 \rightarrow \tau_2 & = & \uparrow \tau_1 \rightarrow \uparrow \tau_2 \\
 \\
 \downarrow \beta & = & \mathbf{B} \\
 \downarrow \mathbf{B} & = & \mathbf{B} \\
 \downarrow \tau_1 \rightarrow \tau_2 & = & \downarrow \tau_1 \rightarrow \downarrow \tau_2
 \end{array}$$

8.0.3 Develop encapsulation wrappers ($\langle \downarrow \tau \rangle$)

The typing rule for our wrappers is the following:

$$\frac{\uparrow \Gamma \vdash e : \tau}{\Gamma \vdash \{e\}_{\downarrow \tau}^+ : \downarrow \tau}$$

i.e., if the term inside has type τ , then the wrapped term has type $\downarrow \tau$. For some types this may be a consequence of the static types, but for others, when we compile we may need to insert operational code (written $\langle \downarrow \tau \rangle$) that enforces this encapsulation. That is, we need to ensure that even if the type τ allows extra behavior, once wrapped, the term has no more behavior than $\downarrow \tau$. For linking types that capture *positive expressivity* this may require some runtime code to ensure the extra behavior does not escape, whereas linking types that capture *negative expressivity* likely will not need any such code. For example, in our the case study we will see in Chapter 10, our linking code can raise exceptions but our core language cannot, so when we compile these boundaries we would insert code that catches any escaping exceptions.

In the example we are using in this section, since β is a subset of \mathbf{B} , and all other types already exist in `SimpleFunLang`, so our target-level wrapping code is a no-op:

$$\langle \downarrow \tau \rangle[e] = e$$

8.0.4 Design a core realizability model ($\models \tau$)

The first three steps are all that the linking types **designer** needs to do: they now have an extended language in which programmers can write linking code, and lift (\uparrow) and lower (\downarrow) relate code in `SimpleFunLang` to the linking code. However, to prove that the resulting language, including the linking code, is sound, the **verifier** still has work to do.

First, if they have not already, they should characterize precisely which *Lambda* programs behave like which `SimpleFunLang` types. We build real-

Stable languages may only require this step once, since the same core model can be reused by multiple extensions.

izability models, rather than another mechanism for proving soundness, as we will later rely upon their ability to reason about target code that did not originate in our source.

First, we build a value relation: $\mathcal{V}[\tau]$ includes the set of values that behave as τ .

$$\begin{aligned}\mathcal{V}[B] &= \{\text{true}, \text{false}\} \\ \mathcal{V}[\tau_1 \rightarrow \tau_2] &= \{\lambda x.e \mid \forall v \in \mathcal{V}[\tau_1]. [x \mapsto v]e \in \mathcal{E}[\tau_2]\}\end{aligned}$$

Then, we build a model for computations, $\mathcal{E}[\tau]$. This should exactly mirror the statement of type soundness for **SimpleFunLang**, so it is crucial that it be agnostic to foreign behaviors. It may be that certain general behavior, like raising errors, should be included in this model, even when the errors are not relevant yet, since we will need to use this model to prove our encapsulation correct, and encapsulation may rely upon runtime errors.

$$\mathcal{E}[\tau] = \{e \mid \exists e'. e \xrightarrow{*} e' \wedge (e' = \text{fail} \vee e' \in \mathcal{V}[\tau])\}$$

8.0.5 Design an extended realizability model ($\models_E \tau$)

In the previous step, we built a model for our core language, indexing our relation by blue τ . Now, we build one for our extended language, indexing our relation by the pink τ . If the linking code allows for more *positive expressivity*, then these relations should include *more* values and computations than in the model for the core language. If, like in the example in this section, the linking code allows for more *negative expressivity*, the model will include relations with finer granularity than existed in the model for the core language.

$$\begin{aligned}\mathcal{V}^+[\beta] &= \{\text{true}\} \\ \mathcal{V}^+[B] &= \{\text{true}, \text{false}\} \\ \mathcal{V}^+[\tau_1 \rightarrow \tau_2] &= \{\lambda x.e \mid \forall v \in \mathcal{V}^+[\tau_1]. [x \mapsto v]e \in \mathcal{E}^+[\tau_2]\} \\ \mathcal{E}^+[\tau] &= \{e \mid \exists e'. e \xrightarrow{*} e' \wedge (e' = \text{fail} \vee e' \in \mathcal{V}^+[\tau])\}\end{aligned}$$

8.0.6 Prove lift ($\uparrow\tau$) sound

In §8.0.2 we insisted that lift maps **SimpleFunLang** types to identical linking types. We must verify this by showing an isomorphism between $\mathcal{V}[\tau]$ and $\mathcal{V}^+[\uparrow\tau]$, exploiting the fact that both are inhabited by *Lambda* values. We need both directions of this proof at different places. We need to prove that essentially $\mathcal{V}[\tau] \subseteq \mathcal{V}^+[\uparrow\tau]$ (modulo difference in world structure) to be able to use bindings defined outside linking code within it (at lifted type). We

use the other direction $\mathcal{V}^+[\uparrow\tau] \subseteq \mathcal{V}[\tau]$ as part of our encapsulation proof, shown in the next step §8.0.7.

Note that these proofs do not rely upon our relations having the same logical structure. Indeed, we'll see in the case study in Chapter 9 that our core language, which is pure, has no logical heap, but our extended language has state and thus a logical heap. While this means that the pure model is inhabited by pairs of step indices and terms and the stateful model has a heap typing in addition to the step index and term, the meaning of the lifted types should ensure that that additional structure should not be relevant for the types in question.

8.0.7 Prove encapsulation $\langle \downarrow\tau \rangle$ enforces lower $(\downarrow\tau)$

In §8.0.3 we insisted that $\langle \downarrow\tau \rangle$ encapsulate its result such that it behaves like a $\downarrow\tau$ in `SimpleFunLang`. We verify this by showing, for every $e \in \mathcal{E}^+[\tau]$, that $\langle \downarrow\tau \rangle[e] \in \mathcal{E}[\tau]$. To do so, it suffices to show that $\langle \downarrow\tau \rangle[e] \in \mathcal{E}^+[\uparrow\downarrow\tau]$ and appeal to the second half of the soundness of lifting from §8.0.6.

8.0.8 Prove “compatibility” lemmas

As is typical, we define semantic closing substitutions $\mathcal{G}[\Gamma]$ and $\mathcal{G}^+[\Gamma]$, where for every binding $x : \tau$ or $x : \tau$, we map x to a value $v \in \mathcal{V}[\tau]$ or $\mathcal{V}^+[\tau]$ respectively:

$$\begin{aligned}\mathcal{G}[\cdot] &= \{\cdot\} \\ \mathcal{G}[x : \tau, \Gamma] &= \{(x \mapsto v, \gamma) \mid v \in \mathcal{V}[\tau] \wedge \gamma \in \mathcal{G}[\Gamma]\} \\ \mathcal{G}^+[\cdot] &= \{\cdot\} \\ \mathcal{G}^+ [x : \tau, \Gamma] &= \{(x \mapsto v, \gamma) \mid v \in \mathcal{V}^+[\tau] \wedge \gamma \in \mathcal{G}^+[\Gamma]\}\end{aligned}$$

Using that substitution, we then prove that the compilation of any well-typed term $\Gamma \vdash e : \tau$, when closed by a substitution in $\mathcal{G}[\Gamma]$, is in $\mathcal{E}[\tau]$. Similarly, we show that the compilation of $\Gamma^+ \vdash e : \tau$, when closed by a substitution in $\mathcal{G}^+[\Gamma]$, is in $\mathcal{E}^+[\tau]$. For the boundary case, which must relate the core and extended models, appeal to the lemmas from §8.0.6 and §8.0.7.

8.0.9 Prove libraries semantically well-typed

For every compiled library e_f that we wish to link with, we must prove that the *Lambda* code e_f belongs to $\mathcal{E}^+[\tau]$.

Although we present the framework linearly, a full treatment will likely require many iterations of these steps. In particular, difficult (or impossible) proof cases may require changes to the implementation of the encapsulation wrappers. In such cases, we recommend looking to the models for guidance; since they are all inhabited by *Lambda* programs with the same operational semantics, they often suggest opportunities for (and threats to) enforcing encapsulation. Indeed, the division between implementation and verification is a soft one: most likely, the two will be developed together interactively. At the same time, we stress that the implementation steps can stand on their own as a valuable recipe for building better *approximations* of sound FFIs.

CASE STUDY: MUTABLE STATE

This section (Chapters 9 and 10) includes joint unpublished work with Andrew Wagner and Amal Ahmed.

In this case study, we exhibit the linking types design methodology by incorporating a library for mutable references into a language that otherwise has no access to the heap. In the linking code we will be able to allocate, read, and write to mutable references, but encapsulation means that the linking code must extensionally behave as if it were pure. This is not, of course, pointless, as mutable state is often used to improve the performance of algorithms that could, in principle, be implemented by threading a pure data structure, and thus are *extensionally* pure.

9.1 A FUNCTIONAL LANGUAGE

For this case study and the next, our core language, **FunLang** is a standard pure, eager, non-terminating functional language with imports.

It has both iso-recursive types (with `fold/unfold`) and recursive functions, as well as sums, products, and simple base types (`unit`, `int`, `bool`). We present the syntax (typeset in blue typewriter font) and static semantics in Figure 9.1. Despite the definition-like syntax, functions are still anonymous expressions; the function's name is only bound in its body, for recursive calls. Most of the rest of the syntax and static semantics is quite standard, with the exceptions of the imports and our wrapping term $\{\mathbf{e}\}_\tau$, which bear some explanation. In plenty of work, imports are ignored, and linking is defined by way of function application. While we could try to do this, having imports be explicit entities is useful for our work, as it allows us to be clear about where the imports can be used. For this reason, we put imports in a separate, explicit environment \mathbf{I} , distinct but threaded along with our normal environment Γ . Importantly, since imports exist to support linking code, the imports are only in scope within the wrapped linking code. We will show the typing rule for the wrapping term later, since the body will be checked with a linking types extension type system (which will also be introduced later), but it will typecheck under an environment that includes \mathbf{I} , unlike any of the terms *outside* the linking code.

In the remainder of this part of the dissertation, we will use the following as a running example:

```
fun fib(n : int){
    if n < 1 { 0 } { if n = 1 { 1 } { fib(n + -1) + fib(n + -2) } }
}
```

We choose this example not because it is a particularly fascinating program, but because it is small enough to serve as a good example yet large enough to permit a non-trivially useful incorporation of linked code.

Like many real languages, we do not provide a formal operational semantics for [FunLang](#), though clearly we could. Although one certainly has an operational semantics *in mind*, the observable semantics in our case is defined by an implementation. Here, that implementation is via compilation to a stack-based target language, [StackLang](#), similar to the target used in Chapter 4.

9.2 A STACK LANGUAGE

Our target language is an untyped, stack-based language called [StackLang](#), which is derived from [Kleffner \(2017\)](#), which in turn derives some features from [Levy \(2001\)](#). It is significantly more expressive than our source language, as target languages often are. On the other hand, it is not especially low-level, as it permits aggregate values and suspended computations (thunks) on the stack. We present the syntax in Figure 9.2, typeset in black typewriter font. The small-step operational semantics, shown in Figure 9.3, is defined as a relation on program configurations $\langle H ; S ; P \rangle$, which are triples of a heap, stack, and program.

Values are placed on the stack with `push`. The binary operators `add`, `less?`, and `equal?` operate on the two integers at the top of the stack. `if0` conditions on the integer at the top of the stack and continues with its first branch if it is zero, and with its second branch otherwise. Despite its syntax, `lam x.P` is not a value, but a computation (as in Call-By-Push-Value ([Levy, 2001](#))) that substitutes the top value on the stack for `x` inside `P`. On the other hand, the value thunk `P` is a suspended computation, so thunk `lam x.P` is the equivalent of a traditional lambda value. `call` takes the thunk `P` at the top of the stack and forces its computation, placing `P` at the head of the program. `fix` performs a fixpointing operation: it takes the thunk at the top of the stack and re-suspends it for recursive calls, and then forces one copy of its computation. `idx` and `len` operate on the array value at the top of the stack. `alloc`, `read`, `write`, and `free` perform standard heap operations, where any [StackLang](#) value can be stored in the heap. `shift k P` and `reset` provide delimited control ([Felleisen, 1988](#); [Danvy and Filinski, 1990](#)): `shift` captures the continuation up to the next `reset` and substitutes it for `k` in `P`. `getlocs`

Types τ $\begin{array}{l} \alpha \mid \text{unit} \mid \text{bool} \mid \text{int} \mid \tau \times \tau \mid \tau + \tau \mid \tau * \tau \\ \mid \mu\alpha.\tau \mid (\tau, \dots, \tau) \rightarrow \tau \end{array}$
 Expressions e $\begin{array}{l} () \mid \text{true} \mid \text{false} \mid \text{if } e \{e\} \mid n \mid e = e \\ \mid e < e \mid e + e \mid x \mid (e, e) \mid \text{fst } e \mid \text{snd } e \\ \mid \text{inl } e \mid \text{inr } e \mid \text{match } e \{x\} y \{e\} \mid \text{fold } e \\ \mid \text{unfold } e \mid \text{fun } f(x_1 : \tau_1, \dots, x_n : \tau_n) \{e\} \\ \mid e(e, \dots, e) \mid \{e\}_\tau \end{array}$
 Imports I $f : \tau \mid f : \tau, I$
 Programs P $e \mid \text{import}(I) e$

$$\begin{array}{c}
 \boxed{\vdash P : \tau} \qquad \frac{\cdot ; \cdot \vdash e : \tau}{\vdash e : \tau} \qquad \frac{I ; \cdot \vdash e : \tau}{\vdash \text{import}(I) e : \tau} \\
 \\
 \boxed{I ; \Gamma \vdash e : \tau} \qquad \frac{}{I ; \Gamma \vdash () : \text{unit}} \qquad \frac{}{I ; \Gamma \vdash \text{true/false} : \text{bool}} \qquad \frac{}{I ; \Gamma \vdash n : \text{int}} \\
 \\
 \frac{I ; \Gamma \vdash e : \text{bool} \quad I ; \Gamma \vdash e_1 : \tau \quad I ; \Gamma \vdash e_2 : \tau}{I ; \Gamma \vdash \text{if } e \{e_1\} \{e_2\} : \tau} \\
 \\
 \frac{I ; \Gamma \vdash e_1 : \text{int} \quad I ; \Gamma \vdash e_2 : \text{int}}{I ; \Gamma \vdash e_1 = e_2 : \text{bool}} \qquad \frac{I ; \Gamma \vdash e_1 : \text{int} \quad I ; \Gamma \vdash e_2 : \text{int}}{I ; \Gamma \vdash e_1 < e_2 : \text{bool}} \\
 \\
 \frac{I ; \Gamma \vdash e_1 : \text{int} \quad I ; \Gamma \vdash e_2 : \text{int}}{I ; \Gamma \vdash e_1 + e_2 : \text{int}} \qquad \frac{x : \tau \in \Gamma}{I ; \Gamma \vdash x : \tau} \\
 \\
 \frac{I ; \Gamma \vdash e_1 : \tau_1 \quad I ; \Gamma \vdash e_2 : \tau_2}{I ; \Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2} \qquad \frac{I ; \Gamma \vdash e : \tau_1 \times \tau_2}{I ; \Gamma \vdash \text{fst/snd } e : \tau_1 / \tau_2} \\
 \\
 \frac{I ; \Gamma \vdash e : \tau_1 / \tau_2 \quad \vdash \tau_2 / \tau_2}{I ; \Gamma \vdash \text{inl/inr } e : \tau_1 + \tau_2} \\
 \\
 \frac{I ; \Gamma \vdash e : \tau_1 + \tau_1 \quad \Gamma, x : \tau_1 \vdash e_1 : \tau \quad \Gamma, y : \tau_2 \vdash e_2 : \tau}{I ; \Gamma \vdash \text{match } e \{x\} y \{e_2\} : \tau} \\
 \\
 \frac{I ; \Gamma \vdash e : \tau[\mu\alpha.\tau/\alpha]}{I ; \Gamma \vdash \text{fold } e : \mu\alpha.\tau} \qquad \frac{I ; \Gamma \vdash e : \mu\alpha.\tau}{I ; \Gamma \vdash \text{unfold } e : \tau[\mu\alpha.\tau/\alpha]} \\
 \\
 \frac{\Gamma, f : (\tau_1, \dots, \tau_n) \rightarrow \tau', x_i : \tau_i \vdash e : \tau'}{I ; \Gamma \vdash \text{fun } f(x_1 : \tau_1, \dots, x_n : \tau_n) \{e\} : (\tau_1, \dots, \tau_n) \rightarrow \tau'}
 \end{array}$$

$$\frac{I ; \Gamma \vdash e : (\tau_1, \dots, \tau_n) \rightarrow \tau' \quad I ; \Gamma \vdash e_i : \tau_i}{I ; \Gamma \vdash e(e_1, \dots, e_n) : \tau'}$$

Figure 9.1: Syntax & static semantics for `FunLang`.

```

Stack S   :=  v, . . . , v | Fail c
Error Code c :=  TYPE | IDX | MEM | CTRL
Program P  :=  · | i; P
Value v    :=  n | thunk P | ℓ | [v, . . .]
Instruction i :=  push v | add | less? | equal? | if0 P P | lam x.P | call
                  | fix | idx | len | alloc | read | write | free | shift k P
                  | reset | getlocs | noop | fail c

```

Figure 9.2: Syntax for StackLang

provides reflective access to the heap: it takes the thunk `lam` and the value `v` at the top of the stack and maps the computation over all locations used in `v`.¹ Unsurprisingly, `noop` does nothing. Finally, `fail c` terminates execution of the entire program with the given error code. Every instruction with a type invariant on the stack uses `fail TYPE` when that invariant is not met, producing a *dynamic* type error; other errors are for unrecoverable problems which may be acceptable results according to a soundness theorem.

9.3 A COMPILER FOR `FunLang`

In Figure 9.4, we present a compiler from `FunLang` to StackLang, which induces the operational semantics of `FunLang`. Many cases are straightforward, but because the target has important consequences for how we link, we will consider them in some detail. We see that a base value is compiled to `push v`, where `v` is a target-level encoding of the value. Note that we use 0 both for `unit` and `true` (to match `if0`). In a typical functional language, like `FunLang`, the values constitute a subset of the expressions, and evaluation stops at values. However, in StackLang, evaluation only stops once the empty program has been reached, and we consider the value on the top of the stack to be the result. So, whereas `v` is the simplest `FunLang` program, `push v` is, by analogy, the simplest StackLang program, even though it takes a step.

`if` is mildly more complicated. We first compile the discriminant `e` (denoted `e+`), which, according to `e`'s type, should be a program fragment that terminates with a (semantic) boolean at the top of the stack. Thus, the type invariant for `if0` should be satisfied, and it can proceed with (the compilation of) the appropriate branch. The compilation of the binary operators follows this pattern.

`inl` and `inr` are slightly different because the result value needs to be tagged. We use arrays to store the tag, 0 or 1, along with the payload. To move the payload value off of the stack and into an array, we use `lam`,

¹ This is a minimal version of the heap reflection that shows up in target languages and is used to, e.g., implement a GC.

$\langle H ; S ; \text{push } v; P \rangle$	$\rightarrow \langle H ; S, v ; P \rangle$	$(S \neq \text{Fail } c)$
$\langle H ; \text{Fail } c ; \text{push } v; P \rangle$	$\rightarrow \langle H ; \text{Fail } c ; \text{fail TYPE} \rangle$	
$\langle H ; S, n_2, n_1 ; \text{add}; P \rangle$	$\rightarrow \langle H ; S, (n_1 + n_2) ; P \rangle$	
$\langle H ; S ; \text{add}; P \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$(S \neq S', n_2, n_1)$
$\langle H ; S, n_2, n_1 ; \text{less?}; P \rangle$	$\rightarrow \langle H ; S, 0 ; P \rangle$	$(n_1 < n_2)$
$\langle H ; S, n_2, n_1 ; \text{less?}; P \rangle$	$\rightarrow \langle H ; S, 1 ; P \rangle$	$(n_1 \geq n_2)$
$\langle H ; S ; \text{less?}; P \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$(S \neq S', n_2, n_1)$
$\langle H ; S, v, v ; \text{equal?}; P \rangle$	$\rightarrow \langle H ; S, 0 ; P \rangle$	
$\langle H ; S, v_2, v_1 ; \text{equal?}; P \rangle$	$\rightarrow \langle H ; S, 1 ; P \rangle$	$v_1 \neq v_2$
$\langle H ; S ; \text{equal?}; P \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$(S \neq S', v_2, v_1)$
$\langle H ; S, 0 ; \text{if0 } P_1 P_2; P \rangle$	$\rightarrow \langle H ; S ; P_1; P \rangle$	
$\langle H ; S, n ; \text{if0 } P_1 P_2; P \rangle$	$\rightarrow \langle H ; S ; P_2; P \rangle$	$(n \neq 0)$
$\langle H ; S ; \text{if0 } P_1 P_2; P \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$(S \neq S', n)$
$\langle H ; S, v ; \text{lam } x.P_1; P_2 \rangle$	$\rightarrow \langle H ; S ; [x \mapsto v]P_1; P_2 \rangle$	
$\langle H ; S ; \text{lam } x.P_1; P_2 \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$(S \neq S', v)$
$\langle H ; S, \text{thunk } P_1 ; \text{call}; P_2 \rangle$	$\rightarrow \langle H ; S ; P_1; P_2 \rangle$	
$\langle H ; S ; \text{call}; P_2 \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$(S \neq S', \text{thunk } P_1)$
$\langle H ; S, \text{thunk } P_1 ; \text{fix}; P_2 \rangle$	$\rightarrow \langle H ; S, \text{thunk} (\text{push } (\text{thunk } P_1), \text{fix}); P_1; P_2 \rangle$	
$\langle H ; S ; \text{fix}; P_2 \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$(S \neq S', \text{thunk } P_1)$
$\langle H ; S, [v_0, \dots, v_{n_2}], n_1 ; \text{idx}; P \rangle$	$\rightarrow \langle H ; S, v_{n_1} ; P \rangle$	$(n_1 \in [0, n_2])$
$\langle H ; S, [v_0, \dots, v_{n_2}], n_1 ; \text{idx}; P \rangle$	$\rightarrow \langle H ; S ; \text{fail IDX} \rangle$	$(n_1 \notin [0, n_2])$
$\langle H ; S ; \text{idx}; P \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$(S \neq S', [v_0, \dots, v_{n_2}], n_1)$
$\langle H ; S, [v_0, \dots, v_n] ; \text{len}; P \rangle$	$\rightarrow \langle H ; S, (n+1) ; P \rangle$	
$\langle H ; S ; \text{len}; P \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$(S \neq S', [v_0, \dots, v_n])$
$\langle H ; S, v ; \text{alloc}; P \rangle$	$\rightarrow \langle H \uplus \{\ell \mapsto v\} ; S, \ell ; P \rangle$	
$\langle H ; \cdot ; \text{alloc}; P \rangle$	$\rightarrow \langle H ; \cdot ; \text{fail TYPE} \rangle$	
$\langle H \uplus \{\ell \mapsto v\} ; S, \ell ; \text{read}; P \rangle$	$\rightarrow \langle H \uplus \{\ell \mapsto v\} ; S, v ; P \rangle$	
$\langle H ; S, \ell ; \text{read}; P \rangle$	$\rightarrow \langle H ; S ; \text{fail MEM} \rangle$	$\ell \notin \text{dom}(H)$
$\langle H ; S ; \text{read}; P \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$(S \neq S', \ell)$
$\langle H \uplus \{\ell \mapsto _\} ; S, \ell, v ; \text{write}; P \rangle$	$\rightarrow \langle H \uplus \{\ell \mapsto v\} ; S ; P \rangle$	
$\langle H ; S, \ell, v ; \text{write}; P \rangle$	$\rightarrow \langle H ; S ; \text{fail MEM} \rangle$	$\ell \notin \text{dom}(H)$
$\langle H ; S ; \text{write}; P \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$(S \neq S', \ell, v)$
$\langle H \uplus \{\ell \mapsto _\} ; S, \ell ; \text{free}; P \rangle$	$\rightarrow \langle H \} ; S ; P \rangle$	
$\langle H ; S, \ell ; \text{free}; P \rangle$	$\rightarrow \langle H ; S ; \text{fail MEM} \rangle$	$\ell \notin \text{dom}(H)$
$\langle H ; S ; \text{free}; P \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$(S \neq S', \ell)$
$\langle H ; S ; \text{shift } k P_1; P_2; \dots ; \text{reset}; P_3 \rangle$	$\rightarrow \langle H ; S ; [k \mapsto \text{thunk } P_2; \dots]P_1; P_3 \rangle$	$\text{reset} \notin P_2; \dots$
$\langle H ; S ; \text{shift } k P_1; P_2 \rangle$	$\rightarrow \langle H ; S ; \text{fail CTRL} \rangle$	$\text{reset} \notin P_2$
$\langle H ; S ; \text{reset}; P \rangle$	$\rightarrow \langle H ; S ; P \rangle$	
$\langle H ; S, \text{thunk lam } I.P_1, v ; \text{getlocs}; P_2 \rangle$	$\rightarrow \langle H ; S, \ell_1, \dots, \ell_n ; \text{lam } I.P_1; \dots ; \text{lam } I.P_1; P_2 \rangle$	$\ell_1, \dots, \ell_n = \text{flocs}(v)$
$\langle H ; S ; \text{getlocs}; P_2 \rangle$	$\rightarrow \langle H ; S ; \text{fail TYPE} \rangle$	$S \neq S', \text{thunk lam } I.P_1, \ell$
$\langle H ; S ; \text{noop}; P \rangle$	$\rightarrow \langle H ; S ; P \rangle$	
$\langle H ; S ; \text{fail } c; P \rangle$	$\rightarrow \langle H ; \text{Fail } c ; \cdot \rangle$	

Figure 9.3: Operational semantics for StackLang

which, as described earlier, is an *instruction* (not a value) that performs substitution with the value at the top of the stack. Pairs and projections are compiled similarly.

Compiling `match` is conceptually like compiling `if`, but its definition is more involved because one must destruct tagged values. e^+ should produce a tagged value at the top of the stack, so we copy it with the macro DUP (defined at the bottom of the figure), project out the payload (at index 1), SWAP the top two elements of the stack, and finally project out the tag. Now, we are ready to condition on the tag and, in the branches, substitute the payload.

In theory, one could entirely erase any remnant of the recursive operators `fold` and `unfold`. Indeed, we do just that for `fold`. However, for reasons that will become clear when we consider the model for soundness, we introduce a noop in the compilation of `unfold`. At a high level, `unfold` produces an expression at a larger type, which threatens the well-foundedness of our semantic model, as it is defined inductively over types. To reconcile this, we employ a standard trick and stratify the model, which requires that `unfold`⁺ take an extra step.

All that remains are `fun`s and application. For functions, fix does most of the heavy lifting. A compiled `fun` is a thunk that first pushes a thunk corresponding to the computation of the body (taking itself as the first argument, f), and then invokes fix. The result of fix will be to perform the fixpoint, passing itself as that first argument. The arguments are in reverse order so that effects (which, for now, include only divergence) are observed left-to-right. Application is conceptually straightforward; the only subtlety is that the compiled function (e^+) needs to be at the top of the stack in order to be called, but it needs to run *first* for a left-to-right evaluation order. Since StackLang does not have a built-in for indexing into the stack, we shuffle the function to the front as we evaluate the arguments.

9.4 LINKING WITH STATE

With a source, target, and compiler in hand, we are now ready to tackle the central problem of this part of the dissertation: how to safely encapsulate inexpressible behavior.

The `fib` program from §9.1 is a classic example of unnecessary exponential computation. A standard trick taught in most undergraduate programming courses is to use memoization, which traditionally requires mutable state. As a quick reminder, the strategy is to store intermediate results in a table so that later computations can reuse them without recomputing them. In our example, the table maps inputs n to their outputs `fastfib(n)`, so that each `fastfib(n)` is only computed once. What we want is to link with a mutable reference library providing `alloc`, `read`, and `write` functions,

$e \rightsquigarrow e^+$	
$()$	\rightsquigarrow push 0
$true/false$	\rightsquigarrow push 0/1
$\text{if } e \{e_1\} \{e_2\}$	\rightsquigarrow $e^+; \text{if0 } (e_1^+) (e_2^+)$
n	\rightsquigarrow push n
$e_1 < / = / + e_2$	\rightsquigarrow $e_1^+; e_2^+; \text{less?/equal?/add}$
x	\rightsquigarrow push x
$\text{inl } e$	\rightsquigarrow $e^+; \text{lam } x.(\text{push } [0, x])$
$\text{inr } e$	\rightsquigarrow $e^+; \text{lam } x.(\text{push } [1, x])$
$\text{match } e \{e_1\} y\{e_2\}$	\rightsquigarrow $e^+; \text{DUP}; \text{push } 1; \text{idx}; \text{SWAP};$ $\text{push } 0; \text{idx}; \text{if0 } (\text{lam } x.e_1^+) (\text{lam } y.e_2^+)$
$\text{fold } e$	\rightsquigarrow e^+
$\text{unfold } e$	\rightsquigarrow $e^+; \text{noop}$
(e_1, e_2)	\rightsquigarrow $e_1^+; e_2^+; \text{lam } x_2. \text{lam } x_1. (\text{push } [x_1, x_2])$
$\text{fst/snd } e$	\rightsquigarrow $e^+; \text{push } 0/1; \text{idx}$
$\text{fun } f(x_1 : \tau_1, \dots, x_n : \tau_n) \{e\}$	\rightsquigarrow $\text{push } (\text{thunk } \text{push } (\text{thunk } \text{lam } f. \text{lam } x_n. \dots. \text{lam } x_1. e^+), \text{fix})$
$e(e_1, \dots, e_n)$	\rightsquigarrow $e^+; e_1^+; \text{SWAP}; e_2^+; \text{SWAP} \dots; e_n^+; \text{SWAP}; \text{call}$
$\text{SWAP} \triangleq \text{lam } x. \text{lam } y. (\text{push } x; \text{push } y)$	
$\text{DROP} \triangleq \text{lam } x. ()$	
$\text{DUP} \triangleq \text{lam } x. (\text{push } x; \text{push } x)$	

Figure 9.4: Compiler from [FunLang](#) to StackLang

but because [FunLang](#) was deliberately designed without their behavior in mind, any [FunLang](#) types we assign to them would necessarily be imprecise! Thus, our type system—and, crucially, our soundness proof—has no way to accurately account for them.

A PRINCIPLED APPROACH One might be tempted to simply *approximate* foreign behavior with existing types; e.g., $\text{ref } \tau \sim \text{int}$, $\text{alloc} : (\tau) \rightarrow \text{int}$, $\text{read} : (\text{int}) \rightarrow \tau$, $\text{write} : (\text{int}, \tau) \rightarrow \text{unit}$. Indeed, this is what FFIs typically do. The problem with this approach is that it changes the type based reasoning of the language: for example, a [FunLang](#) programmer may (rightly) think that duplicate calls with the same input to a function of type $(\text{int}) \rightarrow \tau$ could be eliminated, since [FunLang](#) functions always return the same output, but clearly if that function is `read` (or any code that invokes it, or other stateful code internally), this is not necessarily true.

Instead, our framework starts by giving these foreign functions *precise* types, with which we can then work backwards through the implementation and the soundness proof. Of course, our approach is systematic, and applies to a wide variety of features, not just state.

The first step, and core idea, is to define an extended type system with new *linking types* that can describe foreign behavior. In Figure 9.5, we present the extension for state, where new features are typeset in **pink bold font**

$$\begin{array}{ll}
\text{Core Type } \tau & := \alpha \mid \text{unit} \mid \text{bool} \mid \text{int} \mid \tau \times \tau \mid \tau + \tau \\
& \mid \mu \alpha. \tau \mid (\tau, \dots, \tau) \rightarrow \tau \\
\text{Extended Type } \tau & := \text{unit} \mid \text{bool} \mid \text{int} \mid \tau \times \tau \mid \tau + \tau \mid \mu \alpha. \tau \\
& \mid (\tau, \dots, \tau) \xrightarrow{\bullet} \tau \mid \text{ref } \tau
\end{array}$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash_S x : \tau}$$

$$\frac{\Gamma, f : (\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau', \bar{x}_i : \bar{\tau}_i \vdash_S e : \tau'}{\Gamma \vdash_S \text{fun } f(x_1 : \tau_1, \dots, x_n : \tau_n)\{e\} : (\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau'}$$

$$\frac{\Gamma \vdash_S e : (\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau' \quad \Gamma \vdash_S \bar{e}_i : \bar{\tau}_i}{\Gamma \vdash_S e(e_1, \dots, e_n) : \tau'}$$

Figure 9.5: Linking types for state

As noted in §8.0.1,
it would be
sufficient to only
have $\xrightarrow{\circ}$ exist in
our model, but the
syntax is
convenient for our
proofs.

If we did not have $\xrightarrow{\circ}$, we would lift to $\xrightarrow{\bullet}$, which, while possible, would change the soundness lemma we would need to prove: rather than an isomorphism, we would have an implication $([\![\rightarrow]\!] \implies [\![\xrightarrow{\bullet}]\!])$ following the syntax with the reverse only stated in terms of the model, which still needs a notion of $\xrightarrow{\circ}$.

and we use the identifier **S** on typing judgments and elsewhere. We add a reference type, **ref** τ , without any introduction or elimination forms, since only foreign code can manipulate references. We also replace our function type with a pair of *modal* arrows, where $\xrightarrow{\circ}$ types pure functions and $\xrightarrow{\bullet}$ types stateful functions. We use $\xrightarrow{\bullet}$ when the particular mode is unimportant.

With this linking types extension, we can specify a more precise FFI for a mutable reference library, which we can then **import**. Since **FunLang** does not have polymorphism, we have to pick a concrete type τ when we import them:

```

import( alloc : () →• ref τ,
        read : (ref τ) →• τ,
        write : (ref τ, τ) →• unit )
...

```

Notice that the linking types extension is purely *static*; it does not introduce any new *term-level* syntax. So, intuitively, core programs should be usable inside of extended programs, and pure extended programs should be usable inside of core programs. To make this intuition precise, we use our type-level metafunctions, *lift* and *lower*. Lift, denoted $\uparrow \tau$, maps a core type to an extended type, while lower, denoted $\downarrow \tau$ maps an extended type to a core type. The definitions of lift and lower for the state extension are given in Figure 9.7. Note that core arrows \rightarrow are lifted to pure arrows $\xrightarrow{\circ}$ in the extension, which is why we did not include an introduction form for $\xrightarrow{\circ}$ in Figure 9.5: it suffices to build a core function and lift it! Since references cannot be used directly inside core code, lowering simply erases them. The only truly surprising case is that impure arrows are lowered to core arrows; this surely *seems* unsound, which we will address shortly.

To facilitate the interaction between core and extended programs, we rely on our *boundary* term, $\{e\}_{\downarrow \tau}^S$, that encapsulates foreign behavior from the rest of the core program (Figure 9.6). Here, **e** is a stateful program whose

$$\frac{I^+ \uplus \uparrow\Gamma \vdash_+ e : \tau}{I; \Gamma \vdash \{e\}_{\downarrow\tau}^+ : \downarrow\tau}$$

Figure 9.6: The boundary term over an arbitrary extension, $+$

$\uparrow\tau$	\triangleq	τ	$\downarrow\tau$	\triangleq	τ
$\uparrow\text{unit}$	\triangleq	unit	$\downarrow\text{unit}$	\triangleq	unit
$\uparrow\text{bool}$	\triangleq	bool	$\downarrow\text{bool}$	\triangleq	bool
$\uparrow\text{int}$	\triangleq	int	$\downarrow\text{int}$	\triangleq	int
$\uparrow\tau_1 \times \tau_2$	\triangleq	$\uparrow\tau_1 \times \uparrow\tau_2$	$\downarrow\tau_1 \times \tau_2$	\triangleq	$\downarrow\tau_1 \times \downarrow\tau_2$
$\uparrow\tau_1 + \tau_2$	\triangleq	$\uparrow\tau_1 + \uparrow\tau_2$	$\downarrow\tau_1 + \tau_2$	\triangleq	$\downarrow\tau_1 + \downarrow\tau_2$
$\uparrow\mu\alpha.\tau$	\triangleq	$\mu\alpha.\uparrow\tau$	$\downarrow\mu\alpha.\tau$	\triangleq	$\mu\alpha.\downarrow\tau$
$\uparrow(\tau_1, \dots, \tau_n) \rightarrow \tau'$	\triangleq	$(\uparrow\tau_1, \dots, \uparrow\tau_n) \xrightarrow{\circ} \uparrow\tau'$	$\downarrow(\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau'$	\triangleq	$(\downarrow\tau_1, \dots, \downarrow\tau_n) \rightarrow \downarrow\tau'$
			$\downarrow\text{ref } \tau$	\triangleq	unit

Figure 9.7: Lift and lower functions for state extension

result is a τ when typechecked with the extension S , which means it can be used at type $\downarrow\tau$ in the rest of the program. The boundary acts as a syntactic cue for the typechecker to switch to an extension before entering the body. To do so, the typechecker extracts imports relevant to the extension and lifts every binding from the typing context. This mechanism allows us to, for example, define top-level core functions and use them as pure functions under S boundaries. A well-formedness judgment would disallow different extensions from being mixed in a single import binding.

Next, for every extended type τ , we define a target-level *encapsulation wrapper*, $\langle \downarrow\tau \rangle$, which dynamically enforces that its argument *behave like a* $\downarrow\tau$. While we prove this formally in §9.5, here we give an intuitive explanation of the wrappers for the state extension, which are defined in Figure 9.8. Recall that only two types had non-obvious lower definitions: $\text{ref } \tau$ and \rightarrow^\bullet . The former we decided to erase, and the latter we suspiciously mapped to the pure arrow. The wrappers associated with these types reinforce these decisions

$$\begin{aligned} \langle \downarrow\text{ref } \tau \rangle &\triangleq \text{free; push 0} \\ \langle \downarrow(\tau_1, \dots, \tau_n) \rightarrow^\bullet \tau' \rangle &\triangleq \text{push (thunk lam l.push l; free); getlocs} \\ \langle \downarrow\tau \rangle &\triangleq . \quad \text{for any other } \tau \end{aligned}$$

$$\begin{aligned} \text{ALLOC} &\triangleq \text{thunk push (thunk lam falloc.lam f.push f; alloc); fix} \\ \text{READ} &\triangleq \text{thunk push (thunk lam fread.lam r.push r; read); fix} \\ \text{WRITE} &\triangleq \text{thunk push (thunk lam fwrite.lam f.lam r.push r; push f; write; push 0); fix} \end{aligned}$$

Figure 9.8: State boundary enforcement & target library code

because they give a strategy for encapsulating stateful behavior from the rest of the program. If a boundary returns a `ref` τ , we free its location in memory and return a semantic unit. If a boundary returns an impure function, we use `getlocs` to free any memory associated with that function. In both cases, purity is enforced by indirection: if the memory associated with these terms is used outside of the boundary (which corresponds to a side-effect), then a memory trap (`fail MEM`)² halts the program. All other linking types τ are semantically pure, so no wrapper is needed for them.

In Figure 9.8, we also provide a StackLang implementation of a mutable reference library. While we have not yet seen the tools required to prove that this implementation is safe (we will shortly in §9.5), intuitively, the functions behave as one would expect (N.B., they account for the calling convention of FunLang, as described in §9.3).

THE WHOLE PICTURE With all the pieces of the state extension in place, we return our attention to the `fastfib` example, defined in Figure 9.9. To start, we import our foreign functions, `alloc`, `read`, and `write`, specialized to $\text{int} \rightarrow \text{int}$ payloads so that we can store our memotable. In the body of `fastfib`, we immediately enter an **S** boundary so that we may use the foreign imports. We `alloc` an empty table, `mtbl`, initialized to return a sentinel value, `-1`, on any input. Next, we define the helper function `mutfib`, which does most of the heavy lifting. For any input `x` that is not a base case, `mutfib` begins by checking if `x` is in `mtbl`. If it is, it returns the result. Otherwise, it computes a fresh output and stores it in `mtbl` before returning it. At the top-level, `mutfib` is invoked on the input to `fastfib`. Note that whereas memoized functions can sometimes see speed-ups across top-level calls, here, different top-level calls to `fastfib` are encapsulated from one another; the memotable is dropped after each result. Naturally, one could write a version of `fastfib` over batches of inputs.

In this example, we see how we can use mutable state via our libraries and encapsulate the results. Even though we are confident that this particular implementation of `fastfib` does not try to misuse state (e.g., by exfiltrating the memotable across the boundary), buggy programs are inevitable. Therefore, it is essential that we wrap (the body of) `fastfib` in a boundary, so that runtime wrappers enforce the safe usage of state.

9.5 SOUNDNESS

As described in Chapter 8, we verify semantic type soundness using realizability models, which are sets of target terms indexed by source types. Unlike in the realizability models in Part II, here we first build a core model

² We consider memory traps an acceptable error in our definition of soundness.

```

import( alloc : ((int) → int) → ref ((int) → int),
        read : (ref ((int) → int)) → ((int) → int),
        write : (ref ((int) → int), ((int) → int)) → unit)
fun fastfib(y : int){
  let mtbl = alloc(fun f(n : int){-1} in
    fun mutfib(x : int){
      if x = 0 {0}{if x = 1{1}{}
      let m = read(mtbl) in
      if m(x) = -1{
        let r = mutfib(x + -1) + mutfib(x + -2) in
        let _ = write(mtbl, fun f(n){if n = x{r}{m(x)}}) in
        r
      }{m(x)}
    }
  ){(y)}Sint
}

```

Figure 9.9: Example: fibonacci memoized with state

for **FunLang**, and then build a model for our linking types extension. Our model for **FunLang** will be re-used in the next chapter, where we consider an extension that involves exceptions, since it does not have anything to do with state.

Since realizability models are inhabited by *target* terms, we can interpret linking types (e.g., **ref** τ) whose behavior is inexpressible in core **FunLang**. Also, the fact that the inhabitants of these models share a common operational semantics will be instrumental in proving that the boundary typing rule is sound.

9.5.1 **FunLang** model

Building on the simplified model from Chapter 8 (and prior models presented in this dissertation) we present the full model for core **FunLang** in Fig. 9.10. To distinguish the core and extension models from one another, we annotate each with an identifier; e.g., λ for core **FunLang**. To account for recursive types, the model is *step-indexed*, which means that every inhabitant is actually a pair of a natural number (the step index) and a term. Oftentimes, when the step index is unimportant, we refer only to the term.

We begin with the value relation, $\mathcal{V}^\lambda[\tau]$. Base types are agnostic to step indices, so their interpretation should simply be consistent with the compiler. Notice that $\mathcal{V}^\lambda[\text{bool}]$ is more liberal than the compiler, which only uses 0 and 1 for **bools**, but it is consistent with the StackLang eliminator if0. $\mathcal{V}^\lambda[\tau_1 \times \tau_2]$ contains all two-element arrays whose first element is in $\mathcal{V}^\lambda[\tau_1]$ and whose second element is in $\mathcal{V}^\lambda[\tau_2]$. $\mathcal{V}^\lambda[\tau_1 + \tau_2]$ contains all two-element

$$\begin{aligned}
\mathcal{V}^\lambda[\text{unit}] &= \{(k, 0)\} \\
\mathcal{V}^\lambda[\text{bool}] &= \{(k, n)\} \\
\mathcal{V}^\lambda[\text{int}] &= \{(k, n)\} \\
\mathcal{V}^\lambda[\tau_1 \times \tau_2] &= \{(k, [v_1, v_2]) \mid (k, v_1) \in \mathcal{V}^\lambda[\tau_1] \wedge (k, v_2) \in \mathcal{V}^\lambda[\tau_2]\} \\
\mathcal{V}^\lambda[\tau_1 + \tau_2] &= \{(k, [0, v]) \mid (k, v) \in \mathcal{V}^\lambda[\tau_1]\} \\
&\quad \cup \{(k, [1, v]) \mid (k, v) \in \mathcal{V}^\lambda[\tau_2]\} \\
\mathcal{V}^\lambda[\mu\alpha.\tau] &= \{(k, v) \mid \forall j < k. (j, v) \in \mathcal{V}^\lambda[\tau[\mu\alpha.\tau/\alpha]]\} \\
\mathcal{V}^\lambda[(\tau_1, \dots, \tau_n) \rightarrow \tau'] &= \{(k, \text{thunk push (thunk lam f.lam } x_n. \dots. \text{ lam } x_1.P; \text{fix}) \mid \\
&\quad \forall v_i \ k' < k. \ (k', v_i) \in \mathcal{V}^\lambda[\tau_i] \implies \\
&\quad (k', [x_1 \mapsto v_1, \dots, x_n \mapsto v_n, \\
&\quad f \mapsto (\text{thunk push (thunk lam f.lam } x_n. \dots. \\
&\quad \text{lam } x_1.P); \text{fix})]P) \in \mathcal{E}^\lambda[\tau']\} \\
\mathcal{E}^\lambda[\tau] &= \{(k, P) \mid \forall H, H', S, S', j < k. \langle H ; S ; P \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle \\
&\implies (S' = \text{Fail } c \wedge c \in \text{OKERR}) \vee \exists v. (S' = S, v \wedge (k - j, v) \in \mathcal{V}^\lambda[\tau])\} \\
&\text{where OKERR} \triangleq \{\text{MEM}\} \\
\mathcal{G}^\lambda[\cdot] &= \{(k, \cdot)\} \\
\mathcal{G}^\lambda[\Gamma, x : \tau] &= \{(k, \gamma[x \mapsto v]) \mid (k, v) \in \mathcal{V}^\lambda[\tau] \wedge (k, \gamma) \in \mathcal{G}^\lambda[\Gamma]\}
\end{aligned}$$

Figure 9.10: *FunLang* logical relation

arrays whose first element is a tag $n \in \{0, 1\}$ and whose second element is in $\mathcal{V}^\lambda[\tau_{n+1}]$. $\mathcal{V}^\lambda[\mu\alpha.\tau]$ motivates the use of step indices: naturally, any of its values should also be in the interpretation of the unfolding, $\mathcal{V}^\lambda[\tau[\mu\alpha.\tau/\alpha]]$, but because this is a potentially larger type, a model defined inductively over types alone would not be well-founded. Thus, we decrement the step index before unfolding the type. As in Chapter 8, $\mathcal{V}^\lambda[(\tau_1, \dots, \tau_n) \rightarrow \tau']$ contains all thunks that map well-typed inputs to well-typed outputs. Here, we additionally consider multiple arguments and recursion when selecting and substituting inputs, which we only draw from smaller step indices.

The expression relation, $\mathcal{E}^\lambda[\tau]$, contains pairs (k, P) of step indices and *StackLang computations*. Whereas in Chapter 8, we did not restrict the running time of P , here, we run it for fewer than k steps. Otherwise, the definition is exactly the same as in Chapter 8: P behaves like a τ if, given an *arbitrary* heap and stack, it either (i) runs too long; or (ii) halts with an error that our notion of soundness accepts; or (iii) terminates at a value in $\mathcal{V}^\lambda[\tau]$ at the top of its stack. Since $\mathcal{E}^\lambda[\tau]$ contains only closed computations, we also interpret contexts in $\mathcal{G}^\lambda[\Gamma]$, which contain all closing substitutions γ that map the bindings $x : \tau \in \Gamma$ to well-typed values in $\mathcal{V}^\lambda[\tau]$.

9.5.2 State extension model

We present preliminary definitions used by both our logical relation for the state linking types extension and our exception extension relation in Figure 9.11. The models share much with common models for mutable state (Ahmed, 2004), and thus include a Kripke world W that is made up of a step index k and heap typing Ψ . Heap typings map locations to type interpretations, drawn from the set of valid type interpretations Typ , or the sentinel value \dagger that indicates that the location has been freed. Our type interpretations consist of tuples (W, φ, v) of worlds W , sets of *relevant* locations φ , and target values v . We track relevant locations of v (i.e., the free locations of v or $flocs(v)$), rather than just relying on the location information that is in the heap typing, so that we can know if a value does not close over any locations and is thus extensionally pure. This pattern of tracking locations should look familiar from logical relations for linear state (e.g., (Ahmed et al., 2007)); here the difference will show up in how we maintain these sets.

We define the normal restrictions on step indices on relations and heap typings, and use these to define a later (\triangleright) operator. Next, we define how worlds can evolve with the \sqsubseteq operator: future worlds can have lower step index, and the heap typings must be preserved (updated to lower step index) or marked as dead. Lastly, we define when a heap H , which is a mapping from locations ℓ to values v , satisfies a world W under a relevant location set φ : for any locations in the heap typing that are relevant, if they are not dead, the value must be in the relation specified by the heap typing, but at a future world (to avoid circularity and reflect that it takes a step to retrieve a value from the heap).

With these definitions in mind, we can define the value and expression relations for our state linking types extension in Figure 9.12. We identify relations in this model with the superscript S . On base values, the relation is similar to the relation for `FunLang`, though it has to include both the relevant location sets (which are of course empty, as base values cannot close over locations) and arbitrary worlds W rather than simply step indices k . For pair types $\tau_1 \times \tau_2$, we appeal to the value relation on the two component types, but now have to deal with the location sets: the relevant set of locations for a pair is the union of the relevant locations for the two components. This is, of course, how this model differs from a linear one: in both, we track the locations that are specific to a term, but in a linear model, we would have a disjoint union, whereas in this model, it is fine if the same location is used in both elements of the pair. Sums and recursive types are analogous to the relation to `FunLang`. For reference types `ref` τ , we use the standard technique by choosing the relation that is stored in the

$$\begin{aligned}
\varphi ::= & \{\ell, \dots\} \\
\Psi \cap \varphi &\triangleq \{\ell \mapsto R \mid \ell \in (\text{dom}(\Psi) \cap \varphi) \wedge R = \Psi(\ell)\} \\
AtomVal_n &\triangleq \{(W, \varphi, v) \mid W \in World_n\} \\
HeapTy_n &\triangleq \{\Psi \mid \forall \ell \in \text{dom}(\Psi). \Psi(\ell) = \dagger \vee \Psi(\ell) \in Typ_n\} \\
World_n &\triangleq \{(k, \Psi) \mid k < n \wedge \Psi \in HeapTy_k\} \\
Typ_n &\triangleq \{R \in 2^{AtomVal_n} \mid \forall (W, \varphi, v) \in R. \forall W'. W \sqsubseteq W' \implies (W', \varphi, v) \in R\} \\
World &\triangleq \bigcup_n World_n \\
Typ &\triangleq \bigcup_n Typ_n \\
[R]_j &\triangleq \{(W, \varphi, v) \mid (W, \varphi, v) \in R \wedge W.k < j\} \\
[\Psi]_j &\triangleq \{\ell \mapsto [R]_j \mid \ell \mapsto R \in \Psi\} \\
(k, \Psi) \sqsubseteq (j, \Psi') &\triangleq j \leq k \wedge \forall \ell \in \text{dom}(\Psi). ([\Psi(\ell)]_j = [\Psi'(\ell)]_j \vee \Psi'(\ell) = \dagger) \\
W_1 \sqsubset W_2 &\triangleq W_1.k > W_2.k \wedge W_1 \sqsubseteq W_2 \\
\triangleright (k, \Psi) &\triangleq (k - 1, [\Psi]_{k-1}) \\
H :_\varphi W &\triangleq (\forall \ell \mapsto R \in (W.\Psi \cap \varphi). (\triangleright W, H(\ell)) \in R)
\end{aligned}$$

Figure 9.11: State & exception extension logical relation: preliminary definitions

heap typing at the given location, but additionally require that it is the only relevant location, as clearly a location value has only one relevant location.

We split functions into two cases: ones that can own locations and ones that are (extensionally) pure. The latter are written \circlearrowleft , and mean that there are no relevant locations. Locations can, of course, come from the arguments passed to the function in the sets φ_i , as a function that does not own locations can be passed a value that does, provided its type allows it. The resulting expression thus owns the locations of the arguments unioned together in $\bigcup_i \varphi_i$, and this needs to be in the expression relation: this means that if the return type is a value that does not own locations, the ones from the arguments must not be used in that return value. The function type that can close over locations, $\bullet\circlearrowleft$, is very similar: the only difference is that its relevant location set φ is included in the set that the body runs with.

The expression relation is similar to that of [FunLang](#), though refined for the fact that we now have constraints on what the target heap can look like. In particular, our initial heap H must satisfy the world W under the relevant location set φ , and there must be a final world $W' \sqsupseteq W$ that the final heap satisfies. We require that locations relevant to the term, φ , and relevant to the final value φ' , both be accounted for: what this means is that these locations must have their types preserved or be freed, but cannot be changed to a different type. Essentially, this means that everything we started with must be accounted for, and anything that is relevant to the value must be in the heap at the right type.

As before, we have an environment relation $\mathcal{G}^S[\Gamma]$ that we use to describe closing substitutions that satisfy an environment Γ . The substitutions now have a relevant set of locations that are a union of all the locations relevant to all the values in the substitution.

Finally, we define a shorthand for describing open terms, which includes choose closing substitutions to match the imports. Since imports could come from either the extension presented in this chapter or that in the next, we close with two substitutions. Note that I^S means I filtered to those imports that use types from the state linking types extension.

9.5.3 Proving \uparrow sound

We need to prove:

Lemma 9.5.1 (lift S). $\forall W v. (W, \emptyset, v) \in \mathcal{V}^S[\uparrow\tau] \iff (W.k, v) \in \mathcal{V}^\lambda[\tau]$

Proof. We note, first, that by inspection of the logical relation, all cases of $\mathcal{V}^S[\tau]$ for $\tau = \uparrow\tau$, $\varphi = \emptyset$. That is obviously critically important, as we wouldn't otherwise be able to account for such locations when moving to the relation for [FunLang](#), but it is also part of the design of the type system and the functions \uparrow . This justifies the use of \emptyset in the lemma statements.

$$\begin{aligned}
\mathcal{V}^S[\text{unit}] &= \{(W, \emptyset, 0)\} \\
\mathcal{V}^S[\text{bool}] &= \{(W, \emptyset, \mathsf{n})\} \\
\mathcal{V}^S[\text{int}] &= \{(W, \emptyset, \mathsf{n})\} \\
\mathcal{V}^S[\tau_1 \times \tau_2] &= \{(W, \varphi, [v_1, v_2]) \mid \varphi \subset \text{dom}(W.\Psi) \wedge \varphi_1 \cup \varphi_2 = \varphi \wedge \\
&\quad (W, \varphi_1, v_1) \in \mathcal{V}^S[\tau_1] \wedge (W, \varphi_2, v_2) \in \mathcal{V}^S[\tau_2]\} \\
\mathcal{V}^S[\tau_1 + \tau_2] &= \{(W, \varphi, [0, v]) \mid \varphi \subset \text{dom}(W.\Psi) \wedge (W, \varphi, v) \in \mathcal{V}^S[\tau_1]\} \\
&\cup \{(W, \varphi, [1, v]) \mid \varphi \subset \text{dom}(W.\Psi) \wedge (W, \varphi, v) \in \mathcal{V}^S[\tau_2]\} \\
\mathcal{V}^S[\mu\alpha.\tau] &= \{(W, \varphi, v) \mid (W, \varphi, v) \in \triangleright \mathcal{V}^S[\tau[\mu\alpha.\tau/\alpha]]\} \\
\mathcal{V}^S[\text{ref } \tau] &= \{(W, \{\ell\}, \ell) \mid W.\Psi(\ell) = [\mathcal{V}^S[\tau]]_{W.k} \mid \dagger\} \\
\mathcal{V}^S[(\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau'] &= \{(W, \emptyset, \text{thunk push (thunk lam f.lam } x_n. \dots \text{ lam } x_1.P); \text{fix}) \mid \\
&\quad \forall v_i \varphi_i W' \sqsupseteq W. \varphi_i \subset \text{dom}(W'.\Psi) \wedge (W', \varphi_i, v_i) \in \mathcal{V}^S[\tau_i] \\
&\quad \implies (W', \bigcup_i \varphi_i, [x_1 \mapsto v_1, \dots, x_n \mapsto v_n, \\
&\quad \quad f \mapsto (\text{thunk push (thunk lam f.lam } x_n. \\
&\quad \quad \dots \text{ lam } x_1.P); \text{fix})]P) \in \mathcal{E}^S[\tau']\} \\
\mathcal{V}^S[(\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau'] &= \{(W, \varphi, \text{thunk push (thunk lam f.lam } x_n. \dots \text{ lam } x_1.P); \text{fix}) \mid \\
&\quad \varphi \subset \text{dom}(W.\Psi) \wedge \forall v_i \varphi_i W' \sqsupseteq W. \varphi_i \subset \text{dom}(W'.\Psi) \\
&\quad \wedge (W', \varphi_i, v_i) \in \mathcal{V}^S[\tau_i] \\
&\quad \implies (W', \varphi \cup \bigcup_i \varphi_i, [x_1 \mapsto v_1, \dots, x_n \mapsto v_n, \\
&\quad \quad f \mapsto (\text{thunk push (thunk lam f.lam } x_n. \\
&\quad \quad \dots \text{ lam } x_1.P); \text{fix})]P) \in \mathcal{E}^S[\tau']\}
\end{aligned}$$

$\mathcal{E}^S[\tau] = \{(W, \varphi, P) \mid \forall H: \varphi W, S, H', S', j < W.k. \langle H ; S ; P \rangle \xrightarrow{*} j \langle H' ; S' ; \cdot \rangle\}$
 $\implies (S' = \text{Fail } c \wedge c \in \text{OKERR}) \vee \exists v, W' \sqsupseteq W.$
 $(S' = S, v \wedge H' :_{\varphi' \cup \varphi} W' \wedge (W', \varphi', v) \in \mathcal{V}^S[\tau])\}$

where $\text{OKERR} \triangleq \{\text{MEM}\}$

$$\begin{aligned}
\mathcal{G}^S[\cdot] &= \{(W, \emptyset, \cdot) \mid W \in \text{World}\} \\
\mathcal{G}^S[\Gamma, x : \tau] &= \{(W, \varphi_1 \cup \varphi_2, \gamma[x \mapsto v]) \mid \varphi_i \subset \text{dom}(W.\Psi) \\
&\quad \wedge (W, \varphi_1, v) \in \mathcal{V}^S[\tau] \wedge (W, \varphi_2, \gamma) \in \mathcal{G}^S[\Gamma]\}
\end{aligned}$$

$$\begin{aligned}
\llbracket I; \Gamma \vdash P : \tau \rrbracket &\equiv \\
\forall k \gamma. \forall ((k, \emptyset), \emptyset, \gamma^{I^S}) \in \mathcal{G}^S[I^S]. \forall ((k, \emptyset), \emptyset, \gamma^{I^X}) \in \mathcal{G}^X[I^X]. (k, \gamma) \in \mathcal{G}^\lambda[\Gamma] &\implies \\
(k, \gamma^{I^X}(\gamma^{I^S}(\gamma(P)))) \in \mathcal{E}^\lambda[\tau]
\end{aligned}$$

Figure 9.12: State extension logical relation: main definition

The proof itself then follows via induction over the step index and structure of the type, since the subset of the state relation that we are considering maps directly to the `FunLang` relation, by design:

Case `unit/bool/int`. In this case, the values are trivially in the relation, by definition.

Case $\tau_1 \times \tau_2 / \tau_1 + \tau_2$. These follow straightforwardly by appealing to the inductive hypothesis.

Case $\mu\alpha.\tau$. In this case, we can appeal to our inductive hypothesis at a smaller k (as our type may have gotten larger).

Case $(\tau_1, \dots, \tau_n) \rightarrow \tau'$. This follows by application of the induction hypothesis.

□

9.5.4 Proving $\langle \downarrow \tau \rangle$ satisfies \downarrow

First, we prove two lemmas:

Lemma 9.5.2 (wrap closed **S**). $\forall \tau. fvars(\langle \downarrow \tau \rangle) = \emptyset$

Proof. This follows by simple inspection of the definition. □

Lemma 9.5.3 (encapsulation **S**). $\forall W \varphi v \tau. (W, \varphi, v) \in \mathcal{V}^S[\tau] \implies (W, \varphi, \text{push } v; \langle \downarrow \tau \rangle) \in \mathcal{E}^S[\uparrow \downarrow \tau]$

Proof. We proceed by case analysis on τ , handling the majority of the cases for which $\langle \downarrow \tau \rangle$ is empty first. In those cases, which by inspection, $\uparrow \downarrow \tau = \tau$, the proof reduces to:

$$\forall W \varphi v \tau. (W, \varphi, v) \in \mathcal{V}^S[\tau] \implies (W, \varphi, \text{push } v) \in \mathcal{E}^S[\tau]$$

This follows easily: we choose an arbitrary stack and a heap that satisfies W and φ , we take a single step (if no budget, in relation trivially), and result in terminated program with stack with v on top. As needed, $(W, \varphi, v) \in \mathcal{V}^S[\tau]$, so we are done. Now we handle the other two cases:

Case `ref` τ . Our obligation is to show:

$$\forall W \varphi v \tau. (W, \varphi, v) \in \mathcal{V}^S[\text{ref } \tau] \implies (W, \varphi, \text{push } v; \text{free}; \text{push } 0) \in \mathcal{E}^S[\text{unit}]$$

By inspection of $\mathcal{V}^S[\text{ref } \tau]$, we know for some ℓ , $\varphi = \{\ell\}$, $v = \ell$, and $W.\Psi(\ell) = [\mathcal{V}^S[\tau]]_{W.k}$. This means when we choose a heap H to run

with in $\mathcal{E}^S[\text{unit}]$, we know it will have ℓ bound to some value in $\triangleright \mathcal{V}^S[\tau]_{W,k}$, though as we will see, the actual value does not matter. We will then take three steps:

$$\begin{aligned}\langle H ; S ; \text{push } \ell; \text{free}; \text{push } 0 \rangle &\rightarrow \\ \langle H ; S, \ell ; \text{free}; \text{push } 0 \rangle &\rightarrow \\ \langle H \setminus \ell ; S ; \text{push } 0 \rangle &\rightarrow \langle H \setminus \ell ; S, 0 ; \cdot \rangle\end{aligned}$$

Now we choose W' to be W , but with ℓ updated to be marked as dead, and choose $\varphi' = \emptyset$. This means $(H \setminus \ell) :_\varphi W'$ (since the dead binding in the world is ignored), and by definition, $(W', \emptyset, 0) \in \mathcal{V}^S[\text{unit}]$, so we are done with this case.

Case $(\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau'$. Our obligation is to show:

$$\begin{aligned}\forall W \varphi \vee \tau_1 \tau'. (W, \varphi, v) \in \mathcal{V}^S[(\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau'] \implies \\ (W, \varphi, \text{push } v; \text{lam } x.(\text{push } x; \text{push } x); \\ \text{push } (\text{thunk lam } l. \text{push } l; \text{free}); \text{getlocs}) \in \mathcal{E}^S[(\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau']\end{aligned}$$

Once we pick an arbitrary stack S and a heap $H :_\varphi W$, we take the following steps:

$$\begin{aligned}\langle H ; S ; \text{push } v; \text{lam } x.(\text{push } x; \text{push } x); \text{push } (\text{thunk lam } l. \text{push } l; \text{free}); \text{getlocs} \rangle &\rightarrow \\ \langle H ; S, v ; \text{lam } x.(\text{push } x; \text{push } x); \text{push } (\text{thunk lam } l. \text{push } l; \text{free}); \text{getlocs} \rangle &\xrightarrow{3} \\ \langle H ; S, v, v ; \text{push } (\text{thunk lam } l. \text{push } l; \text{free}); \text{getlocs} \rangle &\rightarrow \\ \langle H ; S, v, v, (\text{thunk lam } l. \text{push } l; \text{free}) ; \text{getlocs} \rangle\end{aligned}$$

Now, we know that getlocs will run the thunk on top of the stack once for every free location one position down the stack, which means everything reachable from our function value. Assume those locations are ℓ_1, \dots, ℓ_k . Then we step as follows:

$$\begin{aligned}\langle H ; S, v, v, (\text{thunk lam } l. \text{push } l; \text{free}) ; \text{getlocs} \rangle &\xrightarrow{3k+1} \\ \langle H \setminus \{\ell_1, \dots, \ell_k\} ; S, v ; \cdot \rangle\end{aligned}$$

Now that we have terminated, we have to fulfill the obligations of $\mathcal{E}^S[(\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau']$. By inspection of $\mathcal{V}^S[(\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau']$, we know that v has form $(\text{thunk push } (\text{thunk lam } f. \text{lam } x_n. \dots. \text{lam } x_1. P); \text{fix})$. We choose $\varphi' = \emptyset$, and W' such that every location in φ has been marked dead. By invariant of the relation, $\varphi = \{\ell_1, \dots, \ell_k\}$. Our heap satisfies the world, by construction, and everything that should be dead is, so the only thing that remains is to show that

$$(W', \emptyset, (\text{thunk push } (\text{thunk lam } f.\text{lam } x_n \dots \text{ lam } x_1.P); \text{fix})) \in \mathcal{V}^S[\langle \tau_1, \dots, \tau_n \rangle \xrightarrow{\circ} \tau']$$

This follows from our hypothesis on ν , once we substitute our empty relevant location set in.

□

Now we proceed to the main lemma:

Lemma 9.5.4 (boundary S). $\llbracket I^S \uplus \uparrow \Gamma \vdash_S P : \tau \rrbracket \implies \llbracket I; \Gamma \vdash P; \langle \downarrow \tau \rangle : \downarrow \tau \rrbracket$

Unlike soundness for lift, this is non-trivial. To start with, the intuitive statement that, for $(W, \varphi, P) \in \mathcal{E}^S[\tau]$, show $(W.k, P; \langle \downarrow \tau \rangle) \in \mathcal{E}^\lambda[\downarrow \tau]$, isn't provable (or true): the problem is that P is a term which may involve locations in φ , and the relation $\mathcal{E}^\lambda[\downarrow \tau]$ for `FunLang` cannot reason about such state. Indeed, that relation specifically says you choose an arbitrary heap to run under, which clearly would get stuck if P tried to access a particular location. But, of course, $\downarrow \tau$ is a type from `FunLang`, so how do we prove this? At a high-level, this relies on both soundness of lift and the lemma proved above. The detailed proof follows.

Proof. Expanding the goal, we see we need to show:

$$\forall k \gamma. \forall ((k, \emptyset), \emptyset, \gamma^{I^S}) \in \mathcal{G}^S[\llbracket I^S \rrbracket]. \forall ((k, \emptyset), \emptyset, \gamma^{I^X}) \in \mathcal{G}^X[\llbracket I^X \rrbracket]. (k, \gamma) \in \mathcal{G}^\lambda[\llbracket \Gamma \rrbracket] \implies \\ (k, \gamma^{I^X}(\gamma^{I^S}(\gamma(P; \langle \downarrow \tau \rangle)))) \in \mathcal{E}^\lambda[\llbracket \downarrow \tau \rrbracket]$$

From Lemma 9.5.2, we know $\langle \downarrow \tau \rangle$ is closed, so we can push the substitutions in to just over P . Further, from the hypothesis, we know that P has no free variables from I^X , so we can eliminate that substitution.

The hypothesis that we are working with says:

$$\forall W \varphi \gamma' (W, \varphi, \gamma') \in \mathcal{G}^S[\llbracket I^S \uplus \uparrow \Gamma \rrbracket] \wedge \varphi = \text{flocs}(\gamma(P)) \implies (k, \varphi, \gamma'(P)) \in \mathcal{E}^S[\llbracket \tau \rrbracket]$$

To instantiate the hypothesis, we need an environment γ' that satisfies $\mathcal{G}^S[\llbracket I^S \uplus \uparrow \Gamma \rrbracket]$. We argue that it is exactly γ composed with γ^{I^S} : we know they are disjoint, and we know the former can be lifted into the latter via Lemma 9.5.1. This means, in particular, that φ is \emptyset .

Since we have no relevant locations, any heap will satisfy the expression relation: in particular, the arbitrary H that we have to consider for our obligation, and we can similarly use the arbitrary stack S . This means that our hypothesis tells us that:

$$\langle H ; S ; (\gamma^{\text{I}^S}(\gamma(P))) \rangle \xrightarrow{*} \langle H' ; S' ; \cdot \rangle$$

Unless we run beyond our step budget, in which case we are trivially in the relation. Similarly, if we run to Fail c, we are also in our relation. Otherwise, we know that $S' = S, v$ and, for a future world $W' \sqsubseteq W$ that H' satisfies with the relevant locations φ' , $(W', \varphi', v) \in \mathcal{V}^S[\tau]$.

Now, what we want to show is that this value is “contained” by the code in $\langle \downarrow \tau \rangle$ to behave like $\downarrow \tau$. But, clearly we can’t show that using the $\mathcal{E}^\lambda[\tau]$ logical relation, as the value still can have locations it is closing over, etc. So, we proceed by two steps. First, we appeal to Lemma 9.5.3

This will tell us that we can evaluate the whole program at question further, to get to a point with a world $W'' \sqsubseteq W'$, $\varphi'', H'' :_{\varphi'' \cup \varphi'} W''$ and $(W'', \varphi'', v') \in \mathcal{V}^S[\uparrow \downarrow \tau]$:

$$\begin{aligned} & \langle H ; S ; (\gamma^{\text{I}^S}(\gamma(P)); \langle \downarrow \tau \rangle) \rangle \xrightarrow{*} \\ & \langle H' ; S, v ; \langle \downarrow \tau \rangle \rangle \xrightarrow{*} \\ & \langle H'' ; S, v' ; \cdot \rangle \end{aligned}$$

Now, we appeal to Lemma 9.5.1

This means that the value that we ran down to is in $(W''.k, v') \in \mathcal{V}^\lambda[\downarrow \tau]$, which is exactly what we need to show.

□

9.5.5 Proving compatibility lemmas

We state the lemmas here, though defer the proofs to Appendix D, as they are quite mechanical; the only interesting one is the last one (the boundary rule), and that is exactly Lemma 9.5.4, that we just proved. We first prove compatibility lemmas for our core language, `FunLang`, and then for our linking types extension. In the next chapter, we will have a new extension, but reuse the core language, and thus reuse the core language compatibility proofs. Since our logical relations are built out of *closed* terms, we need to pick substitutions for our `imports`. Our imports are terms written using the linking types extensions, and thus our proofs for `FunLang` involve first closing with substitutions for *both* extensions: both the state extension covered in this chapter and the exception one (identified by X) covered in Chapter 10). The only place where this becomes material is in the boundary rule: but the typing rule that makes that relevant is only introduced by the extension (and thus we will show the lemma for the boundary for the state extension here, and leave the exception extension boundary rule for Chapter 10).

Lemma 9.5.5 (unit). *Show that $\llbracket I ; \Gamma \vdash \text{push } 0 : \text{unit} \rrbracket$.*

Proof. See D.0.4. \square

Lemma 9.5.6 (bool). *Show for any n , $\llbracket I; \Gamma \vdash \text{push } n : \text{bool} \rrbracket$.*

Proof. See D.0.5. \square

Lemma 9.5.7 (if). *If $\llbracket I; \Gamma \vdash P : \text{bool} \rrbracket$, $\llbracket I; \Gamma \vdash P_1 : \tau \rrbracket$, and $\llbracket I; \Gamma \vdash P_2 : \tau \rrbracket$ then*

$$\llbracket I; \Gamma \vdash P; \text{if0 } P_1 \ P_2 : \tau \rrbracket.$$

Proof. See D.0.6. \square

Lemma 9.5.8 (int). *For any n , show $\llbracket I; \Gamma \vdash \text{push } n : \text{int} \rrbracket$.*

Proof. See D.0.7. \square

Lemma 9.5.9 (op- $=$). *If $\llbracket I; \Gamma \vdash P_1 : \text{int} \rrbracket$ and $\llbracket I; \Gamma \vdash P_2 : \text{int} \rrbracket$, show that $\llbracket I; \Gamma \vdash P_1; P_2; \text{equal?} : \text{bool} \rrbracket$.*

Proof. See D.0.8. \square

Lemma 9.5.10 (op- $<$). *If $\llbracket I; \Gamma \vdash P_1 : \text{int} \rrbracket$ and $\llbracket I; \Gamma \vdash P_2 : \text{int} \rrbracket$, show that $\llbracket I; \Gamma \vdash P_1; P_2; \text{less?} : \text{bool} \rrbracket$.*

Proof. See D.0.9. \square

Lemma 9.5.11 (op- $+$). *If $\llbracket I; \Gamma \vdash P_1 : \text{int} \rrbracket$ and $\llbracket I; \Gamma \vdash P_2 : \text{int} \rrbracket$, show that $\llbracket I; \Gamma \vdash P_1; P_2; \text{add} : \text{int} \rrbracket$.*

Proof. See D.0.10. \square

Lemma 9.5.12 (var). *For any $x : \tau \in \Gamma$, show that $\llbracket I; \Gamma \vdash \text{push } x : \tau \rrbracket$.*

Proof. See D.0.11. \square

Lemma 9.5.13 (pair). *If $\llbracket I; \Gamma \vdash P_1 : \tau_1 \rrbracket$ and $\llbracket I; \Gamma \vdash P_2 : \tau_2 \rrbracket$, show that $\llbracket I; \Gamma \vdash P_1; P_2; \text{lam } x_2. \text{ lam } x_1. \text{push } [x_1, x_2] : \tau_1 \times \tau_2 \rrbracket$.*

Proof. See D.0.12. \square

Lemma 9.5.14 (fst). *If $\llbracket I; \Gamma \vdash P : \tau_1 \times \tau_2 \rrbracket$, show that $\llbracket I; \Gamma \vdash P_1; \text{push } 0; \text{idx} : \tau_1 \rrbracket$.*

Proof. See D.0.13. \square

Lemma 9.5.15 (snd). *If $\llbracket I; \Gamma \vdash P : \tau_1 \times \tau_2 \rrbracket$, show that $\llbracket I; \Gamma \vdash P_1; \text{push } 1; \text{idx} : \tau_2 \rrbracket$.*

Proof. See D.0.14. \square

Lemma 9.5.16 (inl). *If $\llbracket I; \Gamma \vdash P : \tau_1 \rrbracket$, show that $\llbracket I; \Gamma \vdash P; \text{lam } x. \text{push } [0, x] : \tau_1 + \tau_2 \rrbracket$.*

Proof. See D.0.15. \square

Lemma 9.5.17 (inr). *If $\llbracket I; \Gamma \vdash P : \tau_2 \rrbracket$, show that $\llbracket I; \Gamma \vdash P; \text{lam } x.\text{push } [1, x] : \tau_1 + \tau_2 \rrbracket$.*

Proof. See D.0.16. \square

Lemma 9.5.18 (match). *If $\llbracket I; \Gamma \vdash P : \tau_1 + \tau_2 \rrbracket$, $\llbracket I; \Gamma, x : \tau_1 \vdash P_1 : \tau \rrbracket$, and $\llbracket I; \Gamma, y : \tau_2 \vdash P_2 : \tau \rrbracket$, show that $\llbracket I; \Gamma \vdash P; \text{DUP}; \text{push } 1; \text{idx}; \text{SWAP}; \text{push } 0; \text{idx}; \text{if0 } (\text{lam } x.P_1) (\text{lam } y.P_2) : \tau \rrbracket$.*

Proof. See D.0.17. \square

Lemma 9.5.19 (fold). *If $\llbracket I; \Gamma \vdash P : \tau[\mu\alpha.\tau/\alpha] \rrbracket$, show that $\llbracket I; \Gamma \vdash P : \mu\alpha.\tau \rrbracket$.*

Proof. See D.0.18. \square

Lemma 9.5.20 (unfold). *If $\llbracket I; \Gamma \vdash P : \mu\alpha.\tau \rrbracket$, show that $\llbracket I; \Gamma \vdash P; \text{noop} : \tau[\mu\alpha.\tau/\alpha] \rrbracket$.*

Proof. See D.0.19. \square

Lemma 9.5.21 (fun). *If $\llbracket I; \Gamma, f : (\tau_1, \dots, \tau_n) \rightarrow \tau', x_i : \tau_i \vdash P : \tau' \rrbracket$, show that*

$\llbracket I; \Gamma \vdash \text{push } (\text{thunk push } (\text{thunk lam } f.\text{lam } x_n. \dots. \text{lam } x_1.P); \text{fix}) : (\tau_1, \dots, \tau_n) \rightarrow \tau' \rrbracket$

Proof. See D.0.20. \square

Lemma 9.5.22 (app). *If $\llbracket I; \Gamma \vdash P : (\tau_1, \dots, \tau_n) \rightarrow \tau' \rrbracket$ and for $i \in \{1, \dots, n\}$ $\llbracket I; \Gamma \vdash P_i : \tau_i \rrbracket$ then*

$\llbracket I; \Gamma \vdash P; P_1; \text{SWAP} \dots P_n; \text{SWAP}; \text{call} : \tau' \rrbracket$

Proof. See D.0.21. \square

Lemma 9.5.23 (boundary S). $\llbracket I^S \uplus \Gamma \vdash_S P : \tau \rrbracket \implies \llbracket I; \Gamma \vdash P; \langle \downarrow \tau \rangle : \downarrow \tau \rrbracket$

Proof. See D.0.22. \square

For the next set of lemmas, which cover the state extension type system, we use the following notation:

$$\llbracket \Gamma \vdash_S P : \tau \rrbracket \equiv \forall W \varphi \gamma. (W, \varphi, \gamma) \in \mathcal{G}^S[\Gamma] \implies (W, \text{flocs}(\gamma(P)), \gamma(P)) \in \mathcal{E}^S[\tau]$$

Lemma 9.5.24 (unit). *Show that $\llbracket \Gamma \vdash_S \text{push } 0 : \text{unit} \rrbracket$.*

Proof. See D.0.23. \square

Lemma 9.5.25 (bool). *Show for any n , $\llbracket \Gamma \vdash_S n : \text{bool} \rrbracket$.*

Proof. See D.0.24. \square

Lemma 9.5.26 (if). *If $\llbracket \Gamma \vdash_S P : \text{bool} \rrbracket$, $\llbracket \Gamma \vdash_S P_1 : \tau \rrbracket$, and $\llbracket \Gamma \vdash_S P_2 : \tau \rrbracket$ then*

$$\llbracket \Gamma \vdash_S P; \text{if0 } P_1 \ P_2 : \tau \rrbracket.$$

Proof. See D.0.25. \square

Lemma 9.5.27 (int). *For any n , show $\llbracket \Gamma \vdash_S \text{push } n : \text{int} \rrbracket$.*

Proof. See D.0.26. \square

Lemma 9.5.28 (op- $=$). *If $\llbracket \Gamma \vdash_S P_1 : \text{int} \rrbracket$ and $\llbracket \Gamma \vdash_S P_2 : \text{int} \rrbracket$, show that $\llbracket \Gamma \vdash_S P_1; P_2; \text{equal?} : \text{bool} \rrbracket$.*

Proof. See D.0.27. \square

Lemma 9.5.29 (op- $<$). *If $\llbracket \Gamma \vdash_S P_1 : \text{int} \rrbracket$ and $\llbracket \Gamma \vdash_S P_2 : \text{int} \rrbracket$, show that $\llbracket \Gamma \vdash_S P_1; P_2; \text{less?} : \text{bool} \rrbracket$.*

Proof. See D.0.28. \square

Lemma 9.5.30 (op- $+$). *If $\llbracket \Gamma \vdash_S P_1 : \text{int} \rrbracket$ and $\llbracket \Gamma \vdash_S P_2 : \text{int} \rrbracket$, show that $\llbracket \Gamma \vdash_S P_1; P_2; \text{add} : \text{int} \rrbracket$.*

Proof. See D.0.29. \square

Lemma 9.5.31 (var). *For any $x : \tau \in \Gamma$, show that $\llbracket \Gamma \vdash_S \text{push } x : \tau \rrbracket$.*

Proof. See D.0.30. \square

Lemma 9.5.32 (pair). *If $\llbracket \Gamma \vdash_S P_1 : \tau_1 \rrbracket$ and $\llbracket \Gamma \vdash_S P_2 : \tau_2 \rrbracket$ then $\llbracket \Gamma \vdash_S P_1; P_2; \text{lam } x_2.\text{lam } x_1.\text{push } [x_1, x_2] : \tau_1 \times \tau_2 \rrbracket$*

Proof. See D.0.31. \square

Lemma 9.5.33 (fst). *If $\llbracket \Gamma \vdash_S P : \tau_1 \times \tau_2 \rrbracket$, show that $\llbracket \Gamma \vdash_S P_1; \text{push } 0; \text{idx} : \tau_1 \rrbracket$.*

Proof. See D.0.32. \square

Lemma 9.5.34 (snd). *If $\llbracket \Gamma \vdash_S P : \tau_1 \times \tau_2 \rrbracket$, show that $\llbracket \Gamma \vdash_S P_1; \text{push } 1; \text{idx} : \tau_2 \rrbracket$.*

Proof. See D.0.33. \square

Lemma 9.5.35 (inl). *If $\llbracket \Gamma \vdash_S P : \tau_1 \rrbracket$, show that $\llbracket \Gamma \vdash_S P; \text{lam } x.\text{push } [0, x] : \tau_1 + \tau_2 \rrbracket$.*

Proof. See D.0.34. \square

Lemma 9.5.36 (inr). *If $\llbracket \Gamma \vdash_S P : \tau_2 \rrbracket$, show that $\llbracket \Gamma \vdash_S P; \text{lam } x.\text{push } [1, x] : \tau_1 + \tau_2 \rrbracket$.*

Proof. See D.0.35. □

Lemma 9.5.37 (match). *If $\llbracket \Gamma \vdash_S P : \tau_1 + \tau_2 \rrbracket$, $\llbracket \Gamma, x : \tau_1 \vdash_S P_1 : \tau \rrbracket$, and $\llbracket \Gamma, y : \tau_2 \vdash_S P_2 : \tau \rrbracket$, show that*

$$\llbracket \Gamma \vdash_S P; DUP; push 1; idx; SWAP; push 0; idx; if0 (\text{lam } x.P_1) (\text{lam } y.P_2) : \tau \rrbracket$$

Proof. See D.0.36. □

Lemma 9.5.38 (fold). *If $\llbracket \Gamma \vdash_S P : \tau[\mu\alpha.\tau/\alpha] \rrbracket$, show that $\llbracket \Gamma \vdash_S P : \mu\alpha.\tau \rrbracket$.*

Proof. See D.0.37. □

Lemma 9.5.39 (unfold). *If $\llbracket \Gamma \vdash_S P : \mu\alpha.\tau \rrbracket$, show that $\llbracket \Gamma \vdash_S P; noop : \tau[\mu\alpha.\tau] \rrbracket$.*

Proof. See D.0.38. □

Lemma 9.5.40 (fun). *If $\llbracket \Gamma, f : (\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau', x_i : \tau_i \vdash_S P : \tau' \rrbracket$, show that $\llbracket \Gamma \vdash_S P; push (\text{thunk push (thunk lam } f.\text{lam } x_n. \dots. \text{ lam } x_1.\gamma(P)); fix) : (\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau' \rrbracket$*

Proof. See D.0.39. □

Lemma 9.5.41 (app pure). *If $\llbracket \Gamma \vdash_S P : (\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau' \rrbracket$ and for $i \in \{1, \dots, n\}$ $\llbracket \Gamma \vdash_S P_i : \tau_i \rrbracket$ then*

$$\llbracket \Gamma \vdash_S P; P_1; SWAP \dots P_n; SWAP; call : \tau' \rrbracket$$

Proof. See D.0.40. □

Lemma 9.5.42 (app state). *If $\llbracket \Gamma \vdash_S P : (\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau' \rrbracket$ and for $i \in \{1, \dots, n\}$ $\llbracket \Gamma \vdash_S P_i : \tau_i \rrbracket$ then*

$$\llbracket \Gamma \vdash_S P; P_1; SWAP \dots P_n; SWAP; call : \tau' \rrbracket$$

Proof. See D.0.41. □

9.5.6 Proving libraries satisfy types

The next step we need to do is prove that the library code that we are linking with satisfies the types that we are importing it as. We note that a single library may have multiple types that it can be given—and may be usable from different extensions, which have different reasoning principles. This is most noticeable in our case because our source language and extensions lack polymorphism, while many of our library functions are naturally polymorphic: e.g., `ref τ`, not `ref int` or `ref bool`. However, when we prove the libraries sound, we can prove that they satisfy the more general pattern, and thus include them at whatever more concrete type is appropriate.

All the library code we used is repeated below. We show the types that we want to prove that the code has.

$$\begin{array}{lll} \text{ALLOC} & : (\tau) \xrightarrow{\bullet} \text{ref } \tau & \triangleq t-p (t-l \text{ falloc.lam } x.\text{push } x; \text{alloc}); \text{fix} \\ \text{READ} & : (\text{ref } \tau) \xrightarrow{\bullet} \tau & \triangleq t-p (t-l \text{ fread.lam } l.\text{push } l; \text{read}); \text{fix} \\ \text{WRITE} & : (\text{ref } \tau, \tau) \xrightarrow{\bullet} \text{unit} & \triangleq t-p (t-l \text{ fwrite.lam } x.\text{lam } l.\text{push } l; \text{push } x; \text{write}; \text{push } 0); \text{fix} \end{array}$$

where $t-p = \text{thunk push}$ and $t-l = \text{thunk lam}$

Lemma 9.5.43 (alloc sound \mathbf{S}).

$$\forall W \tau. (W, \emptyset, \text{thunk push} (\text{thunk lam falloc.lam } x.\text{push } x; \text{alloc}); \text{fix}) \in \mathcal{V}^{\mathbf{S}}[(\tau^+) \xrightarrow{\bullet} \text{ref } \tau^+]$$

Proof. It suffices to show:

$$\begin{aligned} \forall \varphi \ W' \sqsupseteq W. \varphi \subset \text{dom}(W'.\Psi) \wedge (W', \varphi, v) \in \mathcal{V}^{\mathbf{S}}[\tau^+] \\ \implies (W', \varphi, \text{push } v; \text{alloc}) \in \mathcal{E}^{\mathbf{S}}[\text{ref } \tau^+] \end{aligned}$$

Thus, we choose a $H :_{\varphi} W'$, S , and take two steps $\langle H ; S ; \text{push } v; \text{alloc} \rangle \xrightarrow{2} \langle H, \ell \mapsto v ; S, \ell ; \cdot \rangle$, for fresh ℓ . To complete the proof, we choose W'' to be W' extended with ℓ mapping to $\mathcal{V}^{\mathbf{S}}[\tau^+]$, appropriately restricted, and $\varphi' = \{\ell\}$, which means $(W'', \varphi', \ell) \in \mathcal{V}^{\mathbf{S}}[\text{ref } \tau^+]$ as needed. \square

Lemma 9.5.44 (read sound \mathbf{S}).

$$\forall W \tau. (W, \emptyset, \text{thunk push} (\text{thunk lam fread.lam } l.\text{push } l; \text{read}); \text{fix}) \in \mathcal{V}^{\mathbf{S}}[(\text{ref } \tau^+) \xrightarrow{\bullet} \tau^+]$$

Proof. It suffices to show:

$$\begin{aligned} \forall \ell \ W' \sqsupseteq W. \ell \in \text{dom}(W'.\Psi) \wedge (W', \{\ell\}, \ell) \in \mathcal{V}^{\mathbf{S}}[\text{ref } \tau^+] \\ \implies (W', \{\ell\}, \text{push } \ell; \text{read}) \in \mathcal{E}^{\mathbf{S}}[\tau^+] \end{aligned}$$

Thus, we choose a $H :_{\{\ell\}} W'$, S , and if $W'.\Psi(\ell) \neq \dagger$, take two steps $\langle H ; S ; \text{push } \ell; \text{read} \rangle \xrightarrow{2} \langle H ; S, v ; \cdot \rangle$, which from the invariant on the heap, $(\triangleright W', \emptyset, v) \in \mathcal{V}^{\mathbf{S}}[\tau^+]$ as needed. If $W'.\Psi(\ell) = \dagger$, then the location has been freed and we will reduce to fail MEM, which is also in the relation.

\square

Lemma 9.5.45 (write sound \mathbf{S}).

$$\forall W \tau. (W, \emptyset, \text{thunk push} (\text{thunk lam fwrite.lam } x.\text{lam } l.\text{push } l; \text{push } x; \text{write}; \text{push } 0); \text{fix}) \in \mathcal{V}^{\mathbf{S}}[(\text{ref } \tau^+, \tau^+) \xrightarrow{\bullet} \text{unit}]$$

Proof. It suffices to show:

$$\begin{aligned} \forall \nu \ W' \sqsupseteq W. \{\ell\} \cup \nu \subset \text{dom}(W'.\Psi) \wedge (W', \{\ell\}, \ell) \in \mathcal{V}^{\mathbf{S}}[\text{ref } \tau^+] \wedge (W', \nu, v) \in \mathcal{V}^{\mathbf{S}}[\tau^+] \\ \implies (W', \{\ell\} \cup \nu, \text{push } \ell; \text{push } v; \text{write}; \text{push } 0) \in \mathcal{E}^{\mathbf{S}}[\text{unit}] \end{aligned}$$

Thus, we choose a $H :_{\{\ell\} \cup \varphi} W'$, S , and, if $W'.\Psi(\ell) \neq \dagger$, take four steps:

$$\begin{aligned} & \langle H ; S ; \text{push } \ell ; \text{push } v ; \text{write} ; \text{push } 0 \rangle \rightarrow \\ & \langle H ; S, \ell ; \text{push } v ; \text{write} ; \text{push } 0 \rangle \rightarrow \\ & \langle H ; S, \ell, v ; \text{write} ; \text{push } 0 \rangle \rightarrow \\ & \langle H[\ell \mapsto v] ; S, \ell, v ; \text{push } 0 \rangle \rightarrow \\ & \langle H[\ell \mapsto v] ; S, 0 ; \cdot \rangle \end{aligned}$$

Note that the third step succeeds because the invariant on the heap means that ℓ is bound in it. To complete the proof, it suffices to choose φ' as \emptyset , W'' as an extension that simply decreases the step index, since $(W'', \emptyset, 0) \in \mathcal{V}^S[\text{unit}]$. We know $H[\ell \mapsto v] :_{\{\ell\} \cup \varphi'} W''$ and that W'' is an extension, since the value we updated the location with had the same type as what was at ℓ . If $W'.\Psi(\ell) = \dagger$, then the location has been freed and we will reduce to fail MEM, which is also in the relation.

□

9.5.7 Finally, proving soundness

With all of the compatibility lemmas proved, we can prove the fundamental property of the logical relation:

Theorem 9.5.46 (fundamental property).

If $I; \Gamma \vdash e : \tau$ then $\llbracket I; \Gamma \vdash e^+ : \tau \rrbracket$.

Proof. We prove this by induction over the typing derivation, using a corresponding compatibility lemma for each typing rule. Note that when we cross the boundary, we will switch to using compatibility lemmas for the corresponding extension. □

With that, we can prove type soundness. Note that this references the exception extension covered in Chapter 10, as we close with libraries that could reference it.

Corollary 9.5.47 (type soundness). *If $I; \cdot \vdash e : \tau$ then given libraries γ^{I^S} (where $((k, \emptyset), \emptyset, \gamma^{I^S}) \in \mathcal{G}^S[\llbracket I^S \rrbracket]$) and γ^{I^X} (where $((k, \emptyset), \emptyset, \gamma^{I^X}) \in \mathcal{G}^X[\llbracket I^X \rrbracket]$), for any heap H , stack S , if $\langle H ; S ; \gamma^{I^S} (\gamma^{I^X}(e^+)) \rangle \xrightarrow{*} \langle H' ; S' ; P' \rangle$ then one of:*

- $P' = \cdot$ and $S' = \text{Fail } c$ and $c \in \text{OKERR}$
- $P' = \cdot$ and $S' = S, v$ and $\exists j. (j, v) \in \mathcal{V}^\lambda[\llbracket \tau \rrbracket]$
- $\exists H^* S^* P^*. \langle H' ; S' ; P' \rangle \rightarrow \langle H^* ; S^* ; P^* \rangle$

Proof. This is simply a combination of the fundamental property with the definition of $\mathcal{E}^\lambda[\llbracket \tau \rrbracket]$. □

CASE STUDY: EXCEPTIONS

This extension reuses the same core language, [FunLang](#), as in Chapter 9, and thus the same target, StackLang. We do not reproduce the [FunLang](#) static semantics, the operational semantics of StackLang, or the compiler between them (see Figures 9.1, 9.3, and 9.4).

The key guarantee of a linking types extension is *sound encapsulation*, not only from core code, but from other extensions as well. Indeed, one can use multiple extensions in different parts of the same program. In this section, we develop a second extension: one for exceptional control flow. However, since encapsulated code needs to be isolated from other parts of the program, we cannot have a single boundary with multiple extensions active simultaneously. So that we may freely interleave control flow and mutable state, we expand the extension from the previous section with support for exceptions, rather than developing a new extension from scratch. While it may feel unsatisfying that these features cannot be teased apart, we imagine that many practical linking types extensions really *would* bundle multiple features together, since complex foreign functions are likely to have many different side-effects. Figuring out how to tease apart the semantics and reduce the resulting proof effort is interesting future work.

As before, the first step is to extend the type system. For simplicity, our modal arrows only signal whether a function is pure or impure; they do not distinguish between different (state and control) effects, though our framework is certainly compatible with a fine-grained type system. Therefore, this part of the extension is exactly the same as in the previous section, though we include it again in Figure 10.1 for clarity. For contrast, we typeset this extension in **orange bold font**, use square modal arrows, \rightarrow , and identify the extension with **X**.

$$\begin{array}{l}
 \text{Extended Type } \tau := \tau \mid \text{unit} \mid \text{bool} \mid \text{int} \mid \tau \times \tau \mid \tau + \tau \\
 \quad \mid \mu\alpha.\tau \mid (\tau_1, \dots, \tau_n) \xrightarrow{\square} \tau' \mid \text{ref } \tau \\
 \frac{x : \tau \in \Gamma}{\Gamma \vdash_{\mathbf{X}} x : \tau} \quad \frac{\Gamma, f : (\tau_1, \dots, \tau_n) \xrightarrow{\square} \tau', x_i : \tau_i \vdash_{\mathbf{X}} e : \tau'}{\Gamma \vdash_{\mathbf{X}} \text{fun } f(x_1 : \tau_1, \dots, x_n : \tau_n)\{e\} : (\tau_1, \dots, \tau_n) \xrightarrow{\square} \tau'}
 \end{array}$$

$$\frac{\Gamma \vdash_{\mathbf{X}} e : (\tau_1, \dots, \tau_n) \xrightarrow{\square} \tau' \quad \Gamma \vdash_{\mathbf{X}} e_i : \tau_i}{\Gamma \vdash_{\mathbf{X}} e(e_1, \dots, e_n) : \tau'}$$

Figure 10.1: Linking types for exceptions and state

$\uparrow\tau$	\triangleq	τ
$\uparrow\text{unit}$	\triangleq	unit
$\uparrow\text{bool}$	\triangleq	bool
$\uparrow\text{int}$	\triangleq	int
$\uparrow\tau_1 \times \tau_2$	\triangleq	$\uparrow\tau_1 \times \uparrow\tau_2$
$\uparrow\tau_1 + \tau_2$	\triangleq	$\uparrow\tau_1 + \uparrow\tau_2$
$\uparrow\mu\alpha.\tau$	\triangleq	$\mu\alpha.\uparrow\tau$
$\uparrow(\tau_1, \dots, \tau_n) \rightarrow \tau'$	\triangleq	$(\uparrow\tau_1, \dots, \uparrow\tau_n) \xrightarrow{\square} \uparrow\tau'$

$\downarrow\tau$	\triangleq	τ
$\downarrow\text{unit}$	\triangleq	unit
$\downarrow\text{bool}$	\triangleq	bool
$\downarrow\text{int}$	\triangleq	int
$\downarrow\tau_1 \times \tau_2$	\triangleq	$\downarrow\tau_1 \times \downarrow\tau_2$
$\downarrow\tau_1 + \tau_2$	\triangleq	$\downarrow\tau_1 + \downarrow\tau_2$
$\downarrow\mu\alpha.\tau$	\triangleq	$\mu\alpha.\downarrow\tau$
$\downarrow(\tau_1, \dots, \tau_n) \xrightarrow{\square} \tau'$	\triangleq	$(\downarrow\tau_1, \dots, \downarrow\tau_n) \rightarrow \downarrow\tau'$
$\downarrow(\tau_1, \dots, \tau_n) \xrightarrow{\blacksquare} \tau'$	\triangleq	$(\downarrow\tau_1, \dots, \downarrow\tau_n) \rightarrow \mathbf{U} + (\downarrow\tau')$
$\downarrow\text{ref } \tau$	\triangleq	unit
$\downarrow\tau$	\triangleq	$\mathbf{U} + \downarrow\tau$

where $\mathbf{U} = \mu\alpha.\text{unit} + \text{int} + (\alpha \times \alpha) + (\alpha + \alpha) + ((\alpha) \rightarrow \alpha) + \alpha$

Figure 10.2: Lift and lower functions for exceptions extension

The next step is to define lift and lower, which we present in Figure 10.2. While lift is exactly the same as in the previous case study, lower is quite different. Since code under an \mathbf{X} boundary might throw an uncaught exception, lower must account for the type of an exceptional result. Therefore, we define lower in two steps: the helper metafunction \downarrow accounts for the success case, and the top level \downarrow merges it with the exception case under a sum. Since exceptions might produce a variety of values, we use the universal type \mathbf{U} in the exception case. While the *identity* of exceptions is, indeed, an important aspect of them, in this study we wanted to focus on their control aspect. To handle identity would likely require an open sum type, whether just in the linking code or in the core as well.

Next, we provide a StackLang implementation of an exceptions library in Figure 10.3. Although defining encapsulation wrappers is really the next step of the framework, the library implementation offers some intuition for the way that we model exceptions using delimited continuations in StackLang. A shift with an empty body discards the program until the next reset, which intuitively corresponds to throwing and catching an exception,

CATCH	\triangleq	thunk push (thunk lam fcatch.lam f.push f; call; lam res.push [1, res]; reset); fix
THROW	\triangleq	thunk push (thunk lam fthrow.lam exn.push [0, exn]; shift _()); fix
$\langle \downarrow \text{ref } \tau \rangle$	\triangleq	free; push [1, 0]; reset
$\langle \downarrow (\tau_1, \dots, \tau_n) \xrightarrow{\square} \tau' \rangle$	\triangleq	DUP; push (thunk lam l.push l; free); getlocs; lam res. push [1, res]; reset
$\langle \downarrow \tau \rangle$	\triangleq	lam res.push [1, res]; reset where τ not in above

Figure 10.3: Exception target library & boundary enforcement

respectively. In the implementation, THROW is responsible for tagging an exception value, while CATCH is responsible for tagging a success value. Note that CATCH takes a function corresponding to the computation to run.

With an intuitive understanding of StackLang exceptions, we are ready to define the target-level encapsulation wrappers. Unlike the previous extension, wrappers are required for *every* τ , because the boundary must always be prepared for an uncaught exception. In all cases, the wrapper is responsible for capturing any escaping exceptions, which it does with reset (N.B., if there is no shift under the boundary, this reset is effectively a no-op). For types with pure *values*, the encapsulation wrapper can simply tag the success value before it resets. Meanwhile, references and impure functions are handled as before, modulo tagging and a reset.

With this new extension, we make a new version of our `fib` function (Figure 10.4): this time, we take a list of inputs (1), computing the result of all using a single memo-table (2), and throwing an exception on bad input (3). Notice that there is not a corresponding `catch`, since, in this example, a bad input is an unrecoverable error. Still, our program—and indeed, any program with uncaught exceptions—is safe because the boundary will automatically catch and tag any escaping exceptions.

10.1 SOUNDNESS

Since in Chapter 9 we constructed the `FunLang` model and proved its compatibility lemmas, our task for this section is slightly reduced. First, we need to build a model for the exception extension, then prove the lemmas relating to it.

10.1.1 *Exception extension model*

Our exception extension logical relation shares the same common definitions relating to worlds, heap typings, etc, as our state extension relation, described above and shown previously in Figure 9.11. We use the superscript

```

import( alloc : ((int) → int) → ref ((int) → int),
        read : (ref ((int) → int)) → ((int) → int),
        write : (ref ((int) → int), ((int) → int)) → unit,
        catch : (() → U) → U + U,
        throw : (U) → int)
fun fiblist(lst : μα.(int × α) + unit){}
[let mtbl = alloc(fun f(n : int){-1}) in
 let mf = fun mutfib(y : int){
    if x = 0 {0}{if x = 1{1}{}
    let m = read(mtbl) in
    if m(x) = -1{
       let r = mutfib(x - 1) + mutfib(x - 2) in
       let _ = write(mtbl,fun f(n){if n = x{r}{m(x)}}) in
       r
    }{m(x)}
   } in
 fun mutfiblist(l : μα.(int × α) + unit){
  match unfold l
  x {if fst x < 0 {throw(fold inl ())} {
    fold inl(mf(fst x),mutfiblist(snd x))}}
  y {fold inr()}
 }
 }]{lst}X
U + μα.(int × α) + unit
}

```

(1)
(2)
(3)

Figure 10.4: Example: fibonacci with input checks and memoization

X to identify elements of this relation. The differences show up in the value and expression relations, which we present in Figure 10.5. The value relation cases aside from functions are the same as in the state extension, as function values are the only place where the possibility of exceptions can arise. Structurally, the value relations for \rightarrow and $\xrightarrow{\square}$ are identical to the state extension: what is different is that $\xrightarrow{\square}$ appeals to the $\mathcal{E}^{\text{X}}[\tau]$ relation, and $\xrightarrow{\blacksquare}$ appeals to $\mathcal{E}^{\text{X}}[\tau]\downarrow$. These expression relations capture the exceptional control flow.

In particular, $\mathcal{E}^{\text{X}}[\tau]$ is the exact same expression relation as in the state logical relation, which means that if a term is in that relation, it will either run forever, evaluate to a well-defined error, or terminate in a value of type τ . Clearly, this rules out an exception being thrown, which would be a value of type U . Since \rightarrow says the body is in this relation, the body cannot throw (uncaught) exceptions, which is exactly what we want: extensional exception-free-ness. On the other hand, the $\mathcal{E}^{\text{X}}[\tau]\downarrow$ relation is defined to allow exceptional return, in the following way. It says, if given a continuation K drawn from the continuation relation $\mathcal{K}[\tau \Rightarrow \tau]$, the result of wrapping the program in the continuation should then be in the exception-free relation $\mathcal{E}^{\text{X}}[\tau']$ —i.e., K should have caught any possible exception and handled it, or the normal return value of type τ , and in any case produce a τ' . We syntactically restrict the programs in $\mathcal{E}^{\text{X}}[\tau]\downarrow$ from having resets in them, as this would allow exceptions flowing out to be caught, when they should be caught in code in the continuations or in the $\mathcal{E}^{\text{X}}[\tau]$ relation. $\mathcal{K}[\tau \Rightarrow \tau']$ itself is defined to take an arbitrary result program P , drawn from $\mathcal{R}[\tau]$, and prove that the result of substituting that is in $\mathcal{E}^{\text{X}}[\tau']$. Those results are exactly the two possibilities for programs in $\mathcal{E}^{\text{X}}[\tau]\downarrow$: either returning a normal value of type τ , or throwing a tagged exception using shift. This pattern of relations is not novel: it shows up in other TT -closed relations for exceptions (e.g., (Dreyer et al., 2012; New et al., 2016)), but typically, the relation that we have as $\mathcal{E}^{\text{X}}[\tau]$ is called \mathcal{O} and it makes no restrictions on the shape of value you end up with at the end: just that if you terminate, there will be a value on top of the stack, for example. This is sufficient to reason about exceptions in whole programs, but our need to encapsulate the control effects and allow the resulting value to cross boundaries means we need more structure, hence our dual expression relation structure.

10.1.2 Proving \uparrow sound

Lemma 10.1.1 (lift X). $\forall W \forall v. (W, \emptyset, v) \in \mathcal{V}^{\text{X}}[\uparrow\tau] \iff (W.k, v) \in \mathcal{V}^{\lambda}[\tau]$

Proof. We prove this by simultaneous induction over the size k and the structure of τ .

$$\begin{aligned}
\mathcal{V}^X[\text{unit}] &= \{(W, \emptyset, 0)\} \\
\mathcal{V}^X[\text{bool}] &= \{(W, \emptyset, n)\} \\
\mathcal{V}^X[\text{int}] &= \{(W, \emptyset, n)\} \\
\mathcal{V}^X[\tau_1 \times \tau_2] &= \{(W, \varphi, [v_1, v_2]) \mid \varphi \subset \text{dom}(W.\Psi) \wedge \varphi_1 \cup \varphi_2 = \varphi \wedge \\
&\quad (W, \varphi_1, v_1) \in \mathcal{V}^X[\tau_1] \wedge (W, \varphi_2, v_2) \in \mathcal{V}^X[\tau_2]\} \\
\mathcal{V}^X[\tau_1 + \tau_2] &= \{(W, \varphi, [0, v]) \mid \varphi \subset \text{dom}(W.\Psi) \wedge (W, \varphi, v) \in \mathcal{V}^X[\tau_1]\} \\
&\cup \{(W, \varphi, [1, v]) \mid \varphi \subset \text{dom}(W.\Psi) \wedge (W, \varphi, v) \in \mathcal{V}^X[\tau_2]\} \\
\mathcal{V}^X[\mu\alpha.\tau] &= \{(W, \varphi, v) \mid (W, \varphi, v) \in \triangleright \mathcal{V}^X[\tau[\mu\alpha.\tau/\alpha]]\} \\
\mathcal{V}^X[\text{ref } \tau] &= \{(W, \{\ell\}, \ell) \mid W.\Psi(\ell) = [\mathcal{V}^X[\tau]]_{W.k} \mid \dagger\} \\
\mathcal{V}^X[(\tau_1, \dots, \tau_n) \xrightarrow{\square} \tau'] &= \{(W, \emptyset, \text{thunk push (thunk lam f.lam } x_n. \dots. \text{ lam } x_1.P); \text{fix}) \mid \\
&\forall i \varphi_i W' \sqsupseteq W. \varphi_i \subset \text{dom}(W'.\Psi) \wedge (W', \varphi_i, v_i) \in \mathcal{V}^X[\tau_i] \\
&\implies (W', \bigcup_i \varphi_i, [x_1 \mapsto v_1, \dots, x_n \mapsto v_n, \\
&\quad f \mapsto (\text{thunk push (thunk lam f.lam } x_n. \dots. \text{ lam } x_1.P); \text{fix})]P) \\
&\in \mathcal{E}^X[\tau']\} \\
\mathcal{V}^X[(\tau_1, \dots, \tau_n) \xrightarrow{\blacksquare} \tau'] &= \{(W, \varphi, \text{thunk push (thunk lam f.lam } x_n. \dots. \text{ lam } x_1.P); \text{fix}) \mid \\
&\varphi \subset \text{dom}(W.\Psi) \wedge \forall i \varphi_i W' \sqsupseteq W. \\
&\varphi_i \subset \text{dom}(W'.\Psi) \wedge (W', \varphi_i, v_i) \in \mathcal{V}^X[\tau_i] \\
&\implies (W', \varphi \cup \bigcup_i \varphi_i, [x_1 \mapsto v_1, \dots, x_n \mapsto v_n, \\
&\quad f \mapsto (\text{thunk push (thunk lam f.lam } x_n. \dots. \text{ lam } x_1.P); \text{fix})]P) \\
&\in \mathcal{E}^X[\tau']\}
\end{aligned}$$

$K ::= \text{push } v_1; \text{push } v_2; \dots \text{push } v_n; [\cdot]; P$

$$\begin{aligned}
\mathcal{E}^X[\tau] &= \{(W, \varphi_p, P) \mid \text{reset } \notin P \wedge \forall (W, \varphi_k, K) \in \mathcal{K}[\tau \Rightarrow \tau']. (W, \varphi_p \cup \varphi_k, K[P]) \in \mathcal{E}^X[\tau']\} \\
\mathcal{R}[\tau] &= \{(W, \varphi, \text{push } v) \mid (W, \varphi, v) \in \mathcal{V}^X[\tau]\} \\
&\cup \{(W, \varphi_p \cup \varphi_v, \text{push } [0, v]; \text{shift } -(); P) \mid (W, \varphi_v, v) \in \mathcal{V}^X[U] \wedge \text{reset } \notin P\} \\
\mathcal{K}[\tau \Rightarrow \tau'] &= \{(W, \varphi, K) \mid \varphi = \text{flocs}(K) \wedge \forall W' \sqsupseteq W. (W', \varphi', P) \in \mathcal{R}[\tau']. \\
&\quad (W', \varphi \cup \varphi', K[P]) \in \mathcal{E}^X[\tau']\} \\
\mathcal{E}^X[\tau] &= \{(W, \varphi, P) \mid \forall H: \varphi W, S. \text{running}_{W.k}(\langle H; S; P \rangle) \vee \exists j < W.k. H', S'. \\
&\quad \langle H; S; P \rangle \xrightarrow{*} j \langle H'; S'; \cdot \rangle \wedge ((S' = \text{Fail } c \wedge c \in \text{OKERR}) \\
&\quad \vee \exists v \varphi' W' \sqsupseteq W. (S' = S, v \wedge H':_{\varphi' \cup \varphi} W' \wedge (W', \varphi', v) \in \mathcal{V}^X[\tau])))\}
\end{aligned}$$

where $\text{OKERR} \triangleq \{\text{MEM}\}$

Figure 10.5: Exception extension logical relation: main definition

Case `unit/bool/int`. In this case, the values are trivially in the relation, by definition.

Case $\tau_1 \times \tau_2 / \tau_1 + \tau_2$. These follow straightforwardly by appealing to the inductive hypothesis.

Case $\mu\alpha.\tau$. In this case, we can appeal to our inductive hypothesis at a smaller k (as our type may have gotten larger).

Case $(\tau_1, \dots, \tau_n) \rightarrow \tau'$. This follows by application of the induction hypothesis.

□

10.1.3 Proving $\langle \downarrow \tau \rangle$ satisfies \downarrow

Lemma 10.1.2 (wrap closed \mathbf{X}). $\forall \tau. fvars(\langle \downarrow \tau \rangle) = \emptyset$

Proof. This follows by simple inspection of the definition. □

Lemma 10.1.3 (encapsulation continuation). $\forall W \tau. (W, \emptyset, \langle \downarrow \tau \rangle) \in \mathcal{K}[\tau \rightarrow \uparrow \downarrow \tau]$

Proof. We proceed by case analysis on τ , handling the majority of the cases for which $\langle \downarrow \tau \rangle$ is the default, exception catching case first.

In those cases, which by inspection, $\uparrow \downarrow \tau = \mathbf{U} + \tau$, the proof obligation is to show that

$$(W, \emptyset, \text{lam res.push [1, res]; reset}) \in \mathcal{K}[\tau \rightarrow \mathbf{U} + \tau]$$

That means:

$$\begin{aligned} \forall \varphi. (W, \varphi, v) \in \mathcal{V}^{\mathbf{X}}[\mathbf{U}] &\implies \\ &(W, \varphi, \text{push [0, v]; shift } _)(); \text{lam res.push [1, res]; reset}) \in \mathcal{E}^{\mathbf{X}}[\mathbf{U} + \tau] \\ \wedge \forall \varphi. (W, \varphi, v) \in \mathcal{V}^{\mathbf{X}}[\tau] &\implies \\ &(W, \varphi, \text{push } v; \text{lam res.push [1, res]; reset}) \in \mathcal{E}^{\mathbf{X}}[\mathbf{U} + \tau] \end{aligned}$$

We consider each case in turn. First, we consider the case when an exception value is raised. We choose an heap $H :_{\varphi} W$, and a stack S , and see that the term runs as follows:

$$\begin{aligned} &\langle H ; S ; \text{push [0, v]; shift } _)(); \text{lam res.push [1, res]; reset} \rangle \rightarrow \\ &\langle H ; S, [0, v] ; \text{shift } _)(); \text{lam res.push [1, res]; reset} \rangle \rightarrow \\ &\langle H ; S, [0, v] ; \cdot \rangle \end{aligned}$$

At this point, we clearly satisfy the requirements of the expression relation. In the other case, when no exception is raised, we again choose a heap $H :_{\varphi} W$ (note this is a different set of relevant locations!), and run as follows:

$$\begin{aligned}
& \langle H ; S ; \text{push } v; \text{lam res.push } [1, \text{res}]; \text{reset} \rangle \rightarrow \\
& \langle H ; S, v ; \text{lam res.push } [1, \text{res}]; \text{reset} \rangle \rightarrow \\
& \langle H ; S ; \text{push } [1, v]; \text{reset} \rangle \rightarrow \\
& \langle H ; S, [1, v] ; \text{reset} \rangle \rightarrow \\
& \langle H ; S, [1, v] ; \cdot \rangle
\end{aligned}$$

Again, we satisfy the relation, this time in the other disjunct.

Now, we consider the other two types, which use unique wrapping code:

Case $\text{ref } \tau$. Our obligation is to show:

$$\begin{aligned}
& \forall \varphi. (W, \varphi, v) \in \mathcal{V}^X[\mathbb{U}] \\
& \implies (W, \varphi, \text{push } [0, v]; \text{shift } _(); \text{free}; \text{push } [1, 0]; \text{reset}) \in \mathcal{E}^X[\mathbb{U} + \text{unit}] \\
& \wedge \forall \varphi. (W, \varphi, v) \in \mathcal{V}^X[\text{ref } \tau] \\
& \implies (W, \varphi, \text{push } v; \text{free}; \text{push } [1, 0]; \text{reset}) \in \mathcal{E}^X[\mathbb{U} + \text{unit}]
\end{aligned}$$

The first case is identical to our first one, so we only consider the second case. By inspection of $\mathcal{V}^X[\text{ref } \tau]$, we know for some ℓ , $\varphi = \{\ell\}$, $v = \ell$, and $W.\Psi(\ell) = [\mathcal{V}^X[\tau]]_{W.k}$. This means when we choose a heap $H :_\varphi W$, we know it will have ℓ bound to some value in $\triangleright[\mathcal{V}^X[\tau]]_{W.k}$, though as we will see, the actual value does not matter. We will then take four steps:

$$\begin{aligned}
& \langle H ; S ; \text{push } \ell; \text{free}; \text{push } [1, 0]; \text{reset} \rangle \rightarrow \\
& \langle H ; S, \ell ; \text{free}; \text{push } [1, 0]; \text{reset} \rangle \rightarrow \\
& \langle H \setminus \ell ; S ; \text{push } [1, 0]; \text{reset} \rangle \rightarrow \\
& \langle H \setminus \ell ; S, [1, 0] ; \text{reset} \rangle \rightarrow \\
& \langle H \setminus \ell ; S, [1, 0] ; \cdot \rangle
\end{aligned}$$

Now we choose W' to be W , but with ℓ updated to be marked as dead, and choose $\varphi' = \emptyset$. Still, $(H \setminus \ell) :_\varphi W'$ (as dead elements in the world are ignored), and by definition, $(W', \emptyset, 0) \in \mathcal{V}^X[\text{unit}]$, which, along with the appropriate tag, is sufficient to satisfy $\mathcal{V}^X[\mathbb{U} + \text{unit}]$, so we are done with this case.

Case $(\tau_1, \dots, \tau_n) \xrightarrow{*} \tau'$. Our obligation is to show:

$$\begin{aligned}
& \forall v \varphi. (W, \varphi, v) \in \mathcal{V}^X[\llbracket U \rrbracket] \\
& \implies (W, \varphi, \text{push } [0, v]; \text{shift } _(); \text{lam } x.(\text{push } x; \text{push } x); \\
& \quad \text{push } (\text{thunk lam } l. \text{push } l; \text{free}); \text{getlocs}; \text{lam res. push } [1, \text{res}]; \text{reset}) \\
& \quad \in \mathcal{E}^X[\llbracket U + (\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau' \rrbracket] \\
& \wedge \forall v \varphi. (W, \varphi, v) \in \mathcal{V}^X[\llbracket (\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau' \rrbracket] \\
& \implies (W, \varphi, \text{push } v; \text{lam } x.(\text{push } x; \text{push } x); \\
& \quad \text{push } (\text{thunk lam } l. \text{push } l; \text{free}); \text{getlocs}; \text{lam res. push } [1, \text{res}]; \text{reset}) \\
& \quad \in \mathcal{E}^X[\llbracket U + (\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau' \rrbracket]
\end{aligned}$$

As before, the first case is identical to previous, so we only consider the second case. Once we pick an arbitrary stack S and a heap $H :_\varphi W$, we take the following steps:

$$\begin{aligned}
& \langle H ; S ; \text{push } v; \text{lam } x.(\text{push } x; \text{push } x); \text{push } (\text{thunk lam } l. \text{push } l; \text{free}); \text{getlocs}; \text{lam res. push } [1, \text{res}]; \text{reset} \rangle \xrightarrow{4} \\
& \langle H ; S, v, v ; \text{push } (\text{thunk lam } l. \text{push } l; \text{free}); \text{getlocs}; \text{lam res. push } [1, \text{res}]; \text{reset} \rangle \rightarrow \\
& \langle H ; S, v, v, (\text{thunk lam } l. \text{push } l; \text{free}) ; \text{getlocs}; \text{lam res. push } [1, \text{res}]; \text{reset} \rangle
\end{aligned}$$

Now, we know that getlocs will run the thunk on top of the stack once for every free location one position down the stack, which means everything reachable from our function value. Assume those locations are ℓ_1, \dots, ℓ_k . Then we step as follows:

$$\begin{aligned}
& \langle H ; S, v, v, (\text{thunk lam } l. \text{push } l; \text{free}) ; \text{getlocs}; \text{lam res. push } [1, \text{res}]; \text{reset} \rangle \xrightarrow{3k+1} \\
& \langle H \setminus \{\ell_1, \dots, \ell_k\} ; S, v ; \text{lam res. push } [1, \text{res}]; \text{reset} \rangle \rightarrow \\
& \langle H \setminus \{\ell_1, \dots, \ell_k\} ; S ; \text{push } [1, v]; \text{reset} \rangle \rightarrow \\
& \langle H \setminus \{\ell_1, \dots, \ell_k\} ; S, [1, v] ; \text{reset} \rangle \rightarrow \\
& \langle H \setminus \{\ell_1, \dots, \ell_k\} ; S, [1, v] ; \cdot \rangle
\end{aligned}$$

Now that we have terminated, we have to fulfill the obligations of $\mathcal{E}^X[\llbracket U + (\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau' \rrbracket]$. We choose $\varphi' = \emptyset$, and W' such that every location in φ has been marked dead. By invariant of the relation, $\varphi = \{\ell_1, \dots, \ell_k\}$. Our heap satisfies the world, by construction, and everything that should be dead is, so the only thing that remains is to show that $(W', \emptyset, [1, v]) \in \mathcal{V}^X[\llbracket U + (\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau' \rrbracket]$. This follows from the definition of $\mathcal{V}^X[\llbracket \tau + \tau \rrbracket]$ and our hypothesis on v , once we substitute our empty relevant location set in.

□

10.1.4 Proving libraries satisfy types

We now need to show that our exception library satisfies the proper semantic types. We present the definitions and their intended types first, after which we show the proofs.

$$\begin{aligned} \text{CATCH} &: ((\square \rightarrow \tau) \rightarrow \mathbf{U} + \tau) \\ &\triangleq t-p (t-l \text{fcatch}.lam f.push f; call; lam res.push [1, res]; reset); fix \\ \text{THROW} &: (\mathbf{U}) \rightarrow \tau \\ &\triangleq t-p (t-l \text{fthrow}.lam exn.push [0, exn]; shift _()); fix \end{aligned}$$

where $t-p$ = thunk push and $t-l$ = thunk lam

Lemma 10.1.4 (catch sound \mathbf{X}).

$$\begin{aligned} \forall W \tau. (W, \emptyset, \text{thunk push (thunk lam fcatch.lam f.push f; call;} \\ \text{lam res.push [1, res]; reset); fix}) \\ \in \mathcal{V}^{\mathbf{X}}[((\square \rightarrow \tau) \rightarrow \mathbf{U} + \tau)] \end{aligned}$$

Proof. It suffices to show:

$$\begin{aligned} \varphi \subset \text{dom}(W.\Psi) \wedge \forall v \varphi W' \sqsupseteq W. \varphi \subset \text{dom}(W'.\Psi) \wedge (W', \varphi, v) \in \mathcal{V}^{\mathbf{X}}[(\square \rightarrow \tau)] \\ \implies (W', \varphi, \text{push } v; \text{call; lam res.push [1, res]; reset}) \in \mathcal{E}^{\mathbf{X}}[\mathbf{U} + \tau] \end{aligned}$$

Thus, we need to consider heap $H :_{\varphi \cup \varphi^k} W'$, stack S , and after two steps, are running the body of v . From the definition of $\mathcal{V}^{\mathbf{X}}[(\square \rightarrow \tau)]$, we know that the body, which has to arguments, is in $\mathcal{E}^{\mathbf{X}}[\tau]^{\frac{1}{2}}$. We instantiate that with $K = [\cdot]; \text{lam res.push [1, res]; reset}$, and thus it suffices to show that $(W'', \emptyset, K) \in \mathcal{K}[\tau \Rightarrow \mathbf{U} + \tau]$. In the case that a normal value is returned, this tags it in with 1 and returns it, satisfying the relation. In the case that an exceptional value is produced, it is already tagged with 0, and immediately reduces to the value, so we are done.

□

Lemma 10.1.5 (throw sound \mathbf{X}).

$$\begin{aligned} \forall W \tau. (W, \emptyset, \text{thunk push (thunk lam fthrow.lam exn.push [0, exn]; shift _()); fix}) \\ \in \mathcal{V}^{\mathbf{X}}[(\mathbf{U}) \rightarrow \tau] \end{aligned}$$

Proof. It suffices to show:

$$\begin{aligned} \varphi \subset \text{dom}(W.\Psi) \wedge \forall v \varphi W' \sqsupseteq W. \varphi \subset \text{dom}(W'.\Psi) \wedge (W', \varphi, v) \in \mathcal{V}^{\mathbf{X}}[\mathbf{U}] \\ \implies (W', \varphi, \text{push } [0, v]; \text{shift } _()) \in \mathcal{E}^{\mathbf{X}}[\tau]^{\frac{1}{2}} \end{aligned}$$

We choose an arbitrary τ' , K , and the result is now immediate from the definition of $\mathcal{K}[\tau \Rightarrow \tau']$, since our term is already in the form of the exception result.

□

10.1.5 Compatibility lemmas & type soundness

Since we already proved compatibility lemmas for **FunLang**, we just need to prove compatibility lemmas for the exception extension, and one for the boundary rule for the exception extension. We show the proof for the boundary term, which is interesting; the rest of the compatibility lemmas we defer to Appendix E.

We use the following shorthand for typing rules for the exception notation, which supplements the notation defined in §9.5.5.

$$\llbracket \Gamma \vdash_X P : \tau \rrbracket \equiv \forall W \varphi \gamma. (W, \varphi, \gamma) \in \mathcal{G}^X[\Gamma] \implies (W, \text{flocs}(\gamma(P)), \gamma(P)) \in \mathcal{E}^X[\tau]^\heartsuit$$

Lemma 10.1.6 (boundary X). $\llbracket I^X \uplus \uparrow \Gamma \vdash_X P : \tau \rrbracket \implies \llbracket I; \Gamma \vdash P; \langle \downarrow \tau \rangle : \downarrow \tau \rrbracket$

Proof. The general approach of this proof is similar to the one for **S** (Lemma 9.5.4); the difference, of course, is that the **X** logical relation has a different shape, and so some details are different.

Expanding the goal, we see we need to show:

$$\forall k \gamma. \forall ((k, \emptyset), \emptyset, \gamma^{I^S}) \in \mathcal{G}^S[I^S]. \forall ((k, \emptyset), \emptyset, \gamma^{I^X}) \in \mathcal{G}^X[I^X]. (k, \gamma) \in \mathcal{G}^\lambda[\Gamma] \implies (k, \gamma^{I^X}(\gamma^{I^S}(\gamma(P; \langle \downarrow \tau \rangle)))) \in \mathcal{E}^\lambda[\downarrow \tau]$$

We note due to Lemma 10.1.2 that $\langle \downarrow \tau \rangle$ is closed, so can push the substitution in to only around P .

The hypothesis that we are working with says:

$$\forall W \varphi \gamma (W, \varphi, \gamma) \in \mathcal{G}^X[I^X \uplus \uparrow \Gamma] \implies (W, \varphi, \gamma(P)) \in \mathcal{E}^X[\tau]^\bullet$$

To instantiate the hypothesis, we need an environment γ' that satisfies $\mathcal{G}^S[I^X \uplus \uparrow \Gamma]$. We argue that it is exactly γ composed with γ^{I^X} : we know they are disjoint, and we know the former can be lifted into the latter via Lemma 10.1.1. This means, in particular, that φ is \emptyset . Now, we need to choose a continuation and return type τ_A from $\mathcal{K}[\tau \rightarrow \tau_A]$. We choose $\langle \downarrow \tau \rangle$, with τ_A set to $\uparrow \downarrow \tau$, which we know, with any world and empty φ , is in the relation from Lemma 10.1.3. This then tells us that $(W, \emptyset, \gamma(P); \langle \downarrow \tau \rangle)$ is in $\mathcal{E}^X[\uparrow \downarrow \tau]^\circ$.

This means we can use the arbitrary heap H we are given initially to instantiate this relation, as our world and set of relevant locations makes

no restriction on the heap. We also use the arbitrary stack S we are given. This means that we know that either we run past our step index budget (in which case we are trivially in $\mathcal{E}^\lambda[\Downarrow\tau]\rho$, our overall goal), or after some number of steps we have either run to an acceptable failure state (also okay), or we have terminated in a value v , at a future world W' , with relevant locations φ' such that $(W', \varphi', v) \in \mathcal{V}^X[\Uparrow\Downarrow\tau]$. By inspection of the value relation, we can see for all types $\Uparrow\Downarrow\tau$, φ' will be \emptyset . At this point, the result follows from Lemma 10.1.1. \square

Lemma 10.1.7 (unit). *Show that $[\Gamma \vdash_X \text{push } 0 : \text{unit}]$.*

Proof. See E.0.8. \square

Lemma 10.1.8 (bool). *Show for any n , $[\Gamma \vdash_X n : \text{bool}]$.*

Proof. See E.0.9. \square

Lemma 10.1.9 (if). *If $[\Gamma \vdash_X P_1 : \text{bool}]$, $[\Gamma \vdash_X P_2 : \tau]$, and $[\Gamma \vdash_X P_3 : \tau]$ then*

$$[\Gamma \vdash_X P_1; \text{if0 } P_2 \ P_3 : \tau].$$

Proof. See E.0.10. \square

Lemma 10.1.10 (int). *For any n , show $[\Gamma \vdash_X \text{push } n : \text{int}]$.*

Proof. See E.0.11. \square

Lemma 10.1.11 (op- $=$). *If $[\Gamma \vdash_X P_1 : \text{int}]$ and $[\Gamma \vdash_X P_2 : \text{int}]$, then $[\Gamma \vdash_X P_1; P_2; \text{equal?} : \text{bool}]$.*

Proof. See E.0.12. \square

Lemma 10.1.12 (op- j). *If $[\Gamma \vdash_X P_1 : \text{int}]$ and $[\Gamma \vdash_X P_2 : \text{int}]$, then $[\Gamma \vdash_X P_1; P_2; \text{less?} : \text{bool}]$.*

Proof. See E.0.13. \square

Lemma 10.1.13 (op- $+$). *If $[\Gamma \vdash_X P_1 : \text{int}]$ and $[\Gamma \vdash_X P_2 : \text{int}]$, then $[\Gamma \vdash_X P_1; P_2; \text{add} : \text{int}]$.*

Proof. See E.0.14. \square

Lemma 10.1.14 (var). $[\Gamma \vdash_X \text{push } x : \tau]$

Proof. See E.0.15. \square

Lemma 10.1.15 (pair). *If $[\Gamma \vdash_X P_1 : \tau_1]$ and $[\Gamma \vdash_X P_2 : \tau_2]$ then $[\Gamma \vdash_X P_1; P_2; \text{lam } x_2.\text{lam } x_1.\text{push } [x_1, x_2] : \tau_1 \times \tau_2]$*

Proof. See E.0.16. \square

Lemma 10.1.16 (fst). *If $\llbracket \Gamma \vdash_{\mathbf{X}} P : \tau_1 \times \tau_2 \rrbracket$, then $\llbracket \Gamma \vdash_{\mathbf{X}} P; \text{push } 0; \text{idx} : \tau_1 \rrbracket$.*

Proof. See E.0.17. \square

Lemma 10.1.17 (snd). *If $\llbracket \Gamma \vdash_{\mathbf{X}} P : \tau_1 \times \tau_2 \rrbracket$, then $\llbracket \Gamma \vdash_{\mathbf{X}} P_1; \text{push } 1; \text{idx} : \tau_2 \rrbracket$.*

Proof. See E.0.18. \square

Lemma 10.1.18 (inl). *If $\llbracket \Gamma \vdash_{\mathbf{X}} P : \tau_1 \rrbracket$, then $\llbracket \Gamma \vdash_{\mathbf{X}} P; \text{lam } x.\text{push } [0, x] : \tau_1 + \tau_2 \rrbracket$.*

Proof. See E.0.19. \square

Lemma 10.1.19 (inr). *If $\llbracket \Gamma \vdash_{\mathbf{X}} P : \tau_2 \rrbracket$, then $\llbracket \Gamma \vdash_{\mathbf{X}} P; \text{lam } x.\text{push } [1, x] : \tau_1 + \tau_2 \rrbracket$.*

Proof. See E.0.20. \square

Lemma 10.1.20 (match). *If $\llbracket \Gamma \vdash_{\mathbf{X}} P_0 : \tau_1 + \tau_2 \rrbracket$, $\llbracket \Gamma, x : \tau_1 \vdash_{\mathbf{X}} P_1 : \tau \rrbracket$, and $\llbracket \Gamma, y : \tau_2 \vdash_{\mathbf{X}} P_2 : \tau \rrbracket$, then $\llbracket \Gamma \vdash_{\mathbf{X}} P_0; \text{DUP}; \text{push } 1; \text{idx}; \text{SWAP}; \text{push } 0; \text{idx}; \text{if0 (lam } x.P_1) (\text{lam } y.P_2) : \tau \rrbracket$.*

Proof. See E.0.21. \square

Lemma 10.1.21 (fold). *If $\llbracket \Gamma \vdash_{\mathbf{X}} P : \tau[\mu\alpha.\tau/\alpha] \rrbracket$, then $\llbracket \Gamma \vdash_{\mathbf{X}} P : \mu\alpha.\tau \rrbracket$.*

Proof. See E.0.22. \square

Lemma 10.1.22 (unfold). *If $\llbracket \Gamma \vdash_{\mathbf{X}} P : \mu\alpha.\tau \rrbracket$, then $\llbracket \Gamma \vdash_{\mathbf{X}} P; \text{noop} : \tau[\mu\alpha.\tau] \rrbracket$.*

Proof. See E.0.23. \square

Lemma 10.1.23 (fun). *If $\llbracket \Gamma, f : (\tau_1, \dots, \tau_n) \xrightarrow{\Box} \tau', x_i : \tau_i \vdash_{\mathbf{X}} P : \tau' \rrbracket$, then $\llbracket \Gamma \vdash_{\mathbf{X}} \text{push (thunk push (thunk lam f.lam } x_n. \dots. \text{ lam } x_1.P); fix} : (\tau_1, \dots, \tau_n) \xrightarrow{\Box} \tau' \rrbracket$*

Proof. See E.0.24. \square

Lemma 10.1.24 (app). *If $\llbracket \Gamma \vdash_{\mathbf{X}} P_0 : (\tau_1, \dots, \tau_n) \xrightarrow{\Box} \tau' \rrbracket$ and for $i \in \{1, \dots, n\}$ $\llbracket \Gamma \vdash_{\mathbf{X}} P_i : \tau_i \rrbracket$ then*

$$\llbracket \Gamma \vdash_{\mathbf{X}} P_0; P_1; \text{SWAP} \dots P_n; \text{SWAP}; \text{call} : \tau' \rrbracket$$

Proof. See E.0.25. \square

10.1.6 Finally, soundness

To account for our new extension, we need to update our existing proof of the fundamental property for [FunLang](#) to include the boundary term for \mathbf{X} . The proof itself, of course, still simply dispatches to the appropriate compatibility lemma.

Theorem 10.1.25 (fundamental property). *If $\mathbf{I}; \Gamma \vdash e : \tau$ then $\llbracket \mathbf{I}; \Gamma \vdash e^+ : \tau \rrbracket$.*

Proof. As before, by induction over the typing derivation, using a corresponding compatibility lemma for each typing rule. \square

Type soundness again is a corollary, and thus follows from our re-proven fundamental property:

Corollary 10.1.26 (type soundness). *If $\mathbf{I}; \cdot \vdash e : \tau$ then given libraries $\gamma^{\mathbf{I}^S}$ (where $((k, \emptyset), \emptyset, \gamma^{\mathbf{I}^S}) \in \mathcal{G}^S[\llbracket \mathbf{I}^S \rrbracket]$) and $\gamma^{\mathbf{I}^X}$ (where $((k, \emptyset), \emptyset, \gamma^{\mathbf{I}^X}) \in \mathcal{G}^X[\llbracket \mathbf{I}^X \rrbracket]$), for any heap H , stack S , if $\langle H; S; \gamma^{\mathbf{I}^S}(\gamma^{\mathbf{I}^X}(e^+)) \rangle \xrightarrow{*} \langle H'; S'; P' \rangle$ then one of:*

- $P' = \cdot$ and $S' = \text{Fail } c$ and $c \in \text{OKERR}$
- $P' = \cdot$ and $S' = S, v$ and $\exists j. (j, v) \in \mathcal{V}^\lambda[\llbracket \tau \rrbracket]$
- $\exists H^* S^* P^*. \langle H'; S'; P' \rangle \rightarrow \langle H^*; S^*; P^* \rangle$

Proof. This is simply a combination of the fundamental property with the definition of $\mathcal{E}^\lambda[\llbracket \tau \rrbracket]$. \square

Part IV
CONCLUSIONS

11

RELATED WORK

MULTI-LANGUAGE SEMANTICS Matthews and Findler (2007) propose the idea of a syntactic multi-language as a way of formally studying the semantics of language interoperability. They are motivated by the question “how can we reason formally about multi-language programs?” Their work, extended in Jacob Matthews’ dissertation (Mathews, 2007), gives formal semantics to programs that involved multiple languages by combining the syntax of the languages into a single combined language and using syntactic boundaries to mediate between. As described in Chapter 1, the operational semantics inherits from the original source languages, evaluating terms of A under boundaries to values of A using (nearly) the A operational semantics and translating according to type-directed boundary translations.

While this work is undoubtedly valuable, and indeed, the technique has been used widely (Gray et al., 2005; Gray, 2008; Tov and Pucella, 2010; Osera et al., 2012; Patterson et al., 2017; Scherer et al., 2018; Perconti and Ahmed, 2014; Ahmed and Blume, 2011; New et al., 2016), one critical but often unnoticed issue with this approach is that type soundness is proved of the novel multi-language, where the behaviors prescribed by some type τ in the multi-language need not be the same as the behaviors allowed by τ in the corresponding core language. For simple languages (e.g., where the only effect is divergence as in (Matthews and Findler, 2007)), this may be an immaterial distinction, but even with the addition of state, the operational semantics of the multi-language now must consider the heap, and thus a pure language embedded in such a multi-language may no longer behave the same, as there is now a heap threaded through. Scherer et al. (2018) consider this issue and argue that a better approach is to use fully abstract embedding from the single languages into the multi-language. While this certainly resolves the issue, it comes at a pretty serious restriction in terms of interaction. Therefore, while it may work for carefully crafted scenarios of interoperability, it doesn’t allow for many realistic uses, where the intent is to bring new expressiveness into the language and thus violate equivalences in the original core language.

While the models shown in Part II similarly are defined jointly with shared logical state, and thus have a potential disconnect to the original sources, the approach we used in Part III is different: we first define a model for just one language, and then prove that interoperability respects that model. Defining single language models first and then connecting the converted terms could equally be used when dealing with scenarios accounted for

by the approach in Part II. If our models were binary, this would amount to fully abstract embedding, but unary models allow for a weaker notion of soundness that nonetheless is much more powerful than the syntactic multi-language semantics from (Matthews and Findler, 2007).

In a slightly different vein, there has also been some work mixing bindings (Barrett et al., 2016) and building multi-language runtimes (Würthinger et al., 2013), but this work does not consider formal semantic properties.

GRADUAL TYPING There has been an immense amount of research on gradual typing (or migratory typing) since the initial pair of papers by Tobin-Hochstadt and Felleisen (2006) and Siek and Taha (2006); we do not aim to address much of it here, since while it does address a particular form of interoperability, the motivation and hence applicability of the work is different from ours. The framing of migratory typing, as outlined by Tobin-Hochstadt and Felleisen (2006), is to take an untyped program and migrate it to a typed one: the operational semantics of both languages are taken to be shared, only the static semantics should differ. The purpose of such a migration is in the title of that first paper: “from scripts to programs”. The motivation of the work by Siek and Taha (2006) is slightly less clear: a typed language is relaxed into an untyped version, creating, similarly, a pair of languages that share the same operational semantics. While this untyped language may not be one that programmers already use (as in the hypothesis of Tobin-Hochstadt and Felleisen), the intent is that such a gradually typed system provides some benefits to programmers who wish to move between the two paradigms.

Our aim is quite different: we take as a starting point that programs are multi-lingual, which can be readily confirmed by examination of most large codebases. This multi-linguality is not, however, of the form advocated by the gradual paradigm: in particular, while some languages may have stronger or weaker type systems, the syntax and operational semantics are inevitably different between them, and some of the static reasoning may simply be incomparable. Indeed, some of the key results, (such as the gradual guarantee of Siek et al. (2015), extolled as “graduality” by New and Ahmed (2018)) rely upon the shared operational semantics, such that programs can be migrated merely by adding or removing types. While it is possible that some of those properties could be recovered by extending the notion of convertibility into a cross-language logical relation, we have not attempted this and thus leave these questions for future work.

This means that we have only a shallow accommodation of gradual typing in our work. Specifically, if one language is typed and the other is an untyped version of the first, our interoperability results still hold, but without any of the important details about migration.

RUST The Rust language has a built-in mechanism for embedding “unsafe” code that could not satisfy the typechecker of “safe” Rust. There have been efforts to characterize the semantic behavior of safe Rust (“unsafe code guidelines”) and prove that some unsafe code, while syntactically not well typed, does not violate those properties. Most notably, the RustBelt project (Jung et al., 2018) gives a semantic model of λ_{Rust} types and uses it to prove the soundness of λ_{Rust} typing rules, but also to prove that the λ_{Rust} implementations of standard library features (essentially unsafe code) are semantically sound inhabitants of their ascribed type specification.

We argue that Rust’s goals for unsafe and how RustBelt approaches them fit into the linking types framework: the unsafe code unquestionably has behavior inexpressible in the rest of Rust, and Rust already has a syntactic boundary construct: unsafe blocks. Moreover, for RustBelt, Jung et al. (2018) created a lifetime logic that could be used to semantically model (some of) that behavior. While in this paper we expect syntactic type checkers for the extended language, there is no particular reason why their lifetime logic approach isn’t equally valid, and probably necessary for sufficiently complex behavior. Since RustBelt uses the same lifetime logic to define the semantics of safe Rust types, our lift and lower (\uparrow and \downarrow) are perhaps not as apparent, but the properties they convey are: in particular, since Rust does not insert encapsulation code around unsafe blocks, RustBelt needs to prove that the code inside satisfies a safe Rust type. Put another way, any encapsulation has to be inlined into the library implementation, rather than inserted by the compiler. We could have taken the same approach with our stateful libraries, manually proving that the resulting values were encapsulated, and as a result not needing the operational wrappers. We chose generality and efficiency of proof, whereas the approach of RustBelt chooses specificity and efficiency at runtime. The type obligation that lift and lower imply about the boundary is being satisfied in both cases.

INTEROPERABILITY VIA TYPED TARGETS. Shao and Trifonov (1998); Trifonov and Shao (1999) studied interoperability much earlier, and closer to our context: they consider interoperability mediated by translation to a common target. They tackle the problem that one language has access to control effects and the other does not. Their approach, however, is different: it relies upon a target language with an effect-based type system that is sufficient to capture the safety invariants, whereas while our realizability approach can certainly benefit from typed target languages, it doesn’t rely upon them. While typed intermediate languages obviously offer real benefits, there are also unaddressed problems, foremost of which is designing a usable type system that is sufficiently general to allow (efficient) compilation from all the languages you want to support. While there are ongoing attempts (probably foremost is the TruffleVM project (Grimmer et al., 2015)) to

design such general intermediates, most have focused their attention on untyped or unsound languages, and in the particular case of TruffleVM, there is as-yet no meta-theory.

PROVING PARTICULAR FFIS SOUND. There has been significant work on how to augment existing unsafe FFIs in order to make them safe, primarily by adding type systems, inference, static analysis, etc. For example, [Furr and Foster \(2005a,b, 2008\)](#) study interactions between OCaml and C via the C FFI, and specifically, how to ensure safety by doing type inference on the C code as it operates over a structural representation of OCaml types. There is some faint similarity to our work, if you assume the type systems could be made sound and take the two interacting languages to not be OCaml and C but rather those languages augmented with the richer types and inference systems.

Along the same lines, [Tan et al. \(2006a,b\); Tan and Morrisett \(2007\)](#) study safe interoperability between Java and C via the JNI. They do this by first ensuring safety for C via CCured ([Necula et al., 2002](#)), and then extending that with static and dynamic checks to ensure that the invariants of Java pointers and APIs can not be violated in C. This is a bit more explicitly a case of two now more-or-less typesafe languages interacting. [Hirzel and Grimm \(2007\)](#) also build a system for safe interoperability between Java and C, though their system, Jeanie, relies on a novel syntax that embeds both languages and is responsible for analyzing both together before compiling to Java and C with appropriate JNI usage. There has been plenty of other research studying how to make the JNI safer by analyzing C for various properties (e.g., looking for exception behavior in ([Li and Tan, 2014](#))).

RICH FFIS. There has been lots of work exploring how to make existing FFIs safer, usually by extending the annotations that are written down so that there is less hand-written (and thus error-prone) code to write. Some of this was done in the context of the Haskell FFI, including work by [Chakravarty \(1999\); Jones et al. \(1997\); Finne et al. \(1998\)](#). While they were certainly intending to preserve type invariants from Haskell, or wanted to express type invariants via different mechanisms, and obviously were concerned about soundness, it's not clear from these papers whether any formal soundness properties were proved. Similar work has also been done in other languages, for example, for Standard ML [Blume \(2001\)](#) embedded C types into ML such that ML programs could safely operate over low-level C representations. This approach fits well with our semantic framework (though, given weak types in C would be unsound), as they have ML types that have the same interpretation as corresponding C types (to minimize copying/conversions), realized by minimal wrapper code.

Another approach to having rich FFIs is to co-design both languages, as has been done in the much more recent verification project Everest (Bhargavan et al., 2017), where a low-level C-like language Low* has been designed (Protzenko et al., 2017) to interoperate with an embedding of a subset of assembly suitable for cryptography (Fromherz et al., 2019). By embedding both languages into the verification framework F*, they are able to prove rich properties about the interactions between the two languages.

AN ABSTRACT FRAMEWORK FOR UNSAFE FFIS. Turcotte et al. (2019) advocate a framework using an abstract version of the foreign language, so soundness can be proved without building a full multi-language. They demonstrate this by proving a modified type safety proof of Lua and C interacting via the C FFI, modeling the C as code that can do arbitrary unsound behavior and thus blamed for all unsoundness. While this approach seems promising in the context of unsound languages, it is less clear how it applies to sound languages.

MODELING FFIS VIA STATE MACHINES. Lee et al. (2010) specify the type (and other) constraints that exist in both the JNI and Python/C FFI via state machines and use that to generate runtime checks to enforce these at runtime. While this is practical work and so they do not prove properties about their system, Jinn, there are many similarities between their approach and ours. In particular, the idea that invariants that cannot be expressed via the languages themselves and should instead be checked via inserted code. We would expect that if their approach were applied to safe languages, we would be able to prove that the code that they inserted satisfied semantic interpretations of the respective types.

SEMANTIC MODELS AND REALIZABILITY MODELS The use of semantic models to prove type soundness has a long history (Milner, 1978). We make use of step-indexed models (Appel and McAllester, 2001; Ahmed, 2004), developed as part of the Foundational Proof-Carrying Code (Ahmed et al., 2010) project, which showed how to scale the semantic approach to complex features found in real languages such as recursive types and higher-order mutable state. While much of the recent work that uses step-indexed models is concerned with program equivalence, one recent project that focuses on type soundness is RustBelt (Jung et al., 2018): as described earlier, they give a semantic model of λ_{Rust} types and use it to prove the soundness of λ_{Rust} typing rules, but also to prove that the λ_{Rust} implementation of standard library features (essentially unsafe code) are semantically sound inhabitants of their ascribed type specification.

Unlike the above, our realizability model interprets source types as sets of target terms. As described in Chapter 2, our work takes inspiration from

Nick Benton’s “low-level semantics for high-level types” (dubbed “realistic realizability”) (Benton, 2006). Following that work, he and collaborators proved type soundness of a pair of source languages by building models over an idealized assembly (Benton and Zarfaty, 2007; Benton and Tabareau, 2009). Krishnaswami et al. (2015) make use of a realizability model to prove consistency of LNL_D a core type theory that integrates linearity and full type dependency. The linear parts of their model, like our interpretation of L^3 types, are directly inspired by the semantic model for L^3 by Ahmed et al. (2007). While they consider interoperability and use realizability models, their approach is quite different from ours, as they introduce both term constructors and types (G and F) that allow direct embedding into the other language, thereby changing it, rather than defining conversions into existing types (which, indeed, is probably impossible in their case). More generally, such realizability models have also been used by Jensen et al. (2013) to verify low-level code using a high-level separation logic, and by Benton and Hur (2009) to verify compiler correctness.

Finally, New et al. (New and Ahmed, 2018; New et al., 2019, 2020) make use of realizability models in their work on semantic foundations of gradual typing, work that we have drawn inspiration from, given gradual typing is, among other things, a particular instance of language interoperability. They compile type casts in a surface gradual language to a target Call-By-Push-Value (Levy, 2001) language without casts, build a realizability model of gradual types and type precision as relations on target terms, and prove properties about the gradual surface language using the model.

VERIFICATION-BASED APPROACHES Much work has been done using high-level program logics to reason about target terms, which can be seen as analogous to the realizability approach. Perhaps most relevant, in the context of the interoperability studied in Part II of this dissertation, is the Cito system of Wang et al. (2014), where code to-be-linked is given a specification over the behavior of target code, and compilation can then proceed relying upon that specification. This clearly renders benefits in terms of language independence, since any compiled code that satisfied that specification could be used. However, there is a significant difference from our work: by incorporating the semantics of types of both languages we can prove that the *conversions* preserve those semantics, and thus allow an end user to gain the benefits of type soundness without having to do any verification. Indeed, proving the conversions sound (or, in the case that they can be no-ops, proving that is okay) is the central result of our approach, and such conversions are not a part of the setup of Wang et al. (2014).

12

FUTURE CONSIDERATIONS

THE INTEROPERABILITY CHALLENGE Large software systems are invariably multi-language programs, and yet the reasoning that a programmer can do within a single language component degrades severely when they move to working across languages. Addressing this deficiency is what we call the interoperability challenge and it was the central target of this dissertation, if one which we necessarily attacked limited aspects. Fundamentally, the dissertation is based upon the observation that we can understand interoperability by understanding translations into common substrates. While distinct languages themselves are usually unrelated artifacts: built out of rules (static and dynamic) that have no connection to one another, to operate together they must, at some point, be translated into some common representation. That translation, or compilation, may throw away important aspects of the languages that we program in, but it is our starting point, as it is how we will begin to recover the same reasoning principles across the entire multi-language system as we had in our single-language program.

WHY TYPE SOUNDNESS The goal of this dissertation was to make progress on the interoperability challenge forward, if by a small amount. We focused on type soundness as a goal, as it is a well-defined but useful property that we want for programs. While practical languages are rarely proved formally type-sound, most aspire to such a state, and thus it is a good benchmark to use in bringing our reasoning from the single-language fantasy into the multi-language reality.

TARGET LANGUAGES While realizability models seem fundamental, as having a common substrate is what allows us to move between languages in our reasoning, the actual target languages were not a subject of study in this dissertation, even though we believe they are of critical importance in solving the interoperability problem in reality. We can see, in the examples covered in Chapters 5 and 6, how our target language need not have rich static reasoning principles to be able to support rich static reasoning about interoperability. But we also see, in the case considered in Chapter 9, that ensuring soundness may require certain operations in the target that would not otherwise be necessary. In that case study, we had an ad-hoc heap reflection primitive, but more principled study of how features and expressiveness in the target can be isolated is critical. One other dimension we didn't consider are typed target languages. In the cases we considered,

where we understand both interoperating languages, we can build our models on top of untyped targets. However, in a more “open-world” scenario, where the code to be linked with may be less specified, static types could prove useful to restrict its behavior. In that case, the approximations that static types capture, or really, the imposition of a model into the actual term language they imply, may be useful.

RUNTIME SYSTEMS More fundamentally, runtime systems challenge interoperability. One reason, and perhaps the primary reason, why the main target for interoperability is currently C, and slowly moving towards Rust, is that both languages have runtime systems that mostly inherit from the operating system. When there are multiple garbage collectors and different user-level threading systems, things become much more complicated. Realizability models clarify this, as they allow us to see that when we compile to a shared target, a small bit of code doesn’t compile to a small bit of code, but rather to that code paired with a large chunk of runtime code. Wrapping that entire thing up to behave like something in the other language may be difficult or impossible. Research into the design of target or intermediate languages may help alleviate this, since if the compiler could instead reuse features of the target instead of providing its own runtime, the reasoning that has to be done would be vastly reduced. But doing this requires our target languages be quite flexible, allowing tunable runtimes that can adapt to different modalities to fit the requirements of different source languages.

The notion that C is “runtimeless” is somewhat misleading: the standard C library, malloc, and indeed, parts of the operating system form its runtime.

THE n^2 PROBLEM In this dissertation, we only considered pairs of languages. While, fundamentally, the interactions at the boundaries are always between a pair of languages, there are questions related to duplication of effort when considering realistic systems with many languages: given n languages, we likely do not want $O(n^2)$ sets of conversions. Perhaps common abstract types have a role to play, or common models. This is not a new problem—indeed, Conway (1958) proposed a “universal computer-oriented language” (UNCOL) for exactly this purpose. More recently, this has also obviously been accomplished in the serialization-based approach described in Chapter 1 by defining a common set of types that are allowed for interoperability, and this approach is again being used by the WebAssembly Interface Types project¹. While this lowest common denominator necessarily limits expressivity, perhaps it is a reasonable compromise in reality.

A CONCRETE STUDY From the practical side, there is also, clearly, much work to do. While, to first approximation, the approach to soundness described in Part II captures how many interoperability systems work currently, there are certainly many details that vary, possibly with consequences.

¹ <https://github.com/WebAssembly/interface-types/>

One starting point for this research would be a systematic review of the multi-language systems that exist, not as a series of references but using actual code examples to compare apples to apples. Such a benchmark, perhaps with intended feedback when mistakes are made, would be a useful contribution on its own, but could also serve to assess how closely the framework described in the first part of this dissertation hews to reality.

MAKING LINKING TYPES PRACTICAL There is also significant practical work in realizing the linking types framework showcased in Part III. While we believe that the core of such functionality is fundamentally important in addressing the interoperability problem, many details were simplified for the sake of a first presentation. For example, the relationship between the original type system and the extended type system: having an entirely new type system, with a corresponding new typechecker, even if it can be a simpler implementation, by, e.g., being slower or doing worse inference, is not really a practical consideration. It would require maintaining a set of type checkers, fixing bugs and adding features to all. Likely, there is a more efficient implementation strategy: either by having the extended code typechecked using some additional plugin to the original typechecker, or perhaps with some translation pass.

VERIFIED CODE WITH EFFICIENT IMPLEMENTATIONS One particularly interesting case of interoperability is improving the efficiency of verified code extracted from proof assistants like Coq ([The Coq Development Team, 1999-2020](#)). The core languages of such proof assistants are intentionally small, and thus implementing complex software entirely within the core language, while highly trustworthy, would likely lead to terrible performance. This is true even after extraction to a language with an optimizing compiler like OCaml. Instead, interesting research by [Boulmé \(2021\)](#) shows that you can link against OCaml code implementing, for example, a SAT solver, that is given precise types in Coq and as a result, improve performance significantly. One could imagine going further with this, and combine verification (or partial verification) of the linked code written in a language like Clight (using, perhaps, the Verified Software Toolchain ([Appel, 2011](#))) to bring the imported code in at more useful types than what would be inherent to C: while arbitrary C can only be used in quite limited ways by Coq if any soundness is to be retained, C that has been verified to obey particular semantic types might indeed be able to form the basis of efficient data structures for verified software. If the Coq code is compiled with the verified CertiCoq ([Anand et al., 2017](#)) compiler, which targets Clight, then our realizability approach comes into full clarity: to safely link with hand-written Clight, such code must satisfy a semantic type that is compatibly with the Clight code generated by the CertiCoq compiler.

While in some ways this whole line of work is parallel to our own, we can also see it as particularly compelling application, as to do it correctly relies on realizability models, encapsulation, and semantic types that we have described in this dissertation.

ON TO THE FUTURE There is, clearly, much work to do. But, the interoperability challenge is complex and has, with a few notable exceptions, as-yet been largely ignored. Language research has focused on techniques for single languages or for the specific case where operational semantics are common (gradual typing), sidestepping the messy reality and rendering theory even more theoretical. We hope that this dissertation has shown that the challenge, while large, is by no means intractable, and laid a few more paving stones along the way towards a future where the reasoning that we have about multi-language systems is not different from the reasoning we have within single languages.

LIST OF FIGURES

Figure 4.1	Syntax for RefHL and RefLL	41
Figure 4.2	Static semantics for RefHL and RefLL	42
Figure 4.3	Syntax and operational semantics for StackLang . .	43
Figure 4.4	Compilers for RefHL and RefLL	44
Figure 4.5	Conversions for RefHL and RefLL	46
Figure 4.6	Logical relation for RefHL and RefLL	47
Figure 5.1	Syntax for Affi (dynamic and static), MiniML and LCVM.	54
Figure 5.2	Dynamic semantics for LCVM.	55
Figure 5.3	Static semantics for for Affi (dynamic).	56
Figure 5.4	Static semantics for for Affi (static).	57
Figure 5.5	Static semantics for for MiniML	58
Figure 5.6	Compiler for MiniML	60
Figure 5.7	Compilers for Affi (static and dynamic).	61
Figure 5.8	Convertibility for MiniML and Affi	62
Figure 5.9	Examples of interoperability for MiniML and Affi , with compilations.	62
Figure 5.10	Supporting definitions for logical relation for MiniML and Affi	63
Figure 5.11	Logical Relation for MiniML and Affi	64
Figure 5.12	MiniML & Affi Logical Relation Supporting Definitions (static).	73
Figure 5.13	MiniML & Affi Value and Expression Relation (static).	74
Figure 6.1	Syntax for L³ and MiniML	77
Figure 6.2	Static semantics for L³	78
Figure 6.3	Static semantics for MiniML	79
Figure 6.4	Syntax for LCVM.	80
Figure 6.5	Operational semantics for LCVM.	81
Figure 6.6	Compiler for L³	82
Figure 6.7	Compiler for MiniML	82
Figure 6.8	Logical Relation for MiniML and L³	85
Figure 6.9	Supporting definitions for Logical Relation for MiniML and L³	86
Figure 9.1	Syntax & static semantics for FunLang	109
Figure 9.2	Syntax for StackLang	110
Figure 9.3	Operational semantics for StackLang	111
Figure 9.4	Compiler from FunLang to StackLang	113
Figure 9.5	Linking types for state	114

Figure 9.6	The boundary term over an arbitrary extension, $+$	115
Figure 9.7	Lift and lower functions for state extension	115
Figure 9.8	State boundary enforcement & target library code .	115
Figure 9.9	Example: fibonacci memoized with state	117
Figure 9.10	FunLang logical relation	118
Figure 9.11	State & exception extension logical relation: preliminary definitions	120
Figure 9.12	State extension logical relation: main definition . .	122
Figure 10.1	Linking types for exceptions and state	133
Figure 10.2	Lift and lower functions for exceptions extension . .	134
Figure 10.3	Exception target library & boundary enforcement .	135
Figure 10.4	Example: fibonacci with input checks and memoization	136
Figure 10.5	Exception extension logical relation: main definition	138

B I B L I O G R A P H Y

- A. Ahmed and M. Blume. An equivalence-preserving CPS translation via multi-language semantics. In M. M. T. Chakravarty, Z. Hu, and O. Danvy, editors, *Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011, Tokyo, Japan, September 19-21, 2011*, pages 431–444. ACM, 2011. doi:[10.1145/2034773.2034830](https://doi.org/10.1145/2034773.2034830). URL <https://doi.org/10.1145/2034773.2034830>.
- A. Ahmed, M. Fluet, and G. Morrisett. L3 : A linear language with locations. *Fundamenta Informaticae*, 77(4):397–449, June 2007.
- A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In Z. Shao and B. C. Pierce, editors, *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 340–353. ACM, 2009. doi:[10.1145/1480881.1480925](https://doi.org/10.1145/1480881.1480925). URL <https://doi.org/10.1145/1480881.1480925>.
- A. Ahmed, A. W. Appel, C. D. Richards, K. N. Swadi, G. Tan, and D. C. Wang. Semantic foundations for typed assembly languages. *ACM Transactions on Programming Languages and Systems*, 32(3):1–67, Mar. 2010.
- A. J. Ahmed. *Semantics of Types for Mutable State*. PhD thesis, Princeton University, Nov. 2004.
- A. Anand, A. Appel, G. Morrisett, Z. Paraskevopoulou, R. Pollack, O. S. Belanger, M. Sozeau, and M. Weaver. Certicoq: A verified compiler for coq. In *The third international workshop on Coq for programming languages (CoqPL)*, 2017.
- A. W. Appel. Verified software toolchain. In *European Symposium on Programming*, pages 1–17. Springer, 2011.
- A. W. Appel and D. A. McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.*, 23(5):657–683, 2001. doi:[10.1145/504709.504712](https://doi.org/10.1145/504709.504712). URL <https://doi.org/10.1145/504709.504712>.
- J. Armstrong. *Making reliable distributed systems in the presence of software errors*. PhD thesis, Royal Institute of Technology, 2003.

- E. Barrett, C. F. Bolz, L. Diekmann, and L. Tratt. Fine-grained Language Composition: A Case Study. In S. Krishnamurthi and B. S. Lerner, editors, *30th European Conference on Object-Oriented Programming (ECOOP 2016)*, volume 56 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:27, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-014-9. doi:[10.4230/LIPIcs.ECOOP.2016.3](https://doi.org/10.4230/LIPIcs.ECOOP.2016.3). URL <http://drops.dagstuhl.de/opus/volltexte/2016/6097>.
- N. Benton. Abstracting allocation: The new new thing. In *Computer Science Logic (CSL)*, Sept. 2006.
- N. Benton and C.-K. Hur. Biorthogonality, step-indexing and compiler correctness. In *Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming*, ICFP ’09, pages 97–108, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-332-7. doi:[10.1145/1596550.1596567](https://doi.acm.org/10.1145/1596550.1596567). URL [http://doi.acm.org/10.1145/1596550.1596567](https://doi.acm.org/10.1145/1596550.1596567).
- N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proceedings of the 7th International Conference on Typed Lambda Calculi and Applications*, TLCA’05, page 86–101, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3540255931. doi:[10.1007/11417170_8](https://doi.org/10.1007/11417170_8). URL https://doi.org/10.1007/11417170_8.
- N. Benton and N. Tabareau. Compiling functional types to relational specifications for low level imperative code. In *Proceedings of TLDI’09: 2009 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation, Savannah, GA, USA, January 24, 2009*, pages 3–14, 2009.
- N. Benton and U. Zarfaty. Formalizing and verifying semantic type soundness of a simple compiler. In *Proceedings of the 9th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, PPDP ’07, page 1–12, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595937698. doi:[10.1145/1273920.1273922](https://doi.org/10.1145/1273920.1273922). URL <https://doi.org/10.1145/1273920.1273922>.
- K. Bhargavan, B. Bond, A. Delignat-Lavaud, C. Fournet, C. Hawblitzel, C. Hritcu, S. Ishtiaq, M. Kohlweiss, R. Leino, J. R. Lorch, K. Maillard, J. Pan, B. Parno, J. Protzenko, T. Ramananandro, A. Rane, A. Rastogi, N. Swamy, L. Thompson, P. Wang, S. Z. Béguelin, and J. K. Zinzindohoue. Everest: Towards a verified, drop-in replacement of HTTPS. In B. S. Lerner, R. Bodík, and S. Krishnamurthi, editors, *2nd Summit on Advances in Programming Languages, SNAPL 2017, May 7-10, 2017, Asilomar*,

- CA, USA*, volume 71 of *LIPICs*, pages 1:1–1:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:[10.4230/LIPICs.SNAPL.2017.1](https://doi.org/10.4230/LIPICs.SNAPL.2017.1). URL <https://doi.org/10.4230/LIPICs.SNAPL.2017.1>.
- L. Birkedal and R. Harper. Relational interpretations of recursive types in an operational setting. In *Theoretical Aspects of Computer Software (TACS)*, 1997.
- M. Blume. No-longer-foreign: Teaching an ml compiler to speak c “natively”. *Electronic Notes in Theoretical Computer Science*, 59(1):36–52, 2001.
- S. Boulmé. *Formally Verified Defensive Programming (efficient Coq-verified computations from untrusted ML oracles)*. Habilitation à diriger des recherches, Université Grenoble-Alpes, Sept. 2021. URL <https://hal.archives-ouvertes.fr/tel-03356701>. See also <http://www-verimag.imag.fr/~boulme/hdr.html>.
- M. M. Chakravarty. C→HASKELL, or yet another interfacing tool. In *Symposium on Implementation and Application of Functional Languages*, pages 131–148. Springer, 1999.
- M. E. Conway. Proposal for an uncol. *Commun. ACM*, 1(10):5–8, oct 1958. ISSN 0001-0782. doi:[10.1145/368924.368928](https://doi.org/10.1145/368924.368928). URL <https://doi.org/10.1145/368924.368928>.
- K. Crary and R. Harper. Syntactic logical relations for polymorphic and recursive types. *Computation, Meaning and Logic: Articles dedicated to Gordon Plotkin*, *Electronic Notes in Theoretical Computer Science*, 172, 2007.
- P.-E. Dagand, N. Tabareau, and E. Tanter. Partial type equivalences for verified dependent interoperability. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, ICFP 2016, page 298–310, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342193. doi:[10.1145/2951913.2951933](https://doi.org/10.1145/2951913.2951933). URL <https://doi.org/10.1145/2951913.2951933>.
- O. Danvy and A. Filinski. Abstracting control. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming*, pages 151–160, 1990.
- C. Dimoulas, S. Tobin-Hochstadt, and M. Felleisen. Complete monitors for behavioral contracts. In *European Symposium on Programming (ESOP)*, Mar. 2012.
- D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. *J. Funct. Program.*, 22(4–5):

477–528, aug 2012. ISSN 0956-7968. doi:[10.1017/S095679681200024X](https://doi.org/10.1017/S095679681200024X). URL <https://doi.org/10.1017/S095679681200024X>.

M. Felleisen. The theory and practice of first-class prompts. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL ’88, page 180–190, New York, NY, USA, 1988. Association for Computing Machinery. ISBN 0897912527. doi:[10.1145/73560.73576](https://doi.org/10.1145/73560.73576). URL <https://doi.org/10.1145/73560.73576>.

M. Felleisen. On the expressive power of programming languages. In *Proceedings of the 3rd European Symposium on Programming*, ESOP ’90, page 134–151, Berlin, Heidelberg, 1990. Springer-Verlag. ISBN 3540525920.

R. B. Findler and M. Felleisen. Contracts for higher-order functions. In *Proceedings of the seventh ACM SIGPLAN international conference on Functional programming*, pages 48–59, 2002.

S. Finne, D. Leijen, E. Meijer, and S. Peyton Jones. H/direct: a binary foreign language interface for haskell. In *Proceedings of the third ACM SIGPLAN international conference on Functional programming*, pages 153–162, 1998.

A. S. Foundation. Apache thrift, 2022. URL <https://thrift.apache.org/>.

A. Fromherz, N. Giannarakis, C. Hawblitzel, B. Parno, A. Rastogi, and N. Swamy. A verified, efficient embedding of a verifiable assembly language. *PACMPL*, 3(POPL):63:1–63:30, 2019. doi:[10.1145/3290376](https://doi.org/10.1145/3290376). URL <https://doi.org/10.1145/3290376>.

M. Furr and J. S. Foster. Checking type safety of foreign function calls. *SIGPLAN Not.*, 40(6):62–72, June 2005a. ISSN 0362-1340. doi:[10.1145/1064978.1065019](https://doi.org/10.1145/1064978.1065019). URL <https://doi.org/10.1145/1064978.1065019>.

M. Furr and J. S. Foster. Checking type safety of foreign function calls. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Chicago, Illinois*, pages 62–72, June 2005b.

M. Furr and J. S. Foster. Checking type safety of foreign function calls. *ACM Transactions on Programming Languages and Systems*, July 2008.

J.-Y. Girard. Une extension de l’interprétation de gödel à l’analyse, et son application à l’élimination des coupures dans l’analyse et la théorie des types. In J. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92. Elsevier, 1971. doi:[https://doi.org/10.1016/S0049-237X\(08\)70843-7](https://doi.org/10.1016/S0049-237X(08)70843-7). URL <https://www.sciencedirect.com/science/article/pii/S0049237X08708437>.

- J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, USA, 1989. ISBN 0521371813.
- K. E. Gray. Safe cross-language inheritance. In *European Conference on Object-Oriented Programming*, pages 52–75. Springer, 2008.
- K. E. Gray, R. B. Findler, and M. Flatt. Fine-grained interoperability through mirrors and contracts. *ACM SIGPLAN Notices*, 40(10):231–245, 2005.
- M. Grimmer, C. Seaton, R. Schatz, T. Würthinger, and H. Mössenböck. High-performance cross-language interoperability in a multi-language runtime. In *Proceedings of the 11th Symposium on Dynamic Languages*, pages 78–90, 2015.
- M. Hirzel and R. Grimm. Jeannie: granting java native interface developers their wishes. In R. P. Gabriel, D. F. Bacon, C. V. Lopes, and G. L. S. Jr., editors, *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007, October 21-25, 2007, Montreal, Quebec, Canada*, pages 19–38. ACM, 2007. doi:[10.1145/1297027.1297030](https://doi.org/10.1145/1297027.1297030). URL <https://doi.org/10.1145/1297027.1297030>.
- K. Jacobs, D. Devriese, and A. Timany. Purity of an st monad: Full abstraction by semantically typed back-translation. *Proc. ACM Program. Lang.*, 6(OOPSLA1), apr 2022. doi:[10.1145/3527326](https://doi.org/10.1145/3527326). URL <https://doi.org/10.1145/3527326>.
- J. B. Jensen, N. Benton, and A. Kennedy. High-level separation logic for low-level code. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL ’13, page 301–314, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450318327. doi:[10.1145/2429069.2429105](https://doi.org/10.1145/2429069.2429105). URL <https://doi.org/10.1145/2429069.2429105>.
- S. P. Jones, T. Nordin, and A. Reid. Greencard: a foreign-language interface for haskell. In *Proc. Haskell Workshop*, 1997.
- R. Jung, J.-H. Jourdan, R. Krebbers, and D. Dreyer. RustBelt: Securing the foundations of the rust programming language. In *ACM Symposium on Principles of Programming Languages (POPL)*, 2018.
- A. Kennedy and D. Syme. Design and implementation of generics for the .net common language runtime. In *Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation*, PLDI ’01, page 1–12, New York, NY, USA, 2001. Association for Computing

- Machinery. ISBN 1581134142. doi:[10.1145/378795.378797](https://doi.org/10.1145/378795.378797). URL <https://doi.org/10.1145/378795.378797>.
- R. Kleffner. A foundation for typed concatenative languages. Master’s thesis, Northeastern University, 2017.
- N. R. Krishnaswami, P. Pradic, and N. Benton. Integrating linear and dependent types. In S. K. Rajamani and D. Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 17–30. ACM, 2015. doi:[10.1145/2676726.2676969](https://doi.org/10.1145/2676726.2676969). URL <https://doi.org/10.1145/2676726.2676969>.
- B. Lee, B. Wiedermann, M. Hirzel, R. Grimm, and K. S. McKinley. Jinn: synthesizing dynamic bug detectors for foreign language interfaces. In B. G. Zorn and A. Aiken, editors, *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010, Toronto, Ontario, Canada, June 5-10, 2010*, pages 36–49. ACM, 2010. doi:[10.1145/1806596.1806601](https://doi.org/10.1145/1806596.1806601). URL <https://doi.org/10.1145/1806596.1806601>.
- X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.
- P. B. Levy. *Call-by-Push-Value*. Ph. D. dissertation, Queen Mary, University of London, London, UK, Mar. 2001.
- S. Li and G. Tan. Exception analysis in the java native interface. *Science of Computer Programming*, 89:273–297, 2014.
- P. Mates, J. Perconti, and A. Ahmed. Under control: Compositionally correct closure conversion with mutable state. In *ACM Conference on Principles and Practice of Declarative Programming (PPDP)*, 2019.
- J. B. Mathews. *The Meaning of Multilanguage Programs*. PhD thesis, University of Chicago, 2007.
- J. Matthews and R. B. Findler. Operational semantics for multi-language programs. In M. Hofmann and M. Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 3–10. ACM, 2007. doi:[10.1145/1190216.1190220](https://doi.org/10.1145/1190216.1190220). URL <https://doi.org/10.1145/1190216.1190220>.
- C. Metz. ”why WhatsApp only needs 50 engineers for its 900m users”, 2015. URL <https://web.archive.org/web/20220303181630/https://www.wired.com/2015/09/whatsapp-serves-900-million-users-50-engineers/>.

- R. Milner. A theory of type polymorphism in programming. *J. Comput. Syst. Sci.*, 17:348–375, 1978.
- G. Morrisett, A. Ahmed, and M. Fluet. L3 : A linear language with locations. In *Typed Lambda Calculi and Applications (TLCA), Nara, Japan*, pages 293–307, Apr. 2005.
- G. C. Necula, S. McPeak, and W. Weimer. Ccured: Type-safe retrofitting of legacy code. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 128–139, 2002.
- M. S. New and A. Ahmed. Graduality from embedding-projection pairs. In *ICFP*, volume 2, pages 73:1–73:30, 2018.
- M. S. New, W. J. Bowman, and A. Ahmed. Fully abstract compilation via universal embedding. In J. Garrigue, G. Keller, and E. Sumii, editors, *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18–22, 2016*, pages 103–116. ACM, 2016. doi:[10.1145/2951913.2951941](https://doi.org/10.1145/2951913.2951941). URL <https://doi.org/10.1145/2951913.2951941>.
- M. S. New, D. R. Licata, and A. Ahmed. Gradual type theory. *Proceedings of the ACM on Programming Languages*, 3:15:1–15:31, 2019.
- M. S. New, D. Jamner, and A. Ahmed. Graduality and parametricity: Together again for the first time. In *POPL*, volume 4, pages 46:1–46:32, 2020.
- U. of Tennessee, B. Univ. of California, U. of Colorado Denver, , and N. Ltd. "LAPACK: Linear algebra PACKage", 2021. URL <http://www.netlib.org/lapack/>.
- P. Osera, V. Sjöberg, and S. Zdancewic. Dependent interoperability. In K. Claessen and N. Swamy, editors, *Proceedings of the sixth workshop on Programming Languages meets Program Verification, PLPV 2012, Philadelphia, PA, USA, January 24, 2012*, pages 3–14. ACM, 2012. doi:[10.1145/2103776.2103779](https://doi.org/10.1145/2103776.2103779). URL <https://doi.org/10.1145/2103776.2103779>.
- D. Patterson and A. Ahmed. Linking Types for Multi-Language Software: Have Your Cake and Eat It Too. In B. S. Lerner, R. Bodík, and S. Krishnamurthi, editors, *2nd Summit on Advances in Programming Languages (SNAPL 2017)*, volume 71 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-032-3. doi:[10.4230/LIPIcs.SNAPL.2017.12](https://doi.org/10.4230/LIPIcs.SNAPL.2017.12). URL <http://drops.dagstuhl.de/opus/volltexte/2017/7125>.

- D. Patterson, J. Perconti, C. Dimoulas, and A. Ahmed. Funtal: reasonably mixing a functional language with assembly. In A. Cohen and M. T. Vechev, editors, *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, pages 495–509. ACM, 2017. doi:[10.1145/3062341.3062347](https://doi.org/10.1145/3062341.3062347). URL <https://doi.org/10.1145/3062341.3062347>.
- D. Patterson, N. Mushtak, A. Wagner, and A. Ahmed. Semantic soundness for language interoperability. In *Proceedings of the 43rd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2022, San Diego, California, June 13-17, 2022*. ACM, 2022.
- J. T. Perconti and A. Ahmed. Verifying an open compiler using multi-language semantics. In Z. Shao, editor, *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8410 of *Lecture Notes in Computer Science*, pages 128–148. Springer, 2014. doi:[10.1007/978-3-642-54833-8_8](https://doi.org/10.1007/978-3-642-54833-8_8). URL https://doi.org/10.1007/978-3-642-54833-8_8.
- A. M. Pitts. Existential types: Logical relations and operational equivalence. *Lecture Notes in Computer Science*, 1443:309–326, 1998.
- A. M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10:321–359, 2000.
- A. M. Pitts and I. D. Stark. Operational reasoning for functions with local state. *Higher order operational techniques in semantics*, pages 227–273, 1998.
- A. M. Pitts and I. D. B. Stark. Observable properties of higher order functions that dynamically create local names, or: What’s new? In A. M. Borzyszkowski and S. Sokołowski, editors, *Mathematical Foundations of Computer Science*, volume 711 of *Lecture Notes in Computer Science*, pages 122–141, Berlin, 1993. Springer-Verlag.
- D. A. Powner. Federal agencies need to address aging legacy systems, 2016. URL <https://web.archive.org/web/20160615044750/https://www.gao.gov/assets/680/677454.pdf>.
- J. Protzenko, J. K. Zinzindohoué, A. Rastogi, T. Ramananandro, P. Wang, S. Z. Béguelin, A. Delignat-Lavaud, C. Hritcu, K. Bhargavan, C. Fournet, and N. Swamy. Verified low-level programming embedded in F. *PACMPL*, 1(ICFP):17:1–17:29, 2017. doi:[10.1145/3110261](https://doi.org/10.1145/3110261). URL <https://doi.org/10.1145/3110261>.

- G. Scherer, M. S. New, N. Rioux, and A. Ahmed. Fabulous interoperability for ML and a linear language. In C. Baier and U. D. Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2018. doi:[10.1007/978-3-319-89366-2_8](https://doi.org/10.1007/978-3-319-89366-2_8). URL https://doi.org/10.1007/978-3-319-89366-2_8.
- H. Sexton. Foreign functions and common lisp. *ACM SIGPLAN Lisp Pointers*, 1(5):11–23, 1987.
- Z. Shao and V. Trifonov. Type-directed continuation allocation. In *International Workshop on Types in Compilation*, pages 116–135. Springer, 1998.
- J. G. Siek and W. Taha. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop*, page 81–92, 2006.
- J. G. Siek, M. M. Vitousek, M. Cimini, and J. T. Boyland. Refined criteria for gradual typing. In *1st Summit on Advances in Programming Languages (SNAPL 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
- I. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, Dec. 1994. URL <http://www.inf.ed.ac.uk/~stark/namhof.html>. Also available as Technical Report 363, University of Cambridge Computer Laboratory.
- T. S. Strickland, S. Tobin-Hochstadt, R. B. Findler, and M. Flatt. Chaperones and impersonators: Run-time support for reasonable interposition. In *ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, page 943–962, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450315616. doi:[10.1145/2384616.2384685](https://doi.org/10.1145/2384616.2384685). URL <https://doi.org/10.1145/2384616.2384685>.
- W. Tait. Intensional interpretations of functionals of finite type i. *The Journal of Symbolic Logic*, 1967.
- G. Tan and G. Morrisett. Ilea: Inter-language analysis across java and c. In *Proceedings of the 22Nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications, OOPSLA '07*, pages 39–56, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-786-5. doi:[10.1145/1297027.1297031](https://doi.org/10.1145/1297027.1297031). URL <http://doi.acm.org/10.1145/1297027.1297031>.

- G. Tan, A. W. Appel, S. Chakradhar, A. Raghunathan, S. Ravi, and D. Wang. Safe java native interface. In *Proceedings of IEEE International Symposium on Secure Software Engineering*, volume 97, page 106. Citeseer, 2006a.
- G. Tan, A. W. Appel, S. Chakradhar, R. Srivaths, A. Raghunathan, and D. Wang. Safe java native interface. In *Proceedings of the 2006 IEEE International Symposium on Secure Software Engineering*, pages 97–106, 2006b.
- P. Teplitzky. Closing the cobol programming skills gap, 2019. URL <https://web.archive.org/web/20210120000847/https://techchannel.com/Enterprise/10/2019/closing-cobol-programming-skills-gap>.
- The Coq Development Team. The Coq Proof Assistant, 1999-2020. URL <https://coq.inria.fr/>.
- A. Timany, L. Stefanescu, M. Krogh-Jespersen, and L. Birkedal. A logical relation for monadic encapsulation of state: Proving contextual equivalences in the presence of runst. *Proc. ACM Program. Lang.*, 2(POPL), dec 2017. doi:[10.1145/3158152](https://doi.org/10.1145/3158152). URL <https://doi.org/10.1145/3158152>.
- S. Tobin-Hochstadt and M. Felleisen. Interlanguage migration: From scripts to programs. In *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA ’06*, page 964–974, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 159593491X. doi:[10.1145/1176617.1176755](https://doi.org/10.1145/1176617.1176755). URL <https://doi.org/10.1145/1176617.1176755>.
- J. Tov and R. Pucella. Stateful contracts for affine types. In *Programming Languages and Systems, 19th European Symposium on Programming, ESOP 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20–28, 2010. Proceedings*, Mar. 2010.
- V. Trifonov and Z. Shao. Safe and principled language interoperation. In *European Symposium on Programming*, pages 128–146. Springer, 1999.
- A. Turcotte, E. Arteca, and G. Richards. Reasoning About Foreign Function Interfaces Without Modelling the Foreign Language. In A. F. Donaldson, editor, *33rd European Conference on Object-Oriented Programming (ECOOP 2019)*, volume 134 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:32, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-111-5. doi:[10.4230/LIPIcs.ECOOP.2019.16](https://doi.org/10.4230/LIPIcs.ECOOP.2019.16). URL <http://drops.dagstuhl.de/opus/volltexte/2019/10808>.

- P. Wang, S. Cuellar, and A. Chlipala. Compiler verification meets cross-language linking via data abstraction. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, page 675–690, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450325851. doi:[10.1145/2660193.2660201](https://doi.org/10.1145/2660193.2660201). URL <https://doi.org/10.1145/2660193.2660201>.
- A. Wright and M. Felleisen. A syntactic approach to type soundness. *Inf. Comput.*, 115(1):38–94, nov 1994. ISSN 0890-5401. doi:[10.1006/inco.1994.1093](https://doi.org/10.1006/inco.1994.1093). URL <https://doi.org/10.1006/inco.1994.1093>.
- T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One vm to rule them all. In *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, Onward! 2013, page 187–204, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450324724. doi:[10.1145/2509578.2509581](https://doi.org/10.1145/2509578.2509581). URL <https://doi.org/10.1145/2509578.2509581>.

Part V
APPENDICES

A

VALUE INTEROPERABILITY: MUTABLE REFERENCES

Lemma A.0.1 (Irreducible Configurations Have Empty Programs). *If $\langle H ; S ; P \rangle \not\rightarrow$, then $P = \cdot$.*

Proof. We will prove the contrapositive: if there exist i, P' such that $P = i, P'$, then $\langle H ; S ; P \rangle \rightarrow \langle H^* ; S^* ; P^* \rangle$. This can be demonstrated by a trivial case analysis on H , S , and i , because the dynamics of StackLang are defined so that there is a reduction rule for every possible configuration with a non-empty program. \square

Lemma A.0.2 (Prefix Termination). *If $\langle H ; S ; P \rangle \xrightarrow{j} \langle H' ; S' ; P' \rangle \not\rightarrow$ and $\langle H ; S ; P \rangle \xrightarrow{*} \langle H_\bullet ; S_\bullet ; P_\bullet, P_o \rangle$, then $\langle H_\bullet ; S_\bullet ; P_\bullet \rangle \xrightarrow{j} \langle H'_\bullet ; S'_\bullet ; \cdot \rangle \not\rightarrow$ for some $H'_\bullet, S'_\bullet, j_\bullet \leq j$.*

Proof. There is a constructive proof using induction, but here, we will sketch an intuitive proof by contradiction.

If $\langle H_\bullet ; S_\bullet ; P_\bullet \rangle$ does not step to a stuck configuration in some $j_\bullet \leq j$ steps, then $\langle H_\bullet ; S_\bullet ; P_\bullet \rangle$ runs for at least $j + 1$ steps. Because StackLang is deterministic, we can then construct the reduction sequence

$$\begin{aligned} \langle H ; S ; P \rangle &\xrightarrow{*} \langle H_\bullet ; S_\bullet ; P_\bullet, P_o \rangle \\ &\xrightarrow{j+1} \langle H'_\bullet ; S'_\bullet ; P'_\bullet, P_o \rangle \\ &\xrightarrow{*} \langle H ; S' ; P' \rangle \\ &\not\rightarrow \end{aligned}$$

which is longer than j , contradicting the premise.

Finally, if $\langle H_\bullet ; S_\bullet ; P_\bullet \rangle \xrightarrow{j} \langle H'_\bullet ; S'_\bullet ; P'_\bullet \rangle \not\rightarrow$, then by Lemma A.0.1, $P'_\bullet = \cdot$, which suffices to finish the proof. \square

Note that when applying Lemma A.0.2, we sometimes leave P_o implicit.

Lemma A.0.3 (World Extension).

1. If $(W_1, v) \in \mathcal{V}[\tau]$ and $W_1 \sqsubseteq W_2$ then $(W_2, v) \in \mathcal{V}[\tau]$
2. If $(W_1, \gamma) \in \mathcal{G}[\Gamma]$ and $W_1 \sqsubseteq W_2$ then $(W_2, \gamma) \in \mathcal{G}[\Gamma]$

Proof.

1. By induction on τ . The only interesting cases are:

- If $(W_1, \text{thunk lam } x.P) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]$ and $(W_1 \sqsubseteq W_2)$ then $(W_2, \text{thunk lam } x.P) \in \mathcal{V}[\tau_2 \rightarrow \tau_2]$.

Expanding the definition of $\mathcal{V}[\tau_1 \rightarrow \tau_2]$ in the goal, we are to show that

$$(W', [x \mapsto v]P) \in \mathcal{E}[\tau_2]$$

given arbitrary $W' \sqsupseteq W_2$ and v such that $(W', v) \in \mathcal{V}[\tau_1]$. We have that $W_1 \sqsubseteq W_2$ and $W_2 \sqsubset W'$ so $W_1 \sqsubset W'$ by Lemma A.0.4. Then we finish by expanding the definition of $\mathcal{V}[\tau_1 \rightarrow \tau_2]$ in the premise and specializing as appropriate.

- If $(W_1, \ell) \in \mathcal{V}[\text{ref } \tau]$ and $(W_1 \sqsubseteq W_2)$ then $(W_2, \ell) \in \mathcal{V}[\text{ref } \tau]$. Expanding the definition of $\mathcal{V}[\text{ref } \tau]$ in the goal, we are to show that

$$W_2.\Psi(\ell) = \lfloor \mathcal{V}[\tau] \rfloor_{W2.k}$$

Expanding the definition of $\mathcal{V}[\text{ref } \tau]$ in the premise, we have

$$W_1.\Psi(\ell) = \lfloor \mathcal{V}[\tau] \rfloor_{W1.k}$$

Expanding the definition of \sqsubseteq and specializing where appropriate, we have that

$$W_2.k \leq W_1.k \wedge \lfloor W_1.\Psi(\ell) \rfloor_{W2.k} = \lfloor W_2.\Psi(\ell) \rfloor_{W2.k}$$

Then we finish by substituting $\lfloor \mathcal{V}[\tau] \rfloor_{W1.k}$ for $W_1.\Psi(\ell)$ and expanding the definition of $\lfloor \cdot \rfloor$, noting in particular that for any world W , $\lfloor W.\Psi(l) \rfloor_{W.k} = W.\Psi(l)$.

2. By induction, appealing to the previous case where appropriate.

□

Lemma A.0.4 (World Extension Transitive).

If $W_1 \sqsubseteq W_2$ and $W_2 \sqsubseteq W_3$ then $W_1 \sqsubseteq W_3$.

Proof. Suppose $(k_1, \Psi_1) \sqsubseteq (k_2, \Psi_2)$ and $(k_2, \Psi_2) \sqsubseteq (k_3, \Psi_3)$. Unfolding the definition of \sqsubseteq in the goal, we are to show that

$$k_3 \leq k_1 \wedge \lfloor \Psi_1(\ell) \rfloor_{k_3} = \lfloor \Psi_3(\ell) \rfloor_{k_3}$$

given arbitrary $\ell \in \text{dom}(\Psi_1)$. Unfolding in the premises, we have that

$$\begin{aligned} k_2 &\leq k_1 \wedge \lfloor \Psi_1(\ell) \rfloor_{k_2} = \lfloor \Psi_2(\ell) \rfloor_{k_2} \wedge \\ k_3 &\leq k_2 \wedge \lfloor \Psi_2(\ell) \rfloor_{k_3} = \lfloor \Psi_3(\ell) \rfloor_{k_3} \end{aligned}$$

where on the second line we appeal to the fact that $\text{dom}(\Psi_1) \subseteq \text{dom}(\Psi_2) \subseteq \text{dom}(\Psi_3)$ by definition of \sqsubseteq . For the left disjunct, we have

$$k_3 \leq k_2 \leq k_1$$

by transitivity of \leq . For the right disjunct, it is sufficient to show that

$$\lfloor \Psi_1(\ell) \rfloor_{k_3} = \lfloor \Psi_2(\ell) \rfloor_{k_3}$$

because $=$ is transitive. Expanding the definition of $\lfloor \cdot \rfloor$, we are to show that

$$\{(W, v) \mid (W, v) \in \Psi_1(\ell) \wedge W.k < k_3\} = \{(W, v) \mid (W, v) \in \Psi_2(\ell) \wedge W.k < k_3\}$$

and we have that

$$\{(W, v) \mid (W, v) \in \Psi_1(\ell) \wedge W.k < k_2\} = \{(W, v) \mid (W, v) \in \Psi_2(\ell) \wedge W.k < k_2\}$$

Since $k_3 \leq k_2$, $k < k_2$ if $k < k_3$, so we are done. \square

Lemma A.0.5 (Later Heaps). *If $H : W$ then $H : \triangleright W$.*

Proof. Suppose $H : (k, \Psi)$. Expanding the definition of \triangleright , we are to show that

$$H : (k - 1, \lfloor \Psi \rfloor_{k-1})$$

Expanding the definition of $:$, \triangleright , and $\lfloor \cdot \rfloor$, we are to show that

$$((k - 2, \lfloor \Psi \rfloor_{k-2}), v) \in R \wedge k - 2 < k - 1$$

for some ℓ, v, R such that $\Psi(\ell) = R$ and $H(\ell) = v$. The right disjunct is trivial, so we are to show the left disjunct. Expanding the definition of $:$, \triangleright , and $\lfloor \cdot \rfloor$ in the premise and specializing where appropriate, we have that

$$((k - 1, \lfloor \Psi \rfloor_{k-1}), v) \in R$$

Then since $R \in \text{Typ}$ and $(k - 1, \lfloor \Psi \rfloor_{k-1}) \sqsubseteq (k - 2, \lfloor \Psi \rfloor_{k-2})$, $((k - 2, \lfloor \Psi \rfloor_{k-2}), v) \in R$ by definition of Typ_n . \square

Lemma A.0.6 (Value Lifting). *If $(W, v) \in \mathcal{V}[\tau]$, then $(W, \text{push } v) \in \mathcal{E}[\tau]$.*

Proof. Expanding the definition of $\mathcal{E}[\tau]$, we are to show that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\tau]) \quad (3)$$

given arbitrary $W, v, H: W, S, H', S', j < W.k$ such that $(W, v) \in \mathcal{V}[\tau]$ and

$$\langle H ; S ; \text{push } v \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

By the operational semantics of StackLang, we have that $j = 1$, $H' = H$, and $S' = S, v$. Then we have the right disjunct of (3) by taking $v = v$, $W' = \triangleright W$ and appealing to Lemmas A.0.3, A.0.5. \square

Lemma A.0.7 (Compat ()).

$$\llbracket \Gamma; \Gamma \vdash () : \text{unit} \rrbracket$$

Proof. Expanding the definition of $\llbracket \cdot \rrbracket$ and \cdot^+ , we are to show that

$$(W, \text{close}(\gamma_{\text{F}}, \text{close}(\gamma_{\text{F}}, \text{push } 0))) \in \mathcal{E}[\text{unit}]$$

given arbitrary $W, \gamma_{\text{F}}, \gamma_{\text{F}}$ such that $(W, \gamma_{\text{F}}) \in \mathcal{G}[\text{F}]$ and $(W, \gamma_{\text{F}}) \in \mathcal{G}[\text{I}]$. Since $\text{push } 0$ is already closed, the close operators have no effect. Then we are to show that

$$(W, \text{push } 0) \in \mathcal{E}[\text{unit}]$$

Then applying Lemma A.0.6, we are to show that

$$(W, 0) \in \mathcal{V}[\text{unit}]$$

which we have by definition of $\mathcal{V}[\text{unit}]$. \square

Lemma A.0.8 (Compat B).

$$b \in \mathbb{B} \implies \llbracket \Gamma; \Gamma \vdash b : \text{bool} \rrbracket$$

Proof. As in Lemma A.0.7, except that in the case where $b = \text{false}$, $b^+ = \text{push } 1$ and so 1 is used as the witness for v . \square

Lemma A.0.9 (Compat x).

$$\llbracket \Gamma; \Gamma, x : \tau \vdash x : \tau \rrbracket$$

Proof. Expanding the definition of $\llbracket \cdot \rrbracket$ and \cdot^+ , we are to show that

$$(W, \text{close}(\gamma_{\text{F}}, \text{close}(\gamma_{\text{F}}, \text{push } x))) \in \mathcal{E}[\tau]$$

given arbitrary $W, \gamma_{\text{F}}, \gamma_{\text{F}}$ such that $(W, \gamma_{\text{F}}) \in \mathcal{G}[\text{F}]$ and $(W, \gamma_{\text{F}}) \in \mathcal{G}[\Gamma, x : \tau]$. Expanding the definition of $\mathcal{G}[\cdot]$, we have that

$$\gamma_{\text{F}} = \gamma[x \mapsto v] \wedge (W, v) \in \mathcal{V}[\tau] \wedge (W, \gamma) \in \mathcal{G}[\Gamma]$$

for some γ, v . Since $\gamma_{\Gamma}(x) = v$ and v is closed, we are to show that

$$(W, \text{push } v) \in \mathcal{E}[\![\tau]\!]$$

Then applying Lemma A.0.6, we are to show that

$$(W, v) \in \mathcal{V}[\![\tau]\!]$$

which we have by assumption. \square

Lemma A.0.10 (Compat `inl e`).

$$[\![\Gamma; \Gamma \vdash e : \tau_1]\!] \implies [\![\Gamma; \Gamma \vdash \text{inl } e : \tau_1 + \tau_2]\!]$$

Proof. Expanding the definition of $[\cdot]$ and \cdot^+ and pushing substitutions in the goal, we are to show that

$$(W, (\text{close } (\gamma_{\Gamma}, \text{close } (\gamma_{\Gamma}, e^+)), \text{lam } x. (\text{push } [0, x]))) \in \mathcal{E}[\![\tau_1 + \tau_2]\!]$$

given arbitrary $W, \gamma_{\Gamma}, \gamma_{\Gamma}$ such that $(W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]$ and $(W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]$. Expanding the definition of $\mathcal{E}[\cdot]$, we are to show that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\![\tau_1 + \tau_2]\!]) \quad (4)$$

given arbitrary $H: W, S, H', S', j < W.k$ such that

$$\langle H ; S ; (\text{close } (\gamma_{\Gamma}, \text{close } (\gamma_{\Gamma}, e^+)), \text{lam } x. (\text{push } [0, x])) \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle \not\rightarrow$$

The claim is vacuous when $W.k = 0$, so consider $W.k > 0$. Applying Lemma A.0.2, there is $j_e \leq j, H_e, S_e$ such that

$$\langle H ; S ; (\text{close } (\gamma_{\Gamma}, \text{close } (\gamma_{\Gamma}, e^+))) \rangle \xrightarrow{j_e} \langle H_e ; S_e ; \cdot \rangle \not\rightarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

1. $S_e = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_e = H'$.

In this case, we have the left disjunct of (4).

- 2.

$$\exists v_e, W_e \sqsupseteq W. (S_e = S, v_e \wedge H_e : W_e \wedge (W_e, v_e) \in \mathcal{V}[\![\tau_1]\!])$$

and

$$\langle H_e ; S_e ; \text{lam } x. (\text{push } [0, x]) \rangle \xrightarrow{j-j_e} \langle H' ; S' ; \cdot \rangle \not\rightarrow$$

By the operational semantics of StackLang,

$$\begin{aligned} \langle H_e ; S, v_e ; \text{lam } x. (\text{push } [0, x]) \rangle &\xrightarrow{1} \langle H_e ; S ; \text{push } [0, v_e] \rangle \\ &\xrightarrow{1} \langle H_e ; S, [0, v_e] ; \cdot \rangle \end{aligned}$$

so $H' = H_e$ and $S' = S, [0, v_e]$. Then we show the right disjunct of (4) by taking $v = [0, v_e]$ and $W' = \triangleright^2 W_e$, noting that $W \sqsubseteq W_e \sqsubseteq W'$ by Lemma A.0.4. All that remains is to show that $(W', [0, v_e]) \in \mathcal{V}[\tau_1 + \tau_2]$, which, by definition, requires $(W', v_e) \in \mathcal{V}[\tau_1]$. Recall that $(W_e, v_e) \in \mathcal{V}[\tau_1]$. Then simply apply Lemmas A.0.3, A.0.5.

□

Lemma A.0.11 (Compat `inr e`).

$$[\Gamma; \Gamma \vdash e : \tau_2] \implies [\Gamma; \Gamma \vdash \text{inr } e : \tau_1 + \tau_2]$$

Proof. As in Lemma A.0.10, exchanging τ_1, τ_2 and 0, 1 where appropriate.

□

Lemma A.0.12 (Compat `if`).

$$[\Gamma; \Gamma \vdash e : \text{bool}] \wedge [\Gamma; \Gamma \vdash e_1 : \tau] \wedge [\Gamma; \Gamma \vdash e_2 : \tau] \implies [\Gamma; \Gamma \vdash \text{if } e \ e_1 \ e_2 : \tau]$$

Proof. Expanding the definition of $[\cdot]$ and \cdot^+ in the goal and pushing substitutions, we are to show that

$$\begin{aligned} (W, (\text{close } (\gamma_\Gamma, \text{close } (\gamma_\Gamma, e^+)), \text{ if0 close } (\gamma_\Gamma, \text{close } (\gamma_\Gamma, e_1^+)) \ \text{close } (\gamma_\Gamma, \text{close } (\gamma_\Gamma, e_2^+)))) \\ \in \mathcal{E}[\tau] \end{aligned}$$

given arbitrary $W, \gamma_\Gamma, \gamma_\Gamma$ such that $(W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]$ and $(W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]$. Expanding the definition of $\mathcal{E}[\cdot]$, we are to show that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\tau]) \quad (5)$$

given arbitrary $H: W, S, H', S', j < W.k$ such that

$$\langle H ; S ; \text{close } (\gamma_\Gamma, \text{close } (\gamma_\Gamma, e^+)), \text{ if0 } (\dots) (\dots) \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle \not\rightarrow$$

The claim is vacuous when $W.k = 0$, so consider $W.k > 0$. Applying Lemma A.0.2, there is $j_e \leq j, H_e, S_e$ such that

$$\langle H ; S ; \text{close } (\gamma_\Gamma, \text{close } (\gamma_\Gamma, e^+)) \rangle \xrightarrow{j_e} \langle H_e ; S_e ; \cdot \rangle \not\rightarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

1. $S_e = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_e = H'$.

In this case, we have the left disjunct of (5).

2.

$$\exists v_e, W_e \sqsupseteq W. (S_e = S, v_e \wedge H_e : W_e \wedge (W_e, v_e) \in \mathcal{V}[\text{bool}])$$

and

$$\langle H_e ; S_e ; \text{if } 0 \text{ close } (\gamma_{\Gamma}, \text{close } (\gamma_{\Gamma}, e_1^+)) \text{ close } (\gamma_{\Gamma}, \text{close } (\gamma_{\Gamma}, e_2^+)) \rangle \xrightarrow{j \dashv j_e} \langle H' ; S' ; \cdot \rangle \not\rightarrow$$

Expanding the definition of $\mathcal{V}[\text{bool}]$, we have that

$$v_e = n$$

Without loss of generality, suppose $v_e = n = 0$. Then by the operational semantics of StackLang,

$$\begin{aligned} & \langle H_e ; S_e, 0 ; \text{if } 0 \text{ close } (\gamma_{\Gamma}, \text{close } (\gamma_{\Gamma}, e_1^+)) \text{ close } (\gamma_{\Gamma}, \text{close } (\gamma_{\Gamma}, e_2^+)) \rangle \\ & \xrightarrow{1} \langle H_e ; S ; \text{close } (\gamma_{\Gamma}, \text{close } (\gamma_{\Gamma}, e_1^+)) \rangle \end{aligned}$$

Now, by expanding the definition of $[\cdot]$ and $\mathcal{E}[\cdot]$ in the second premise and specializing where appropriate, we have that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W_e. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\tau])$$

If we have the left disjunct, then we have the left disjunct of (5). If we have the right disjunct, then we have the right disjunct of (5) since $W \sqsubseteq W_e \sqsubseteq W'$ by Lemma A.0.4.

The case in which $v_n = n \neq 0$ proceeds analogously over the third premise, exchanging 0, n where appropriate.

□

Lemma A.0.13 (Compat `match`).

$$\begin{aligned} & [\Gamma; \Gamma \vdash e : \tau_1 + \tau_2] \wedge [\Gamma; \Gamma, x : \tau_1 \vdash e_1 : \tau] \wedge [\Gamma; \Gamma, y : \tau_2 \vdash e_2 : \tau] \\ & \implies [\Gamma; \Gamma \vdash \text{match } e \text{ x}\{e_1\} \text{ y}\{e_2\} : \tau_1 + \tau_2] \end{aligned}$$

Proof. Expanding the definition of $\llbracket \cdot \rrbracket$ and \cdot^+ in the goal and pushing substitutions, we are to show that

$$(W, (\text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e^+)), P, \text{if0 } (\text{lam } x. \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e_1^+))) (\text{lam } y. \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e_2^+)))) \in \mathcal{E}[\tau]$$

where $P = \text{DUP}$, push 1, idx, SWAP, push 0, idx

given arbitrary $W, \gamma_{\Gamma}, \gamma_{\Gamma}$ such that $(W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]$ and $(W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]$. Expanding the definition of $\mathcal{E}[\cdot]$, we are to show that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\tau]) \quad (6)$$

given arbitrary $H: W, S, H', S', j < W.k$ such that

$$\langle H ; S ; \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e^+)), P, \text{if0 } (\text{lam } x. \dots) (\text{lam } y. \dots) \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle$$

The claim is vacuous when $W.k = 0$, so consider $W.k > 0$. Applying Lemma A.0.2, there is $j_e \leq j, H_e, S_e$ such that

$$\langle H ; S ; \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e^+)), \rangle \xrightarrow{j_e} \langle H_e ; S_e ; \cdot \rangle \rightsquigarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

1. $S_e = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_e = H'$.

In this case, we have the left disjunct of (6).

- 2.

$$\exists v_e, W_e \sqsupseteq W. (S_e = S, v_e \wedge H_e : W_e \wedge (W_e, v_e) \in \mathcal{V}[\tau_1 + \tau_2])$$

and

$$\langle H_e ; S_e ; P, \text{if0 } (\text{lam } x. \dots) (\text{lam } y. \dots) \rangle \xrightarrow{j-j_e} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

Expanding the definition of $\mathcal{V}[\tau_1 + \tau_2]$, we have that

$$(\exists v_1. v_e = [0, v_1] \wedge v_1 \in \mathcal{V}[\tau_1]) \vee (\exists v_2. v_e = [1, v_2] \wedge v_2 \in \mathcal{V}[\tau_2])$$

Without loss of generality, suppose we have the left disjunct. Then by the operational semantics of StackLang,

$$\begin{aligned} & \langle H_e ; S_e, [0, v_1] ; P, \text{if0 } (\text{lam } x. \dots) (\text{lam } y. \dots) \rangle \\ & \xrightarrow{11} \langle H_e ; S, v_1, 0 ; \text{if0 } (\text{lam } x. \dots) (\text{lam } y. \dots) \rangle \\ & \xrightarrow{1} \langle H_e ; S, v_1 ; (\text{lam } x. \text{close}(\gamma_T, \text{close}(\gamma_T, e_1^+))) \rangle \\ & \xrightarrow{1} \langle H_e ; S ; \text{close}(\gamma_T, \text{close}(\gamma_{T,x:\tau_1}[x \mapsto v_1], e_1^+)) \rangle \end{aligned}$$

where in the last step we push the substitution inside γ_T . Now, by expanding the definition of $\llbracket \cdot \rrbracket$ and $\mathcal{E}\llbracket \cdot \rrbracket$ in the second premise and specializing where appropriate, we have that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W_e. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}\llbracket \tau \rrbracket)$$

If we have the left disjunct, then we have the left disjunct of (6). If we have the right disjunct, then we have the right disjunct of (6) since $W \sqsubseteq W_e \sqsubseteq W'$ by Lemma A.0.4.

The case in which $v_e = [1, v_2]$ proceeds analogously over the third premise, exchanging τ_1, τ_2 and 0, 1 and x, y where appropriate.

□

Lemma A.0.14 (Compat (e_1, e_2)).

$$\llbracket \Gamma; \Gamma \vdash e_1 : \tau_1 \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_2 : \tau_2 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2 \rrbracket$$

Proof. Expanding the definition of $\llbracket \cdot \rrbracket$ and \cdot^+ in the goal and pushing substitutions, we are to show that

$$(W, (\text{close}(\gamma_T, \text{close}(\gamma_T, e_1^+)), \text{close}(\gamma_T, \text{close}(\gamma_T, e_2^+)), \text{lam } x_2. \text{lam } x_1. (\text{push } [x_1, x_2]))) \in \mathcal{E}\llbracket \tau_1 \times \tau_2 \rrbracket$$

given arbitrary W, γ_T, γ_T such that $(W, \gamma_T) \in \mathcal{G}\llbracket \Gamma \rrbracket$ and $(W, \gamma_T) \in \mathcal{G}\llbracket \Gamma \rrbracket$. Expanding the definition of $\mathcal{E}\llbracket \cdot \rrbracket$, we are to show that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}\llbracket \tau_1 \times \tau_2 \rrbracket) \quad (7)$$

given arbitrary $H: W, S, H', S', j < W.k$ such that

$$\langle H ; S ; \text{close}(\gamma_T, \text{close}(\gamma_T, e_1^+)), \text{close}(\gamma_T, \text{close}(\gamma_T, e_2^+)), \dots \rangle \xrightarrow{j} \langle H' ; S' ; \rangle \not\rightarrow$$

The claim is vacuous when $W.k = 0$, so consider $W.k > 0$. Applying Lemma A.0.2, there is $j_1 \leq j, H_1, S_1$ such that

$$\langle H ; S ; \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e_1^+)) \rangle \xrightarrow{j_1} \langle H_1 ; S_1 ; \cdot \rangle \rightsquigarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

1. $S_1 = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_1 = H'$.

In this case, we have the left disjunct of (7).

- 2.

$$\exists v_1, W_1 \sqsupseteq W. (S_1 = S, v_1 \wedge H_1 : W_1 \wedge (W_1, v_1) \in \mathcal{V}[\tau_1])$$

and

$$\langle H_1 ; S_1 ; \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e_2^+)), \text{lam } x_2. \text{lam } x_1. (\text{push} [x_1, x_2]) \rangle \xrightarrow{j-j_1} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

Applying Lemma A.0.2 again, there is $j_2 \leq j - j_1, H_2, S_2$ such that

$$\langle H ; S ; \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e_2^+)) \rangle \xrightarrow{j_2} \langle H_2 ; S_2 ; \cdot \rangle \rightsquigarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

- a) $S_2 = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_2 = H'$.

In this case, we have the left disjunct of (7).

- b)

$$\exists v_2, W_2 \sqsupseteq W_1. (S_2 = S_1, v_2 \wedge H_2 : W_2 \wedge (W_2, v_2) \in \mathcal{V}[\tau_2])$$

and

$$\langle H_2 ; S_2 ; \text{lam } x_2. \text{lam } x_1. (\text{push} [x_1, x_2]) \rangle \xrightarrow{j-j_1-j_2} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

Recall that $S_1 = S, v_1$, so $S_2 = S_1, v_2 = S, v_1, v_2$. Then by the operational semantics of StackLang,

$$\langle H_2 ; S, v_1, v_2 ; \text{lam } x_2. \text{lam } x_1. (\text{push} [x_1, x_2]) \rangle \xrightarrow{3} \langle H_2 ; S, [v_1, v_2] ; \cdot \rangle \rightsquigarrow$$

so $H' = H_2$ and $S' = S, [v_1, v_2]$. Then we show the right disjunct of (7) by taking $v = [v_1, v_2]$ and $W' = \triangleright^3 W_2$, noting that $W \sqsubseteq W_1 \sqsubseteq W_2 \sqsubseteq W'$ by Lemma A.0.4. All that remains is to show that $(W', [v_1, v_2]) \in \mathcal{V}[\tau_1 \times \tau_2]$, which requires $(W', v_1) \in \mathcal{V}[\tau_1]$ and $(W', v_2) \in \mathcal{V}[\tau_2]$. Recall that $(W_1, v_1) \in \mathcal{V}[\tau_1]$ and $(W_2, v_2) \in \mathcal{V}[\tau_2]$. Then simply apply Lemmas A.0.3, A.0.5.

□

Lemma A.0.15 (Compat `fst e`).

$$\llbracket \textcolor{brown}{\Gamma}; \Gamma \vdash e : \tau_1 \times \tau_2 \rrbracket \implies \llbracket \textcolor{brown}{\Gamma}; \Gamma \vdash \mathbf{fst} \ e : \tau_1 \rrbracket$$

Proof. Expanding the definition of $\llbracket \cdot \rrbracket$ and \cdot^+ and pushing substitutions in the goal, we are to show that

$$(W, (\text{close}(\gamma_{\textcolor{brown}{\Gamma}}, \text{close}(\gamma_{\Gamma}, e^+)), \text{push } 0, \text{idx})) \in \mathcal{E}[\llbracket \tau_1 \rrbracket]$$

given arbitrary $W, \gamma_{\textcolor{brown}{\Gamma}}, \gamma_{\Gamma}$ such that $(W, \gamma_{\textcolor{brown}{\Gamma}}) \in \mathcal{G}[\textcolor{brown}{\Gamma}]$ and $(W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]$. Expanding the definition of $\mathcal{E}[\cdot]$, we are to show that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\llbracket \tau_1 \rrbracket]) \quad (8)$$

given arbitrary $H: W, S, H', S', j < W.k$ such that

$$\langle H ; S ; \text{close}(\gamma_{\textcolor{brown}{\Gamma}}, \text{close}(\gamma_{\Gamma}, e^+)), \text{push } 0, \text{idx} \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle \not\rightarrow$$

The claim is vacuous when $W.k = 0$, so consider $W.k > 0$. Applying Lemma A.0.2, there is $j_e \leq j, H_e, S_e$ such that

$$\langle H ; S ; \text{close}(\gamma_{\textcolor{brown}{\Gamma}}, \text{close}(\gamma_{\Gamma}, e^+)) \rangle \xrightarrow{j_e} \langle H_e ; S_e ; \cdot \rangle \not\rightarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

1. $S_e = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_e = H'$.

In this case, we have the left disjunct of (8).

- 2.

$$\exists v_e, W_e \sqsupseteq W. (S_e = S, v_e \wedge H_e : W_e \wedge (W_e, v_e) \in \mathcal{V}[\llbracket \tau_1 \times \tau_2 \rrbracket])$$

and

$$\langle H_e ; S_e ; \text{push } 0, \text{idx} \rangle \xrightarrow{j-j_e} \langle H' ; S' ; \cdot \rangle \not\rightarrow$$

Expanding the definition of $\mathcal{V}[\llbracket \tau_1 \times \tau_2 \rrbracket]$ we have that

$$v_e = [v_1, v_2] \wedge (W_e, v_1) \in \mathcal{V}[\llbracket \tau_1 \rrbracket] \wedge (W_e, v_2) \in \mathcal{V}[\llbracket \tau_2 \rrbracket]$$

Then by the operational semantics of StackLang,

$$\langle H_e ; S, [v_1, v_2] ; \text{push } 0, \text{idx} \rangle \xrightarrow{2} \langle H ; S, v_1 ; \cdot \rangle \not\rightarrow$$

so $H' = H_e$ and $S' = S, v_1$. Then we show the right disjunct of (8) by taking $v = v_1$ and $W' = \triangleright^2 W_e$, noting that $W \sqsubseteq W_e \sqsubseteq W'$

by Lemma A.0.4. All that remains is to show that $(W', v_1) \in \mathcal{V}[\tau_1]$. Recall that $(W_e, v_1) \in \mathcal{V}[\tau_1]$, so simply apply Lemmas A.0.3, A.0.5.

□

Lemma A.0.16 (Compat `snd e`).

$$\llbracket \Gamma; \Gamma \vdash e : \tau_1 \times \tau_2 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash \text{snd } e : \tau_2 \rrbracket$$

Proof. As in Lemma A.0.15, exchanging τ_1, τ_2 and 0, 1 where appropriate.

□

Lemma A.0.17 (Compat $\lambda x : \tau. e$).

$$\llbracket \Gamma; \Gamma, x : \tau_1 \vdash e : \tau_2 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2 \rrbracket$$

Proof. Expanding the definition of $\llbracket \cdot \rrbracket$ and \cdot^+ in the goal and pushing substitutions, we are to show that

$$(W, \text{push}(\text{thunk lam } x. \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e^+)))) \in \mathcal{E}[\tau_1 \rightarrow \tau_2]$$

given arbitrary $W, \gamma_\Gamma, \gamma_\Gamma$ such that $(W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]$ and $(W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]$. Applying Lemma A.0.6, we are to show that

$$(W, (\text{thunk lam } x. \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e^+)))) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]$$

Expanding the definition of $\mathcal{V}[\tau_1 \rightarrow \tau_2]$ and pushing the substitution into γ_Γ , we are to show that

$$(W', \text{close}(\gamma_\Gamma, \text{close}(\gamma_{\Gamma, x:\tau_1}[x \mapsto v], e^+))) \in \mathcal{E}[\tau_2]$$

given arbitrary $W' \sqsupseteq W$ and v such that $(W', v) \in \mathcal{V}[\tau_1]$. We have this by expanding the definition of $\llbracket \cdot \rrbracket$ in the premise and specializing where appropriate. □

Lemma A.0.18 (Compat $e_1 e_2$).

$$\llbracket \Gamma; \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_2 : \tau_1 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash e_1 e_2 : \tau_2 \rrbracket$$

Proof. Expanding the definition of $\llbracket \cdot \rrbracket$ and \cdot^+ in the goal and pushing substitutions, we are to show that

$$(W, \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e_1^+)), \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e_2^+)), \text{SWAP}, \text{call}) \in \mathcal{E}[\tau_2]$$

given arbitrary $W, \gamma_{\text{F}}, \gamma_{\text{T}}$ such that $(W, \gamma_{\text{F}}) \in \mathcal{G}[\text{F}]$ and $(W, \gamma_{\text{T}}) \in \mathcal{G}[\text{T}]$. Expanding the definition of $\mathcal{E}[\cdot]$, we are to show that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\tau_2]) \quad (9)$$

given arbitrary $H: W, S, H', S', j < W.k$ such that

$$\langle H ; S ; \text{close}(\gamma_{\text{F}}, \text{close}(\gamma_{\text{T}}, e_1^+)), \text{close}(\gamma_{\text{F}}, \text{close}(\gamma_{\text{T}}, e_2^+)), \text{SWAP}, \text{call} \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

The claim is vacuous when $W.k = 0$, so consider $W.k > 0$. Applying Lemma A.0.2, there is $j_1 \leq j, H_1, S_1$ such that

$$\langle H ; S ; \text{close}(\gamma_{\text{F}}, \text{close}(\gamma_{\text{T}}, e_1^+)) \rangle \xrightarrow{j_1} \langle H_1 ; S_1 ; \cdot \rangle \rightsquigarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

1. $S_1 = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_1 = H'$.

In this case, we have the left disjunct of (9).

- 2.

$$\exists v_1, W_1 \sqsupseteq W. (S_1 = S, v_1 \wedge H_1 : W_1 \wedge (W_1, v_1) \in \mathcal{V}[\tau_1 \rightarrow \tau_2])$$

and

$$\langle H_1 ; S_1 ; \text{close}(\gamma_{\text{F}}, \text{close}(\gamma_{\text{T}}, e_2^+)), \text{SWAP}, \text{call} \rangle \xrightarrow{j-j_1} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

Applying Lemma A.0.2 again, there is $j_2 \leq j - j_1, H_2, S_2$ such that

$$\langle H ; S ; \text{close}(\gamma_{\text{F}}, \text{close}(\gamma_{\text{T}}, e_2^+)) \rangle \xrightarrow{j_2} \langle H_2 ; S_2 ; \cdot \rangle \rightsquigarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

- a) $S_2 = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_2 = H'$.

In this case, we have the left disjunct of (9).

- b)

$$\exists v_2, W_2 \sqsupseteq W_1. (S_2 = S_1, v_2 \wedge H_2 : W_2 \wedge (W_2, v_2) \in \mathcal{V}[\tau_1])$$

and

$$\langle H_2 ; S_2 ; \text{SWAP}, \text{call} \rangle \xrightarrow{j-j_1-j_2} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

Recall that $(W_1, v_1) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]$, so by expanding the definition of $\mathcal{V}[\tau_1 \rightarrow \tau_2]$ and specializing as appropriate, we have that

$$v_1 = \text{thunk lam } x.P \wedge (W_2, [x \mapsto v_2]P) \in \mathcal{E}[\tau_2]$$

Recall that $S_1 = S, v_1$, so $S_2 = S_1, v_2 = S, \text{thunk lam } x.P, v_2$. Then by the operational semantics of StackLang,

$$\begin{aligned} \langle H_2 ; S, \text{thunk lam } x.P, v_2 ; \text{SWAP}, \text{call} \rangle &\xrightarrow{4} \langle H_2 ; S, v_2, \text{thunk lam } x.P ; \text{call} \rangle \\ &\xrightarrow{1} \langle H_2 ; S, v_2 ; \text{lam } x.P \rangle \\ &\xrightarrow{1} \langle H_2 ; S ; [x \mapsto v_2]P \rangle \\ &\xrightarrow{j-j_1-j_2-6} \langle H' ; S' ; \cdot \rangle \\ &\not\rightarrow \end{aligned}$$

Now, recall that $(W_2, [x \mapsto v_2]P) \in \mathcal{E}[\tau_2]$, so by expanding the definition of $\mathcal{E}[\cdot]$ and specializing where appropriate, we have that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W_2. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\tau_2])$$

If we have the left disjunct, then we have the left disjunct of (9).

If we have the right disjunct, then we have the right disjunct of (9) since $W \sqsubseteq W_1 \sqsubseteq W_2 \sqsubseteq W'$ by Lemma A.0.4.

□

Lemma A.0.19 (Compat ref e).

$$[\Gamma; \Gamma \vdash e : \tau] \implies [\Gamma; \Gamma \vdash \text{ref } e : \text{ref } \tau]$$

Proof. Expanding the definition of $[\cdot]$ and \cdot^+ in the goal and pushing substitutions, we are to show that

$$(W, \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e^+)), \text{alloc}) \in \mathcal{E}[\text{ref } \tau]$$

given arbitrary $W, \gamma_\Gamma, \gamma_\Gamma$ such that $(W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]$ and $(W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]$. Expanding the definition of $\mathcal{E}[\cdot]$, we are to show that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\text{ref } \tau]) \quad (10)$$

given arbitrary $H : W, S, H', S', j < W.k$ such that

$$\langle H ; S ; \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e^+)), \text{alloc} \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle \not\rightarrow$$

The claim is vacuous when $W.k = 0$, so consider $W.k > 0$. Applying Lemma A.0.2, there is $j_e \leq j, H_e, S_e$ such that

$$\langle H ; S ; \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e^+)), \rangle \xrightarrow{j_e} \langle H_e ; S_e ; \cdot \rangle \rightsquigarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

1. $S_e = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_e = H'$.

In this case, we have the left disjunct of (10).

- 2.

$$\exists v_e, W_e \sqsupseteq W. (S_e = S, v_e \wedge H_e : W_e \wedge (W_e, v_e) \in \mathcal{V}[\tau])$$

and

$$\langle H_e ; S_e ; \text{alloc} \rangle \xrightarrow{j - j_e} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

By the operational semantics of StackLang,

$$\langle H_e ; S, v_e ; \text{alloc} \rangle \xrightarrow{1} \langle H_e \uplus \{\ell \mapsto v_e\} ; S, \ell ; \cdot \rangle \rightsquigarrow$$

for some ℓ , so $H' = H_e \uplus \{\ell \mapsto v_e\}$ and $S' = S, \ell$. Then we have the right disjunct of (10) by taking $v = \ell$ and $W' = (W_e.k - 1, \lfloor W_e.\Psi \rfloor_{W_e.k-1} \uplus \{\ell \mapsto \lfloor \mathcal{V}[\tau] \rfloor_{W_e.k-1}\})$, observing that $(W', \ell) \in \mathcal{V}[\text{ref } \tau]$ by definition and $W \sqsubseteq W_e \sqsubseteq W'$ by Lemma A.0.4.

□

Lemma A.0.20 (Compat $!e$).

$$[\Gamma; \Gamma \vdash e : \text{ref } \tau] \implies [\Gamma; \Gamma \vdash !e : \tau]$$

Proof. Expanding the definition of $[\cdot]$ and \cdot^+ in the goal and pushing substitutions, we are to show that

$$(W, \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e^+)), \text{read}) \in \mathcal{E}[\tau]$$

given arbitrary $W, \gamma_{\Gamma}, \gamma_{\Gamma}$ such that $(W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]$ and $(W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]$. Expanding the definition of $\mathcal{E}[\cdot]$, we are to show that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\tau]) \quad (11)$$

given arbitrary $H : W, S, H', S', j < W.k$ such that

$$\langle H ; S ; \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e^+)), \text{read} \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

The claim is vacuous when $W.k = 0$, so consider $W.k > 0$. Applying Lemma A.0.2, there is $j_e \leq j$, H_e, S_e such that

$$\langle H ; S ; \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e^+)), \rangle \xrightarrow{j_e} \langle H_e ; S_e ; \cdot \rangle \rightsquigarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

1. $S_e = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_e = H'$.

In this case, we have the left disjunct of (11).

- 2.

$$\exists v_e, W_e \sqsupseteq W. (S_e = S, v_e \wedge H_e : W_e \wedge (W_e, v_e) \in \mathcal{V}[\text{ref } \tau])$$

and

$$\langle H_e ; S_e ; \text{read} \rangle \xrightarrow{j-j_e} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

Expanding the definition of $\mathcal{V}[\cdot]$, we have that

$$v_e = \ell \wedge W_e.\Psi(\ell) = [\mathcal{V}[\tau]]_{W_e.k}$$

so $H_e = H'_e \uplus \{\ell \mapsto v_\ell\}$ for some v_ℓ such that $v_\ell \in [\mathcal{V}[\tau]]_{W_e.k}$. Then by the operational semantics of StackLang,

$$\langle H'_e \uplus \{\ell \mapsto v_\ell\} ; S, \ell ; \text{read} \rangle \xrightarrow{1} \langle H'_e \uplus \{\ell \mapsto v_\ell\} ; S, v_\ell ; \cdot \rangle \rightsquigarrow$$

so $H' = H'_e \uplus \{\ell \mapsto v_\ell\}$ and $S' = S, v_\ell$. Then we have the right disjunct of (11) by taking $v = v_\ell$ and $W' = \triangleright W_e$, noting that $W \sqsubseteq W_e \sqsubseteq W' = \triangleright W_e$ by Lemma A.0.4.

□

Lemma A.0.21 (Compat $e_1 := e_2$).

$$[\![\Gamma; \Gamma \vdash e_1 : \text{ref } \tau]\!] \wedge [\![\Gamma; \Gamma \vdash e_2 : \tau]\!] \implies [\![\Gamma; \Gamma \vdash e_1 := e_2 : \text{unit}]\!]$$

Proof. Expanding the definition of $[\cdot]$ and \cdot^+ in the goal and pushing substitutions, we are to show that

$$(W, (\text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e_1^+)), \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e_2^+)), \text{write}, \text{push } 0)) \in \mathcal{E}[\text{unit}]$$

given arbitrary $W, \gamma_{\Gamma}, \gamma_{\Gamma}$ such that $(W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]$ and $(W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]$. Expanding the definition of $\mathcal{E}[\cdot]$, we are to show that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\text{unit}]) \quad (12)$$

given arbitrary $\mathsf{H}: W, \mathsf{S}, \mathsf{H}', \mathsf{S}', j < W.k$ such that

$$\langle \mathsf{H}; \mathsf{S}; \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e_1^+)), \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e_2^+)), \text{write}, \text{push } 0 \rangle \xrightarrow{j} \langle \mathsf{H}'; \mathsf{S}'; \cdot \rangle \rightsquigarrow$$

The claim is vacuous when $W.k = 0$, so consider $W.k > 0$. Applying Lemma A.0.2, there is $j_1 \leq j$, $\mathsf{H}_1, \mathsf{S}_1$ such that

$$\langle \mathsf{H}; \mathsf{S}; \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e_1^+)), \rangle \xrightarrow{j_1} \langle \mathsf{H}_1; \mathsf{S}_1; \cdot \rangle \rightsquigarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

1. $\mathsf{S}_1 = \mathsf{S}' = \text{Fail } c \wedge c \in \text{OKERR}$ and $\mathsf{H}_1 = \mathsf{H}'$.

In this case, we have the left disjunct of (12).

- 2.

$$\exists v_1, W_1 \sqsupseteq W. (\mathsf{S}_1 = \mathsf{S}, v_1 \wedge \mathsf{H}_1 : W_1 \wedge (W_1, v_1) \in \mathcal{V}[\text{ref } \tau])$$

and

$$\langle \mathsf{H}_1; \mathsf{S}_1; \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e_2^+)), \text{write}, \text{push } 0 \rangle \xrightarrow{j-j_1} \langle \mathsf{H}'; \mathsf{S}'; \cdot \rangle \rightsquigarrow$$

Expanding the definition of $\mathcal{V}[\text{ref } \tau]$, we have that

$$v_1 = \ell \wedge W_1.\Psi(\ell) = [\mathcal{V}[\tau]]_{W1.k}$$

for some ℓ .

Applying Lemma A.0.2 again, there is $j_2 \leq j - j_1$, $\mathsf{H}_2, \mathsf{S}_2$ such that

$$\langle \mathsf{H}_1; \mathsf{S}_1; \text{close}(\gamma_{\Gamma}, \text{close}(\gamma_{\Gamma}, e_2^+)), \rangle \xrightarrow{j_2} \langle \mathsf{H}_2; \mathsf{S}_2; \cdot \rangle \rightsquigarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

- a) $\mathsf{S}_2 = \mathsf{S}' = \text{Fail } c \wedge c \in \text{OKERR}$ and $\mathsf{H}_2 = \mathsf{H}'$.

In this case, we have the left disjunct of (12).

- b)

$$\exists v_2, W_2 \sqsupseteq W_1. (\mathsf{S}_2 = \mathsf{S}_1, v_2 \wedge \mathsf{H}_2 : W_2 \wedge (W_2, v_2) \in \mathcal{V}[\tau])$$

and

$$\langle \mathsf{H}_2; \mathsf{S}_2; \text{write}, \text{push } 0 \rangle \xrightarrow{j-j_1-j_2} \langle \mathsf{H}'; \mathsf{S}'; \cdot \rangle \rightsquigarrow$$

Recall that $W_1.\Psi(\ell) = [\mathcal{V}[\tau]]_{W1.k}$. Then since $W_1 \sqsubseteq W_2$, we also have that

$W_2.\Psi(\ell) = [\mathcal{V}[\tau]]_{W2.k}$. Then since $\mathsf{H}_2 : W_2$, we may write $\mathsf{H}_2 = \mathsf{H}'_2 \uplus \{\ell \mapsto v_{\ell}\}$ for some v_{ℓ} such that $v_{\ell} \in [\mathcal{V}[\tau]]_{W2.k}$.

Recall that $S_1 = S, \ell$, so $S_2 = S_1, v_2 = S, \ell, v_2$. Then by the operational semantics of StackLang,

$$\langle H'_2 \uplus \{\ell \mapsto v_\ell\} ; S, \ell, v_2 ; \text{write}, \text{push } 0 \rangle \xrightarrow{2} \langle H'_2 \uplus \{\ell \mapsto v_2\} ; S, 0 ; \cdot \rangle$$

so $H' = H'_2 \uplus \{\ell \mapsto v_2\}$ and $S' = S, 0$. Then we show the right disjunct of (10) by taking $v = 0$ and $W' = \triangleright^2 W_2$, noting that $W \sqsubseteq W_1 \sqsubseteq W_2 \sqsubseteq W'$ by Lemma A.0.4. All that remains is to show that $(W', 0) \in \mathcal{V}[\text{unit}]$, which we have by definition.

□

Lemma A.0.22 (Compat $\langle e \rangle_\tau$).

$$\llbracket \Gamma; \Gamma \vdash e : \tau \rrbracket \wedge \tau \sim \tau \implies \llbracket \Gamma; \Gamma \vdash \langle e \rangle_\tau : \tau \rrbracket$$

Proof. Expanding the definition of $\llbracket \cdot \rrbracket$ and \cdot^+ and pushing substitutions in the goal, we are to show that

$$(W, (\text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e^+)), C_{\tau \mapsto \tau})) \in \mathcal{E}[\tau]$$

given arbitrary $W, \gamma_\Gamma, \gamma_\Gamma$ such that $(W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]$ and $(W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]$.

We proceed by appealing to Lemma 4.0.1, which says that it suffices to show that:

$$(W, (\text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e^+)))) \in \mathcal{E}[\tau]$$

But this is exactly what our hypothesis tells us, appropriately applied. □

Lemma A.0.23 (Compat n).

$$\llbracket \Gamma; \Gamma \vdash n : \text{int} \rrbracket$$

Proof. As in Lemma A.0.7, exchanging unit , int and $0, n$ where appropriate. □

Lemma A.0.24 (Compat x).

$$\llbracket \Gamma; \Gamma, x : \tau \vdash x : \tau \rrbracket$$

Proof. As in Lemma A.0.9, exchanging τ, τ where appropriate. □

Lemma A.0.25 (Compat $[e_1, \dots, e_n]$).

$$\llbracket \Gamma; \Gamma \vdash e_1 : \tau \rrbracket \wedge \dots \wedge \llbracket \Gamma; \Gamma \vdash e_n : \tau \rrbracket \implies \llbracket \Gamma; \Gamma \vdash [e_1, \dots, e_n] : [\tau] \rrbracket$$

Proof. As in Lemma A.0.14, exchanging τ_1, τ_2 with $[\tau]$ and generalizing $n \neq 2$ where appropriate. □

Lemma A.0.26 (Compat $\mathbf{e}_1[\mathbf{e}_2]$).

$$\llbracket \Gamma; \Gamma \vdash \mathbf{e}_1 : [\tau] \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash \mathbf{e}_2 : \text{int} \rrbracket \implies \llbracket \Gamma; \Gamma \vdash \mathbf{e}_1[\mathbf{e}_2] : \tau \rrbracket$$

Proof. Expanding the definition of $\llbracket \cdot \rrbracket$ and \cdot^+ and pushing substitutions in the goal, we are to show that

$$(W, (\text{close}(\gamma_{\textcolor{teal}{T}}, \text{close}(\gamma_{\textcolor{brown}{T}}, \mathbf{e}_1^+)), \text{close}(\gamma_{\textcolor{teal}{T}}, \text{close}(\gamma_{\textcolor{brown}{T}}, \mathbf{e}_2^+)), \text{idx})) \in \mathcal{E}[\tau]$$

given arbitrary $W, \gamma_{\textcolor{teal}{T}}, \gamma_{\textcolor{brown}{T}}$ such that $(W, \gamma_{\textcolor{teal}{T}}) \in \mathcal{G}[\Gamma]$ and $(W, \gamma_{\textcolor{brown}{T}}) \in \mathcal{G}[\Gamma]$. Expanding the definition of $\mathcal{E}[\cdot]$, we are to show that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\tau]) \quad (13)$$

given arbitrary $H, S, H', S', j < W.k$ such that

$$\langle H ; S ; \text{close}(\gamma_{\textcolor{brown}{T}}, \text{close}(\gamma_{\textcolor{brown}{T}}, \mathbf{e}_1^+)), \text{close}(\gamma_{\textcolor{teal}{T}}, \text{close}(\gamma_{\textcolor{brown}{T}}, \mathbf{e}_2^+)), \text{idx} \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

The claim is vacuous when $W.k = 0$, so consider $W.k > 0$. Applying Lemma A.0.2, there is $j_1 \leq j, H_1, S_1$ such that

$$\langle H ; S ; \text{close}(\gamma_{\textcolor{brown}{T}}, \text{close}(\gamma_{\textcolor{brown}{T}}, \mathbf{e}_1^+)), \text{close}(\gamma_{\textcolor{teal}{T}}, \text{close}(\gamma_{\textcolor{brown}{T}}, \mathbf{e}_2^+)), \text{idx} \rangle \xrightarrow{j_1} \langle H_1 ; S_1 ; \cdot \rangle \rightsquigarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

1. $S_1 = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_1 = H'$.

In this case, we have the left disjunct of (13).

- 2.

$$\exists v_1, W_1 \sqsupseteq W. (S_1 = S, v_1 \wedge H_1 : W_1 \wedge (W_1, v_1) \in \mathcal{V}[\tau])$$

and

$$\langle H_1 ; S_1 ; \text{close}(\gamma_{\textcolor{teal}{T}}, \text{close}(\gamma_{\textcolor{brown}{T}}, \mathbf{e}_2^+)), \text{idx} \rangle \xrightarrow{j - j_1} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

Expanding the definition of $\mathcal{V}[\tau]$ we have that

$$v_1 = [v'_1, \dots, v'_n] \wedge (W_1, v'_1) \in \mathcal{V}[\tau] \wedge \dots \wedge (W_1, v'_n) \in \mathcal{V}[\tau]$$

Applying Lemma A.0.2 again, there is $j_2 \leq j - j_1, H_2, S_2$ such that

$$\langle H_1 ; S_1 ; \text{close}(\gamma_{\textcolor{teal}{T}}, \text{close}(\gamma_{\textcolor{brown}{T}}, \mathbf{e}_2^+)), \text{idx} \rangle \xrightarrow{j_2} \langle H_2 ; S_2 ; \cdot \rangle \rightsquigarrow$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

a) $S_2 = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_2 = H'$.

In this case, we have the left disjunct of (13).

b)

$$\exists v_2, W_2 \sqsupseteq W_1. (S_2 = S, v_2 \wedge H_2 : W_2 \wedge (W_2, v_2) \in \mathcal{V}[\text{int}])$$

and

$$\langle H_2 ; S_2 ; \text{idx} \rangle \xrightarrow{j_1-j_2} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

Expanding the definition of $\mathcal{V}[\text{int}]$ we have that

$$v_2 = n_i$$

for some n_i .

Recall that $S_1 = S, [v'_1, \dots, v'_n]$, so $S_2 = S_1, n_i = S, [v'_1, \dots, v'_n], n_i$.

Then there are two cases:

- i. $n_i \in [1, \dots, n]$. Then by the operational semantics of StackLang,

$$\langle H_2 ; S, [v'_1, \dots, v'_n], n_i ; \text{idx} \rangle \xrightarrow{1} \langle H_2 ; S, v_{ni} ; \cdot \rangle$$

so $H' = H_2$ and $S' = S, v'_{ni}$. Then we have the right disjunct of (13) by taking $v = v_{ni}$ and $W' = \triangleright W_2$, noting that $W \sqsubseteq W_1 \sqsubseteq W_2 \sqsubseteq W'$ by Lemma A.0.4 All that remains is to show that $(W', v'_{ni}) \in \mathcal{V}[\tau]$. Recall that $(W_1, v'_{ni}) \in \mathcal{V}[\tau]$, so simply apply Lemmas A.0.3, A.0.5.

- ii. $n_i \notin [1, \dots, n]$. Then by the operational semantics of StackLang,

$$\begin{aligned} \langle H_2 ; S, [v'_1, \dots, v'_n], n_i ; \text{idx} \rangle &\xrightarrow{1} \langle H_2 ; S ; \text{error} \rangle \\ &\xrightarrow{1} \langle H_2 ; \text{Fail } c ; \cdot \rangle \end{aligned}$$

so $S' = \text{Fail } c \wedge c \in \text{OKERR}$. Then we have the left disjunct of (13).

□

Lemma A.0.27 (Compat if0).

$$[\Gamma; \Gamma \vdash e : \text{int}] \wedge [\Gamma; \Gamma \vdash e_1 : \tau] \wedge [\Gamma; \Gamma \vdash e_2 : \tau] \implies [\Gamma; \Gamma \vdash \text{if0 } e \ e_1 \ e_2 : \tau]$$

Proof. As in Lemma A.0.18, exchanging `bool`, τ with `int`, τ where appropriate. □

Lemma A.0.28 (Compat $\lambda x : \tau.e$).

$$\llbracket \Gamma; \Gamma, x : \tau_1 \vdash e : \tau_2 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash \lambda x : \tau_1.e : \tau_1 \rightarrow \tau_2 \rrbracket$$

Proof. As in Lemma A.0.17, exchanging τ_1, τ_2 with τ_1, τ_2 where appropriate. \square

Lemma A.0.29 (Compat $e_1 e_2$).

$$\llbracket \Gamma; \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_2 : \tau_1 \rrbracket \implies \llbracket \Gamma; \Gamma \vdash e_1 e_2 : \tau_2 \rrbracket$$

Proof. As in Lemma A.0.18, exchanging τ_1, τ_2 with τ_1, τ_2 where appropriate. \square

Lemma A.0.30 (Compat $e_1 + e_2$).

$$\llbracket \Gamma; \Gamma \vdash e_1 : \text{int} \rrbracket \wedge \llbracket \Gamma; \Gamma \vdash e_2 : \text{int} \rrbracket \implies \llbracket \Gamma; \Gamma \vdash e_1 + e_2 : \text{int} \rrbracket$$

Proof. Expanding the definition of $\llbracket \cdot \rrbracket$ and \cdot^+ and pushing substitutions in the goal, we are to show that

$$(W, (\text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e_1^+)), \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e_2^+)), \text{add})) \in \mathcal{E}[\text{int}]$$

given arbitrary $W, \gamma_\Gamma, \gamma_\Gamma$ such that $(W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]$ and $(W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]$. Expanding the definition of $\mathcal{E}[\cdot]$, we are to show that

$$S' = \text{Fail } c \wedge c \in \text{OKERR} \vee \exists v, W' \sqsupseteq W. (S' = S, v \wedge H' : W' \wedge (W', v) \in \mathcal{V}[\text{int}]) \quad (14)$$

given arbitrary $H: W, S, H', S', j < W.k$ such that

$$\langle H; S; \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e_1^+)), \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e_2^+)), \text{add} \rangle \xrightarrow{j} \langle H'; S'; \cdot \rangle$$

The claim is vacuous when $W.k = 0$, so consider $W.k > 0$. Applying Lemma A.0.2, there is $j_1 \leq j, H_1, S_1$ such that

$$\langle H; S; \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e_1^+)), \text{close}(\gamma_\Gamma, \text{close}(\gamma_\Gamma, e_2^+)), \text{add} \rangle \xrightarrow{j_1} \langle H_1; S_1; \cdot \rangle \dashv$$

Then by expanding $\mathcal{E}[\cdot]$ in the premise and specializing as appropriate, either:

1. $S_1 = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_1 = H'$.

In this case, we have the left disjunct of (14).

- 2.

$$\exists v_1, W_1 \sqsupseteq W. (S_1 = S, v_1 \wedge H_1 : W_1 \wedge (W_1, v_1) \in \mathcal{V}[\text{int}])$$

and

$$\langle H_1 ; S_1 ; \text{close}(\gamma_T, \text{close}(\gamma_{\text{e}}, e_2^+)), \text{add} \rangle \xrightarrow{j-j_1} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

Expanding the definition of $\mathcal{V}[\![\text{int}]\!]$ we have that

$$v_1 = n_1$$

for some n_1 . Applying Lemma A.0.2 again, there is $j_2 \leq j - j_1, H_2, S_2$ such that

$$\langle H_1 ; S_1 ; \text{close}(\gamma_T, \text{close}(\gamma_{\text{e}}, e_2^+)), \text{add} \rangle \xrightarrow{j_2} \langle H_2 ; S_2 ; \cdot \rangle \rightsquigarrow$$

Then by expanding $\mathcal{E}[\![\cdot]\!]$ in the premise and specializing as appropriate, either:

a) $S_2 = S' = \text{Fail } c \wedge c \in \text{OKERR}$ and $H_2 = H'$.

In this case, we have the left disjunct of (14).

b)

$$\exists v_2, W_2 \sqsupseteq W_1. (S_2 = S, v_2 \wedge H_2 : W_2 \wedge (W_2, v_2) \in \mathcal{V}[\![\text{int}]\!])$$

and

$$\langle H_2 ; S_2 ; \text{add} \rangle \xrightarrow{j-j_1-j_2} \langle H' ; S' ; \cdot \rangle \rightsquigarrow$$

Expanding the definition of $\mathcal{V}[\![\text{int}]\!]$ we have that

$$v_2 = n_2$$

for some n_2 .

Recall that $S_1 = S, n_1$, so $S_2 = S_1, n_2 = S, n_1, n_2$. Then by the operational semantics of StackLang,

$$\langle H_2 ; S, n_1, n_2 ; \text{add} \rangle \xrightarrow{1} \langle H_2 ; S, (n_1 + n_2) ; \cdot \rangle \rightsquigarrow$$

so $H' = H_2$ and $S' = S, (n_1 + n_2)$. Then we have the right disjunct of (14) by taking $v = n_1 + n_2$ and $W' = \triangleright W_2$, noting that $W \sqsubseteq W_1 \sqsubseteq W_2 \sqsubseteq W'$ by Lemma A.0.4 and that $(W', n_1 + n_2) \in \mathcal{V}[\![\tau]\!]$ by definition.

□

Lemma A.0.31 (Compat ref e).

$$[\![\Gamma; \Gamma \vdash e : \tau]\!] \implies [\![\Gamma; \Gamma \vdash \text{ref } e : \text{ref } \tau]\!]$$

Proof. As in Lemma A.0.19, exchanging $\tau, \text{ref } \tau$ where appropriate. □

Lemma A.0.32 (Compat $\mathbf{!e}$).

$$\llbracket \Gamma; \mathbf{T} \vdash \mathbf{e} : \mathbf{ref} \ \tau \rrbracket \implies \llbracket \Gamma; \mathbf{T} \vdash \mathbf{!e} : \tau \rrbracket$$

Proof. As in Lemma A.0.20, exchanging τ, τ where appropriate. \square

Lemma A.0.33 (Compat $\mathbf{e}_1 := \mathbf{e}_2$).

$$\llbracket \Gamma; \mathbf{T} \vdash \mathbf{e}_1 : \mathbf{ref} \ \tau \rrbracket \wedge \llbracket \Gamma; \mathbf{T} \vdash \mathbf{e}_2 : \tau \rrbracket \implies \llbracket \Gamma; \mathbf{T} \vdash \mathbf{e}_1 := \mathbf{e}_2 : \mathbf{int} \rrbracket$$

Proof. As in Lemma A.0.21, exchanging τ, τ where appropriate. \square

Lemma A.0.34 (Compat $(\mathbf{e})_\tau$).

$$\llbracket \Gamma; \mathbf{T} \vdash \mathbf{e} : \tau \rrbracket \wedge \tau \sim \tau \implies \llbracket \Gamma; \mathbf{T} \vdash (\mathbf{e})_\tau : \tau \rrbracket$$

Proof. As in A.0.22, exchanging τ, τ where appropriate. \square

B

VALUE INTEROPERABILITY: AFFINE FUNCTIONS

B.1 DYNAMIC LOGICAL RELATION

We first present various supporting lemmas, and then all of the compatibility lemmas.

Lemma B.1.1 (Expression Relation Contains Value Relation).

$$\mathcal{V}[\![\tau]\!]_\rho \subseteq \mathcal{E}[\![\tau]\!]_\rho$$

Proof. All terms in the value relation are irreducible, and thus are trivially in the expression relation. \square

Lemma B.1.2 (Split Substitutions). *For any world W and substitution γ such that*

$$(W, \gamma) \in \mathcal{G}[\![\Omega_1 \uplus \Omega_2]\!]_\rho$$

there exist substitutions γ_1, γ_2 such that $\gamma = \gamma_1 \uplus \gamma_2$ and

$$(W, \gamma_1) \in \mathcal{G}[\![\Omega_1]\!]_\rho$$

and

$$(W, \gamma_2) \in \mathcal{G}[\![\Omega_2]\!]_\rho$$

Moreover, for any $i, j \in \{1, 2\}$, for any $\Gamma; \Omega_j; \Delta; \Gamma \vdash e : \tau$,

$$\gamma^i(e^+) = \gamma_j^i(e^+)$$

and for any $\Delta; \Gamma; \Omega_j \vdash e : \tau$,

$$\gamma^i(e^+) = \gamma_j^i(e^+)$$

Proof. First, we need to show that there exist substitutions γ_1 and γ_2 . This follows from the inductive structure of $\mathcal{G}[\![\Omega]\!]_\rho$, where we can separate the parts that came from $\mathcal{G}[\![\Omega_1]\!]_\rho$ and $\mathcal{G}[\![\Omega_2]\!]_\rho$. The second follows from the fact that the statics means that the rest of the substitution must not occur in the term, and thus $\gamma^i(e^+) = \gamma_1^i(\gamma_2^i(e^+)) = \gamma_1^i(e^+)$ (for example). \square

Lemma B.1.3 (World Extension).

1. If $(W_1, v_1, v_2) \in \mathcal{V}[\![\tau]\!]_\rho$ and $W_1 \sqsubseteq W_2$ then $(W_2, v_1, v_2) \in \mathcal{V}[\![\tau]\!]_\rho$

2. If $(W_1, \gamma_1, \gamma_2) \in \mathcal{G}[\Gamma]_\rho$ and $W_1 \sqsubseteq W_2$ then $(W_2, \gamma_1, \gamma_2) \in \mathcal{G}[\Gamma]_\rho$

Proof. We note that world extension allows three things: the step index to decrease, the heap typing to add bindings (holding all existing bindings at same relation, module decreasing step index), and add flag references (ensuring existing flag references can go from UNUSED to USED, but not the other way). In all cases, this is straightforward based on the definition (relying on Lemma B.1.4 in some cases). \square

Lemma B.1.4 (World Extension Transitive). *If $W_1 \sqsubseteq W_2$ and $W_2 \sqsubseteq W_3$ then $W_1 \sqsubseteq W_3$.*

Proof. Straightforward based on the definition. \square

Lemma B.1.5 (Heaps in Later World). *For any $W \in \text{World}$ and $H_1, H_2 : W$, it holds that $H_1, H_2 : \triangleright W$.*

Proof. Since heap typings map to relations that are by definition closed under world extension, and world extension cannot remove locations, only restrict them to future step indices, this holds by definition. \square

Lemma B.1.6 (Logical Relations for `Miniml` in *Typ*). *For any Δ , $\rho \in \mathcal{D}[\Delta]$, and τ , if $\Delta \vdash \tau$, then $\mathcal{V}[\tau]_\rho \in \text{Typ}$.*

Proof. By the definition of *Typ*, it suffices to show, for all natural numbers n , $[\mathcal{V}[\tau]_\rho]_n \in \text{Typ}_n$. This requires us to show two things: first, that it is in $\mathbf{2}^{\text{AtomVal}^n}$, and second that it is closed under world extension. The latter holds by Lemma B.1.3. For the former, we note that we are required to show that the worlds are in World_n , which holds by definition. \square

Lemma B.1.7 (Compositionality). $(W, v_1, v_2) \in \mathcal{V}[\tau]_{\rho[\alpha \mapsto \mathcal{V}[\tau']_\rho]} \iff (W, v_1, v_2) \in \mathcal{V}[\tau'/\alpha]_\rho$

Proof. It suffices to show $\mathcal{V}[\tau]_{\rho[\alpha \mapsto \mathcal{V}[\tau']_\rho]} = \mathcal{V}[\tau'/\alpha]_\rho$, which we will do by induction on τ . We show the interesting cases:

CASE $\tau = \alpha$.

$$\begin{aligned} \mathcal{V}[[\alpha \mapsto \tau']\alpha]_\rho &= \mathcal{V}[\tau']_\rho && (\text{by sub}) \\ &= \rho[\alpha \mapsto \mathcal{V}[\tau']_\rho](\alpha) && (\text{by lookup}) \\ &= \mathcal{V}[\alpha]_{\rho[\alpha \mapsto \mathcal{V}[\tau']_\rho]} && (\text{by def } \mathcal{V}[\cdot].) \end{aligned}$$

CASE $\tau = \beta \neq \alpha$.

$$\begin{aligned}\mathcal{V}[[\alpha \mapsto \tau']\beta]_\rho &= \mathcal{V}[\beta]_\rho && (\text{by sub}) \\ &= \rho(\beta) && (\text{by def } \mathcal{V}[\cdot].) \\ &= \rho[\alpha \mapsto \mathcal{V}[\tau']_\rho](\beta) && (\text{by lookup}) \\ &= \mathcal{V}[\beta]_{\rho[\alpha \mapsto \mathcal{V}[\tau']_\rho]} && (\text{by def } \mathcal{V}[\cdot].)\end{aligned}$$

The other cases are straightforward by expanding the definitions of $\mathcal{V}[\cdot]., \mathcal{E}[\cdot].$ and applying the induction hypotheses. \square

Lemma B.1.8 (Expression Relation for Closed Types). *For any MiniML type τ where $\cdot \vdash \tau$ and any ρ ,*

$$\mathcal{E}[\tau]_\rho = \mathcal{E}[\tau].$$

Proof. Since $\mathcal{E}[\tau]_\rho$ is defined in terms of $\mathcal{V}[\tau]_\rho$, this proof is analogous to Lemma B.1.7, though since what we are substituting is not used, the interpretation can be arbitrary. \square

Lemma B.1.9 (Closing MiniML Terms). *For any MiniML term e where $\Gamma; \Omega; \Delta; \Gamma \vdash e : \tau$, for any $W, \gamma_\Gamma, \gamma_\Omega, \rho$ where $\rho \in \mathcal{D}[\Delta]$, $(W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho$, $(W, \gamma_\Omega) \in \mathcal{G}[\Omega].$, and $(W, \gamma_\Omega) \in \mathcal{G}[\Omega].$, it holds that*

$$\gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e^+)))$$

and

$$\gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e^+)))$$

are closed terms.

Proof. Since free variables are compiled to free variables, and no other free variables are introduced via compilation, this follows trivially from the structure of $\mathcal{G}[\Gamma]_\rho$. \square

Lemma B.1.10 (Closing Affi Terms). *For any Affi term e where $\Delta; \Gamma; \Omega \vdash e : \tau$, for any $W, \gamma_\Gamma, \gamma_\Omega, \rho$ where $\rho \in \mathcal{D}[\Delta]$, $(W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho$, $(W, \gamma_\Omega) \in \mathcal{G}[\Omega].$, and $(W, \gamma_\Omega) \in \mathcal{G}[\Omega].$, it holds that*

$$\gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e^+)))$$

and

$$\gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e^+)))$$

are closed terms.

Proof. Since free variables are compiled to free variables, and no other free variables are introduced via compilation, this follows trivially from the structure of $\mathcal{G}[\Gamma]_\rho$. \square

Lemma B.1.11 (Anti-reduction). *If $(W', e'_1, e'_2) \in \mathcal{E}[\tau]_\rho$, then $\forall j \ e_1 \ e_2 \ W \ H_1 \ H_2 \ H'_1 \ H'_2. \ W \sqsubseteq W' \wedge j < W.k \wedge H_1, H_2 : W \wedge \langle H_1, e_1 \rangle \xrightarrow{j} \langle H'_1, e'_1 \rangle \wedge \langle H_2, e_2 \rangle \xrightarrow{*} \langle H'_2, e'_2 \rangle \wedge H'_1, H'_2 : W' \wedge W'.k \geq W.k - j \wedge \text{freevars}(e_1) = \text{freevars}(e_2) = \emptyset \implies (W, e_1, e_2) \in \mathcal{E}[\tau]_\rho$*

Proof. Expanding the expression relation, given

$$\forall H_1, H_2 : W, \ e_1^*, \ H_1^*, \ j' < W.k. \ \langle H_1, e_1 \rangle \xrightarrow{j'} \langle H_1^*, e_1^* \rangle \rightsquigarrow$$

we must show either e_1^* is fail CONV or there exist v_2, H_2^*, W^* such that

$$\langle H_2, e_2 \rangle \xrightarrow{*} \langle H_2^*, v_2 \rangle \wedge W \sqsubseteq W^* \wedge H_1^*, H_2^* : W^* \wedge (W^*, e_1^*, v_2) \in \mathcal{V}[\tau]_\rho$$

By confluence, if $\langle H_1, e_1 \rangle \xrightarrow{j} \langle H'_1, e'_1 \rangle$ and $\langle H_1, e_1 \rangle \xrightarrow{j''} \langle H_1^*, e_1^* \rangle \rightsquigarrow$, then

$$\langle H'_1, e'_1 \rangle \xrightarrow{j'-j} \langle H_1^*, e_1^* \rangle \rightsquigarrow$$

Thus, by applying $(W', e'_1, e'_2) \in \mathcal{E}[\tau]_\rho$, since $j' - j < W.k - j \leq W'.k$, we find either e_1^* is fail CONV, in which case we are done, or there exist v_2, H_2^*, W^* such that

$$\langle H'_2, e'_2 \rangle \xrightarrow{*} \langle H_2^*, v_2 \rangle \wedge W' \sqsubseteq W^* \wedge H_1^*, H_2^* : W^* \wedge (W^*, e_1^*, v_2) \in \mathcal{V}[\tau]_\rho$$

Now, since $W \sqsubseteq W'$ and $W' \sqsubseteq W^*$, we have $W \sqsubseteq W^*$ by Lemma B.1.4. Moreover, since $\langle H_2, e_2 \rangle \xrightarrow{*} \langle H'_2, e'_2 \rangle$ and $\langle H'_2, e'_2 \rangle \xrightarrow{*} \langle H_2^*, v_2 \rangle$, we have $\langle H_2, e_2 \rangle \xrightarrow{*} \langle H_2^*, v_2 \rangle$. This suffices to finish the proof. \square

Lemma B.1.12 (Conversions Evaluation Contexts). $C_{\tau_A \mapsto \tau_B} \in \mathsf{K}$

Proof. By inspection of the conversions, which are either empty or have shape if $[\cdot] \dots$ or $\text{let } x = [\cdot] \dots$. \square

Theorem B.1.13 (Convertibility Soundness). *If $\tau_A \sim \tau_B$ then*

$$\forall (W, e_1, e_2) \in \mathcal{E}[\tau_A]. \implies (W, C_{\tau_A \mapsto \tau_B}(e_1), C_{\tau_A \mapsto \tau_B}(e_2)) \in \mathcal{E}[\tau_B]. \wedge \forall (W, e_1, e_2) \in \mathcal{E}[\tau_B]. \implies (W, C_{\tau_B \mapsto \tau_A}(e_1), C_{\tau_B \mapsto \tau_A}(e_2)) \in \mathcal{E}[\tau_A].$$

Proof. We prove this by simultaneous induction on the structure of the convertibility relation.

unit ~ unit There are two directions to this proof:

$$\forall (W, e_1, e_2) \in \mathcal{E}[\text{unit}]. \implies (W, C_{\text{unit} \leftrightarrow \text{unit}}(e_1), C_{\text{unit} \leftrightarrow \text{unit}}(e_2)) \in \mathcal{E}[\text{unit}].$$

and:

$$\forall (W, e_1, e_2) \in \mathcal{E}[\text{unit}]. \implies (W, C_{\text{unit} \leftrightarrow \text{unit}}(e_1), C_{\text{unit} \leftrightarrow \text{unit}}(e_2)) \in \mathcal{E}[\text{unit}].$$

Both directions are trivially similar to each other, so we will only prove the first direction. Expanding the definition of the convertibility boundaries, we refine this to:

$$\forall (W, e_1, e_2) \in \mathcal{E}[\text{unit}]. \implies (W, e_1, e_2) \in \mathcal{E}[\text{unit}].$$

Since the expression relation is shared between the two languages, this holds because $\mathcal{V}[\text{unit}] = \mathcal{V}[\text{unit}] = \{(W, (), ())\}$.

int ~ bool There are two directions to this proof:

$$\forall (W, e_1, e_2) \in \mathcal{E}[\text{int}]. \implies (W, C_{\text{int} \leftrightarrow \text{bool}}(e_1), C_{\text{int} \leftrightarrow \text{bool}}(e_2)) \in \mathcal{E}[\text{bool}].$$

and:

$$\forall (W, e_1, e_2) \in \mathcal{E}[\text{bool}]. \implies (W, C_{\text{bool} \leftrightarrow \text{int}}(e_1), C_{\text{bool} \leftrightarrow \text{int}}(e_2)) \in \mathcal{E}[\text{int}].$$

Consider the first direction. Expanding the definition of the convertibility boundaries, we refine this to:

$$\forall (W, e_1, e_2) \in \mathcal{E}[\text{int}]. \implies (W, e_1, e_2) \in \mathcal{E}[\text{bool}].$$

Unlike the previous case, we cannot prove both directions by equality of value relations, since our binary relation for $\mathcal{V}[\text{bool}]$ renders all non-zero numbers equal (a unary relation would admit the same proof). However, the first implication does follow, since $\mathcal{V}[\text{int}] \subseteq \mathcal{V}[\text{bool}]$.

Next, consider the second direction. Expanding the convertibility boundaries, we must show:

$$\forall (W, e_1, e_2) \in \mathcal{E}[\text{bool}]. \implies (W, \text{if } e_1 0 1, \text{if } e_2 0 1) \in \mathcal{E}[\text{int}].$$

Expanding the expression relation, we must show that given

$$\forall H_1, H_2: W, e'_1, H'_1, j < W.k. \langle H_1, \text{if } e_1 0 1 \rangle \xrightarrow{j} \langle H'_1, e'_1 \rangle \rightsquigarrow$$

it holds that:

$$\begin{aligned} e'_1 = \text{fail CONV} \vee (\exists v_2 H'_2 W'. \langle H_2, \text{if } e_2 0 1 \rangle \xrightarrow{*} \langle H'_2, v_2 \rangle \\ \wedge W \sqsubseteq W' \wedge H'_1, H'_2 : W' \wedge (W', e'_1, v_2) \in \mathcal{V}[\text{int}]_\rho) \end{aligned}$$

By $(W, e_1, e_2) \in \mathcal{E}[\text{bool}]$, we find that either $\langle H_1, e_1 \rangle$ either steps to fail CONV, in which case $\langle H_1, \text{if } e_1 0 1 \rangle$ takes another step to fail CONV and we are done, or steps to an irreducible configuration $\langle H_1^*, e_1^* \rangle$, in which case $\langle H_2, e_2 \rangle$ steps to an irreducible configuration $\langle H_2^*, e_2^* \rangle$ and there exists some world W' such that $W \sqsubseteq W'$, $H_1^*, H_2^* : W'$, and $(W', e_1^*, e_2^*) \in \mathcal{V}[\text{bool}]$. There are then two cases:

1. $e_1^* = e_2^* = 0$. In this scenario, we have

$$\langle H_1, \text{if } e_1 0 1 \rangle \xrightarrow{*} \langle H_1^*, \text{if } 0 0 1 \rangle \rightarrow \langle H_1^*, 0 \rangle$$

and

$$\langle H_2, \text{if } e_2 0 1 \rangle \xrightarrow{*} \langle H_2^*, \text{if } 0 0 1 \rangle \rightarrow \langle H_2^*, 0 \rangle$$

Then, we have from before that $W \sqsubseteq W'$ and $H_1^*, H_2^* : W'$, and one can easily see that $(W', 0, 0) \in \mathcal{V}[\text{int}]$, which suffices to finish the proof.

2. $e_1^* = n_1$ and $e_2^* = n_2$ with $n_1, n_2 \neq 0$. In this scenario, we have

$$\langle H_1, \text{if } e_1 0 1 \rangle \xrightarrow{*} \langle H_1^*, \text{if } n_1 0 1 \rangle \rightarrow \langle H_1^*, 1 \rangle$$

and

$$\langle H_2, \text{if } e_2 0 1 \rangle \xrightarrow{*} \langle H_2^*, \text{if } n_2 0 1 \rangle \rightarrow \langle H_2^*, 1 \rangle$$

Then, we have from before that $W \sqsubseteq W'$ and $H_1^*, H_2^* : W'$, and one can easily see that $(W', 1, 1) \in \mathcal{V}[\text{int}]$, which suffices to finish the proof.

$\tau_1 \otimes \tau_2 \sim \tau_1 \times \tau_2$ There are two directions to this proof:

$$\begin{aligned} \forall (W, e_1, e_2) \in \mathcal{E}[\tau_1 \otimes \tau_2]. \\ \implies (W, C_{\tau_1 \otimes \tau_2 \mapsto \tau_1 \times \tau_2}(e_1), C_{\tau_1 \otimes \tau_2 \mapsto \tau_1 \times \tau_2}(e_2)) \in \mathcal{E}[\tau_1 \times \tau_2]. \end{aligned}$$

and:

$$\begin{aligned} \forall (W, e_1, e_2) \in \mathcal{E}[\tau_1 \times \tau_2]. \\ \implies (W, C_{\tau_1 \times \tau_2 \mapsto \tau_1 \otimes \tau_2}(e_1), C_{\tau_1 \times \tau_2 \mapsto \tau_1 \otimes \tau_2}(e_2)) \in \mathcal{E}[\tau_1 \otimes \tau_2]. \end{aligned}$$

Both directions are trivially similar to each other, so we will only prove the first direction. Expanding the definition of the convertibility boundaries, we refine this to:

$$\begin{aligned} \forall (W, e_1, e_2) \in \mathcal{E}[\tau_1 \otimes \tau_2]. \implies \\ (W, \text{let } x = e_1 \text{ in } (C_{\tau_1 \mapsto \tau_1}(\text{fst } x), C_{\tau_2 \mapsto \tau_2}(\text{snd } x)), \\ \text{let } x = e_2 \text{ in } (C_{\tau_1 \mapsto \tau_1}(\text{fst } x), C_{\tau_2 \mapsto \tau_2}(\text{snd } x))) \in \mathcal{E}[\tau_1 \times \tau_2]. \end{aligned}$$

By Lemma B.1.11 and the hypothesis $(W, e_1, e_2) \in \mathcal{E}[\tau_1 \otimes \tau_2]$, we can reduce this to proving

$$\begin{aligned} (W', (C_{\tau_1 \mapsto \tau_1}(\text{fst } v_1), C_{\tau_2 \mapsto \tau_2}(\text{snd } v_1)), \\ (C_{\tau_1 \mapsto \tau_1}(\text{fst } v_2), C_{\tau_2 \mapsto \tau_2}(\text{snd } v_2))) \in \mathcal{E}[\tau_1 \times \tau_2]. \end{aligned}$$

where $W' \sqsubseteq W$ and $(W', v_1, v_2) \in \mathcal{V}[\tau_1 \otimes \tau_2]$. Now we again appeal to Lemma B.1.11, relying on Lemma B.1.12 to focus the conversions on values in $\mathcal{V}[\tau_i]$, and Lemma B.1.1 combined with our induction hypothesis to render the result.

$$\boxed{\tau_1 \multimap \tau_2 \sim (\text{unit} \rightarrow \tau_1) \rightarrow \tau_2}$$

There are two directions, we first prove the former implication, that is, that:

$$\begin{aligned} \forall (W, e_1, e_2) \in \mathcal{E}[\tau_1 \multimap \tau_2]. \implies \\ (W, C_{\tau_1 \multimap \tau_2 \mapsto (\text{unit} \rightarrow \tau_1) \rightarrow \tau_2}(e_1), C_{\tau_1 \multimap \tau_2 \mapsto (\text{unit} \rightarrow \tau_1) \rightarrow \tau_2}(e_2)) \\ \in \mathcal{E}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]. \end{aligned}$$

Expanding the definition of the convertibility boundaries, we refine our goal to:

$$\begin{aligned} (W, \text{let } x = e_1 \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}}()) \\ \quad \text{in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(x x_{\text{access}}), \\ \text{let } x = e_2 \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}}()) \\ \quad \text{in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(x x_{\text{access}})) \\ \in \mathcal{E}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]. \end{aligned}$$

We appeal to Lemma B.1.11 to reduce this to

$$\begin{aligned} (W^\dagger, \lambda x_{\text{thnk}}. \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}}()) \\ \quad \text{in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}}), \\ \lambda x_{\text{thnk}}. \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}}()) \\ \quad \text{in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}})) \\ \in \mathcal{E}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]. \end{aligned}$$

where $W \sqsubseteq W^\dagger$ and $(W^\dagger, v_1, v_2) \in \mathcal{V}[\tau_1 \multimap \tau_2]$. Since these are clearly values, what remains to show is that these two values are related at W^\dagger in $\mathcal{V}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]$.

The definition of $\mathcal{V}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]$ says that we need to take any $W^\dagger \sqsubset W'$, v'_1 , and v'_2 that are in $\mathcal{V}[\text{unit} \rightarrow \tau_1]$. and show that

$$(W', [x_{\text{thnk}} \mapsto v'_1] \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} ()) \text{ in} \\ \text{let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}}), \\ [x_{\text{thnk}} \mapsto v'_2] \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} ()) \text{ in} \\ \text{let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}})) \in \mathcal{E}[\tau_2].$$

Where if we substitute, we get:

$$(W', \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(v'_1 ()) \text{ in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}}), \\ \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(v'_2 ()) \text{ in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}})) \\ \in \mathcal{E}[\tau_2].$$

Now we can expand the definition of $\text{once}(\cdot)$, to get:

$$(W', \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(v'_1 ()) \text{ in let } x_{\text{access}} = \\ (\text{let } r_{\text{fresh}} = \text{ref } 1 \text{ in } \lambda_. \{ \text{if } !r_{\text{fresh}} \{ \text{fail CONV} \} \{ r_{\text{fresh}} := 0; x_{\text{conv}} \} \}) \\ \text{in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}}), \\ \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(v'_2 ()) \text{ in let } x_{\text{access}} = \\ (\text{let } r_{\text{fresh}} = \text{ref } 1 \text{ in } \lambda_. \{ \text{if } !r_{\text{fresh}} \{ \text{fail CONV} \} \{ r_{\text{fresh}} := 0; x_{\text{conv}} \} \}) \\ \text{in } C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}})) \\ \in \mathcal{E}[\tau_2].$$

From our induction hypothesis, instantiated with W' , we know that, if they don't run forever or fail, $(W', C_{\tau_1 \mapsto \tau_1}(v'_1()), C_{\tau_1 \mapsto \tau_1}(v'_2()))$ will be in $\mathcal{E}[\tau_1]$. if $(W', v'_1(), v'_2())$ is in $\mathcal{E}[\tau_1]$. Since we got v'_1 and v'_2 from $\mathcal{V}[\text{unit} \rightarrow \tau_1]$, the latter holds, and thus we know the converted terms will eventually run to related values vc_1 and vc_2 at some future world W'' of W' in $\mathcal{V}[\tau_1]$. We can further step, substituting those values and reducing to a future world W''' that has in W''' . Θ a pair of fresh locations (ℓ_1, ℓ_2) pointing to UNUSED. That means, by appeal to Lemma B.1.11, we reduce our obligation to showing:

$$(W''', C_{\tau_2 \mapsto \tau_2}(v_1 (\lambda_. \{ \text{if } !\ell_1 \{ \text{fail CONV} \} \{ \ell_1 := 0; vc_1 \} \})), \\ C_{\tau_2 \mapsto \tau_2}(v_2 (\lambda_. \{ \text{if } !\ell_2 \{ \text{fail CONV} \} \{ \ell_2 := 0; vc_2 \} \}))) \in \mathcal{E}[\tau_2].$$

Our induction hypothesis reduces this to proving that:

$$(W''', v_1(\lambda_.\{\text{if } !\ell_1 \{\text{fail CONV}\} \{\ell_1 := 0; \text{vc}_1\}\}), v_2(\lambda_.\{\text{if } !\ell_2 \{\text{fail CONV}\} \{\ell_2 := 0; \text{vc}_2\}\})) \in \mathcal{E}[\tau_2].$$

If we return to how we got v_1 and v_2 , we know they are in $\mathcal{V}[\tau_1 \multimap \tau_2]$. with world W^\dagger , but via Lemma B.1.3, they are also related under W''' . From that definition, we know that v_i has the form $\lambda a.e_i$, and that:

$$((W^*.k, W^*. \Psi, W^*. \Theta \uplus (\ell_1, \ell_2) \mapsto \text{UNUSED}), \text{close}(\{a \mapsto \text{guard}(v_1^*, \ell_1)\}, e_1), \text{close}(\{a \mapsto \text{guard}(v_2^*, \ell_2)\}, e_2)) \in \mathcal{E}[\tau_2].$$

Given any related values v_1^* and v_2^* at a future world W^* of W''' . If we expand out the definition of $\text{guard}(\cdot)$, we note that it exactly matches the terms that we have, and thus our vc_1 and vc_2 are exactly v_1^* and v_2^* , which we already know are related at $\mathcal{V}[\tau_1]$, and due to Lemma B.1.3, they are related not only at W'' but also at W^* . Thus, we are done with the first direction.

Now we have to prove the other direction, that is, that:

$$\begin{aligned} \forall (W, e_1, e_2) \in \mathcal{E}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]. \implies \\ (W, C_{(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2 \mapsto \tau_1 \multimap \tau_2}(e_1), C_{(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2 \mapsto \tau_1 \multimap \tau_2}(e_2)) \in \mathcal{E}[\tau_1 \multimap \tau_2]. \end{aligned}$$

Expanding the definition of the convertibility boundaries, we refine our goal to:

$$\begin{aligned} (W, \text{let } x = e_1 \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} (\))) \text{ in } C_{\tau_2 \mapsto \tau_2}(x x_{\text{access}}), \\ \text{let } x = e_2 \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} (\))) \text{ in } C_{\tau_2 \mapsto \tau_2}(x x_{\text{access}})) \\ \in \mathcal{E}[\tau_1 \multimap \tau_2]. \end{aligned}$$

We appeal to Lemma B.1.11 to reduce this together

$$\begin{aligned} (W^\dagger, \lambda x_{\text{thnk}}. \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} (\))) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}}), \\ \lambda x_{\text{thnk}}. \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} (\))) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}})) \\ \in \mathcal{E}[\tau_1 \multimap \tau_2]. \end{aligned}$$

where $W \sqsubseteq W^\dagger$ and $(W^\dagger, v_1, v_2) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]$. Since these are clearly values, what remains to show is that they are in $\mathcal{V}[\tau_1 \multimap \tau_2]$.

The definition of $\mathcal{V}[\tau_1 \multimap \tau_2]$. says that we need to take any $W^\dagger \sqsubset W'$, $v'_1, v'_2, \ell_1, \ell_2$ where (W^\dagger, v'_1, v'_2) are in $\mathcal{V}[\tau_1]$. and (ℓ_1, ℓ_2) are not in either $W'. \Psi$ or $W'. \Theta$ and show that

$$(W', [x_{\text{thnk}} \mapsto \text{guard}(v'_1, \ell_1)] \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} ())) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}}), \\ [x_{\text{thnk}} \mapsto \text{guard}(v'_2, \ell_2)] \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} ())) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}})) \\ \in \mathcal{E}[\tau_2].$$

Where if we substitute, we get:

$$(W', \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ())) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}}), \\ \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_2, \ell_2) ())) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}})) \\ \in \mathcal{E}[\tau_2].$$

First, let's expand the definition of once(·):

$$(W', \text{let } x_{\text{access}} = \text{let } r_{\text{fresh}} = \text{ref UNUSED in} \\ \lambda_{-}.\{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ())\} \\ \text{in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}})\} \\ \text{let } x_{\text{access}} = \text{let } r_{\text{fresh}} = \text{ref UNUSED in} \\ \lambda_{-}.\{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_2, \ell_2) ())\} \\ \text{in } C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}})\} \\ \in \mathcal{E}[\tau_2]).$$

From Lemma B.1.11 we can take three steps forward: allocating a new reference (ℓ'_i), substituting it for r_{fresh} , and then substituting all of x_{access} , and thus suffices to show that:

$$(W^\dagger, C_{\tau_2 \mapsto \tau_2}(v_1 (\lambda_{-}.\{\text{if } !\ell'_1 \{\text{fail CONV}\} \{\ell'_1 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ())\}\}), \\ C_{\tau_2 \mapsto \tau_2}(v_2 (\lambda_{-}.\{\text{if } !\ell'_2 \{\text{fail CONV}\} \{\ell'_2 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_2, \ell_2) ())\}\})) \\ \in \mathcal{E}[\tau_2].$$

Where W^\diamond has a new pair of references in $W^\diamond.\Theta$ (set to UNUSED), a smaller step index, but otherwise is identical to W' .

For this, we can appeal to our induction hypothesis, which requires us to show that:

$$(W^\diamond, v_1 (\lambda_{-}.\{\text{if } !\ell'_1 \{\text{fail CONV}\} \{\ell'_1 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ())\}), \\ v_2 (\lambda_{-}.\{\text{if } !\ell'_2 \{\text{fail CONV}\} \{\ell'_2 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_2, \ell_2) ())\})) \\ \in \mathcal{E}[\tau_2].$$

Recalling that v_1 and v_2 came from $\mathcal{V}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]$, we can proceed by appealing to the definition of that relation, which tells us that for any

arguments in $\mathcal{V}[\text{unit} \rightarrow \tau_1]$, the result of substituting will be in $\mathcal{E}[\tau_2]$. It thus remains to show that:

$$(W^*, \lambda_{-}\{\text{if } !\ell'_1 \{\text{fail CONV}\} \{\ell'_1 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ())\}, \\ \lambda_{-}\{\text{if } !\ell'_2 \{\text{fail CONV}\} \{\ell'_2 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_2, \ell_2) ())\}\} \\ \in \mathcal{V}[\text{unit} \rightarrow \tau_1]).$$

Where W^* is some future world of W^\diamond . From the definition of $\mathcal{V}[\text{unit} \rightarrow \tau_1]$, we have to show that substituting () for the unused argument results in terms in $\mathcal{E}[\tau_1]$, at some arbitrary future world W^{**} .

We proceed first by case analysis on whether the affine flags (ℓ'_1, ℓ'_2) have been set to USED, which they can be in a future world. If they have been, we can expand the definition of the expression relation, choose heaps $H_1^{**}, H_2^{**} : W^{**}$, and show that

$$\langle H_1, \text{if } !\ell'_1 \{\text{fail CONV}\} \{\ell'_1 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ())\} \rangle \xrightarrow{2} \langle H_1, \text{fail CONV} \rangle$$

At which point we are done.

Thus, we now consider if (ℓ'_1, ℓ'_2) are still set to UNUSED. If that's the case, we instead appeal to Lemma B.1.11, taking three steps to move into the else branches and update the affine flags to USED. That means we reduce our task to showing that in a world W^{***} , which now has those locations marked used in Θ , we need to show:

$$(W^{***}, C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1)()), C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_2, \ell_2)()) \in \mathcal{E}[\tau_1].$$

We now again appeal to our induction hypothesis, expanding the definition of $\text{guard}(\cdot)$ at the same time to yield the following obligation:

$$(W^{***}, (\lambda_{-}\{\text{if } !\ell_1 \{\text{fail CONV}\} \{\ell_1 := \text{USED}; v'_1\}\})(), \\ (\lambda_{-}\{\text{if } !\ell_2 \{\text{fail CONV}\} \{\ell_2 := \text{USED}; v'_2\}\})() \in \mathcal{E}[\tau_1].$$

We can appeal to Lemma B.1.11 to take one step, eliminating the pointless beta-reduction (for simplicity, we use the same name for the world, even though it is a future world):

$$(W^{***}, \text{if } !\ell_1 \{\text{fail CONV}\} \{\ell_1 := \text{USED}; v'_1\}), \text{if } !\ell_2 \{\text{fail CONV}\} \{\ell_2 := \text{USED}; v'_2\}) \in \mathcal{E}[\tau_1].$$

Now we again do case analysis on whether (ℓ_1, ℓ_2) is USED in $W^{***}.\Theta$. If it is, then, as before, we trivially reduce the left side to failure and are done. If it is not, then we update those affine flags and reduce both sides to the values v'_1 and v'_2 , at a future world W^{final} . Now we knew, originally, that those values were in $\mathcal{V}[\tau_1]$ at world W^\dagger , but since, through many

applications of Lemma B.1.4 and Lemma B.1.3, that also means that they are related at W^{final} , we are finally done.

□

Lemma B.1.14 (Compat `unit`).

$$\Gamma; \Omega; \Delta; \Gamma \vdash () \preceq () : \text{unit}$$

Proof. Expanding the conclusion, given

$$\forall W. \forall \rho \gamma_\Gamma \gamma_\Omega \gamma_\Omega. \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \gamma_\Omega) \in \mathcal{G}[\Omega]. \wedge (W, \gamma_\Omega) \in \mathcal{G}[\Omega].$$

we must show

$$(W, \gamma_\Gamma^1(\gamma_\Omega^1(\textcolor{blue}{()^+}))), \gamma_\Gamma^2(\gamma_\Omega^2(\textcolor{blue}{()^+}))) \in \mathcal{E}[\text{unit}]_\rho$$

$\textcolor{blue}{()^+} = ()$ is a closed term, so the closings have no effect. Ergo,

$$\gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(\textcolor{blue}{()^+}))) = \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(\textcolor{blue}{()^+}))) = ()$$

One can easily see $(W, (), ()) \in \mathcal{V}[\text{unit}]_\rho$, which suffices to show $(W, (), ()) \in \mathcal{E}[\text{unit}]_\rho$ by Lemma B.1.1. This suffices to finish the proof.

□

Lemma B.1.15 (Compat `int`).

$$\Gamma; \Omega; \Delta; \Gamma \vdash \mathbb{Z} \preceq \mathbb{Z} : \text{int}$$

Proof. Expanding the conclusion, given

$$\forall W. \forall \rho \gamma_\Gamma \gamma_\Omega \gamma_\Omega. \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \gamma_\Omega) \in \mathcal{G}[\Omega]. \wedge (W, \gamma_\Omega) \in \mathcal{G}[\Omega].$$

we must show

$$(W, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(\textcolor{blue}{n^+})))), \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(\textcolor{blue}{n^+})))) \in \mathcal{E}[\text{int}]_\rho$$

$n^+ = n$ is a closed term, so the closings have no effect. Ergo,

$$\gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(\textcolor{blue}{n^+}))) = \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(\textcolor{blue}{n^+}))) = n$$

Since $n \in \mathbb{Z}$, one can easily see $(W, n, n) \in \mathcal{V}[\text{int}]_\rho$, which suffices to show $(W, n, n) \in \mathcal{E}[\text{int}]_\rho$ by Lemma B.1.1. This suffices to finish the proof. □

Lemma B.1.16 (Compat `x`).

$$\Delta \vdash \tau \wedge x : \tau \in \Gamma \implies \Gamma; \Omega; \Delta; \Gamma \vdash x \preceq x : \tau$$

Proof. Expanding this conclusion, given

$$\forall W. \forall \rho \gamma_{\Gamma} \gamma_{\Omega}. \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega].$$

we must show

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(x^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(x^+)))) \in \mathcal{E}[\tau]_{\rho}$$

Notice that $x^+ = x$. Then, since $x \notin \Omega$ and $(W, \gamma_{\Omega}) \in \mathcal{G}[\Omega]$, we have

$$\gamma_{\Omega}^1(x) = \gamma_{\Omega}^2(x) = x$$

Next, since $x \notin \Gamma$ and $(W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]$, we have

$$\gamma_{\Gamma}^1(x) = \gamma_{\Gamma}^2(x) = x$$

Finally, since $x : \tau \in \Gamma$ and $(W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]$, there must exist v_1, v_2 such that

$$\gamma_{\Gamma}(x) = (v_1, v_2) \wedge (W, v_1, v_2) \in \mathcal{V}[\tau]_{\rho}$$

Thus,

$$\gamma_{\Gamma}^1(x) = v_1 \wedge \gamma_{\Gamma}^2(x) = v_2$$

Ergo, since $(W, v_1, v_2) \in \mathcal{V}[\tau]_{\rho}$, this suffices to show that

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(x^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(x^+)))) \in \mathcal{V}[\tau]_{\rho}$$

By Lemma B.1.1, $\mathcal{V}[\tau]_{\rho} \subseteq \mathcal{E}[\tau]_{\rho}$, which suffices to finish the proof. \square

Lemma B.1.17 (Compat \times).

$$\begin{aligned} & \Gamma; \Omega_1; \Delta; \Gamma \vdash e_1 \preceq e_1 : \tau_1 \\ & \wedge \Gamma; \Omega_2; \Delta; \Gamma \vdash e_2 \preceq e_2 : \tau_2 \\ \implies & \Gamma; \Omega_1 \uplus \Omega_2; \Delta; \Gamma \vdash (e_1, e_2) \preceq (e_1, e_2) : \tau_1 \times \tau_2 \end{aligned}$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho \gamma_{\Gamma} \gamma_{\Omega}. \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega] \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega_1 \uplus \Omega_2].$$

then

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1((e_1, e_2)^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2((e_1, e_2)^+))) \in \mathcal{E}[\tau_1 \times \tau_2]_{\rho}$$

Notice that both of the expressions have no free variables by Lemma B.1.9. We can push the compiler and substitutions through the pair to refine that to:

$$(W, (\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+))), \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_2^+)))), \\ (\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+)))) \in \mathcal{E}[\tau_1 \times \tau_2]_{\rho}$$

We now appeal to Lemma B.1.15, which combined with our hypotheses and Lemma B.1.1 is sufficient to complete the proof. \square

Lemma B.1.18 (Compat `fst`).

$$\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau_1 \times \tau_2 \implies \Gamma; \Omega; \Delta; \Gamma \vdash \text{fst } e \preceq \text{fst } e : \tau_1$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Omega} \cdot \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega] \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega].$$

then

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\text{fst } e^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\text{fst } e^+)))) \in \mathcal{E}[\tau_1]_{\rho}$$

We can push the compiler and substitutions through `fst` to refine that to:

$$(W, \text{fst } \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+))), \text{fst } (\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+))))) \in \mathcal{E}[\tau_1]_{\rho}$$

We then appeal to Lemma B.1.11, using our hypothesis to first reduce the problem to:

$$(W', \text{fst } v_1, \text{fst } v_2) \in \mathcal{E}[\tau_1]_{\rho}$$

where $W \sqsubseteq W'$ and $(W', v_1, v_2) \in \mathcal{V}[\tau_1 \times \tau_2]_{\rho}$. We can then complete the proof by appealing again to Lemma B.1.11, since we know v_i are pairs and from Lemma B.1.1, their first projections are in $\mathcal{E}[\tau_1]_{\rho}$. \square

Lemma B.1.19 (Compat `snd`).

$$\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau_1 \times \tau_2 \implies \Gamma; \Omega; \Delta; \Gamma \vdash \text{snd } e \preceq \text{snd } e : \tau_2$$

Proof. This proof is essentially identical to that of `fst`. \square

Lemma B.1.20 (Compat `inl`).

$$\Delta \vdash \tau_2 \wedge \Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau_1 \implies \Gamma; \Omega; \Delta; \Gamma \vdash \text{inl } e \preceq \text{inl } e : \tau_1 + \tau_2$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Omega} \cdot \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega] \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega].$$

then

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\text{inl } e^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\text{inl } e^+))) \in \mathcal{E}[\tau_1 + \tau_2]_{\rho}$$

We can push the compiler and substitutions through `inl` to refine that to:

$$(W, \text{inl } \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\text{inl } e^+))), \text{inl } \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\text{inl } e^+))) \in \mathcal{E}[\tau_1 + \tau_2]_{\rho}$$

We complete the proof by appealing to Lemma B.1.11, our hypothesis, and Lemma B.1.1. \square

Lemma B.1.21 (Compat `inr`).

$$\Delta \vdash \tau_1 \wedge \Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau_2 \implies \Gamma; \Omega; \Delta; \Gamma \vdash \text{inr } e \preceq \text{inr } e : \tau_1 + \tau_2$$

Proof. This proof is essentially identical to that of `inl`. \square

Lemma B.1.22 (Compat `match`).

$$\begin{aligned} & \Gamma; \Omega_1; \Delta; \Gamma \vdash e \preceq e : \tau_1 + \tau_2 \\ & \wedge \Gamma; \Omega_2; \Delta; \Gamma[x : \tau_1] \vdash e_1 \preceq e_1 : \tau \\ & \wedge \Gamma; \Omega_2; \Delta; \Gamma[y : \tau_2] \vdash e_2 \preceq e_2 : \tau \\ \implies & \Gamma; \Omega_1 \uplus \Omega_2; \Delta; \Gamma \vdash \text{match } e \{x\{e_1\} y\{e_2\}\} \preceq \text{match } e \{x\{e_1\} y\{e_2\}\} : \tau \end{aligned}$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \cdot \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega_1 \uplus \Omega_2].$$

then

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\text{match } e \{x\{e_1\} y\{e_2\}\}^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\text{match } e \{x\{e_1\} y\{e_2\}\}^+))) \in \mathcal{E}[\tau]_{\rho}$$

We can push the compiler and substitutions through the match to refine that to:

$$(W, \text{match } \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\text{match } e \{x\{e_1\} y\{e_2\}\}^+)) \times \{\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\text{match } e_1 \{y\{e_2\}\}^+))\} y\{\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\text{match } e_2 \{y\{e_2\}\}^+))\}, \\ \text{match } \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\text{match } e \{x\{e_1\} y\{e_2\}\}^+)) \times \{\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\text{match } e_1 \{y\{e_2\}\}^+))\} y\{\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\text{match } e_2 \{y\{e_2\}\}^+))\}), \in \mathcal{E}[\tau]_{\rho}$$

In order to proceed, first notice that, by applying Lemma B.1.2 twice, we find that $\gamma_{\Omega} = \gamma_1 \uplus \gamma_2$ where

$$(W, \gamma_1) \in \mathcal{G}[\Omega_1] \text{ and } (W, \gamma_2) \in \mathcal{G}[\Omega_2].$$

and for any $i \in \{1, 2\}$,

$$\gamma_{\Omega}^i(e^+) = \gamma_1^i(e^+) \text{ and } \gamma_{\Omega}^i(e) = \gamma_2^i(e)$$

Thus, we can rewrite our goal to:

$$(W, \text{match } \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Gamma}^1(\gamma_{\Gamma}^1(e^+)))) \times \{\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_1^+))))\} y\{\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_2^+))))\}, \\ \text{match } \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Gamma}^2(e^+))) \times \{\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Gamma}^2(\gamma_2^2(e_1^+))))\} y\{\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Gamma}^2(\gamma_2^2(e_2^+))))\}, \in \mathcal{E}[\tau]_{\rho})$$

Now we appeal to Lemma B.1.11 to reduce this to

$$(W_1, \text{match } e_1^* \times \{\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_1^+))))\} y\{\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_2^+))))\}, \\ \text{match } e_1^{\dagger} \times \{\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Gamma}^2(\gamma_2^2(e_1^+))))\} y\{\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Gamma}^2(\gamma_2^2(e_2^+))))\}, \in \mathcal{E}[\tau]_{\rho})$$

For some future world W_1 where $W \sqsubseteq W_1$ and $(W_1, e_1^*, e_1^{\dagger}) \in \mathcal{V}[\tau_1 + \tau_2]_{\rho}$.

Given $(W_1, e_1^*, e_1^{\dagger}) \in \mathcal{V}[\tau_1 + \tau_2]_{\rho}$, there must exist v_1^*, v_1^{\dagger} such that either $e_1^* = \text{inl } v_1^*$, $e_1^{\dagger} = \text{inl } v_1^{\dagger}$, and $(W_1, v_1^*, v_1^{\dagger}) \in \mathcal{V}[\tau_1]_{\rho}$ or $e_1^* = \text{inr } v_1^*$, $e_1^{\dagger} = \text{inr } v_1^{\dagger}$, and $(W_1, v_1^*, v_1^{\dagger}) \in \mathcal{V}[\tau_2]_{\rho}$.

First, consider the case where $e_1^* = \text{inl } v_1^*$ and $e_1^{\dagger} = \text{inl } v_1^{\dagger}$; the other case is analogous. We complete the proof by appeal to Lemma B.1.11 using our first hypothesis, noting that the substitution of v_1^* and v_1^{\dagger} satisfy the extended substitution. \square

Lemma B.1.23 (Compat \rightarrow).

$$\Gamma; \Omega; \Delta; \Gamma[x : \tau_1] \vdash e \preceq e : \tau_2 \implies \Gamma; \Omega; \Delta; \Gamma \vdash \lambda x : \tau_1. e \preceq \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Omega}. \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega] \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega].$$

then

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\lambda x : \tau_1. e^+))), \gamma_{\Gamma}^1(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\lambda x : \tau_1. e^+)))) \in \mathcal{E}[\tau_1 \rightarrow \tau_2]_{\rho}$$

We can push the compiler and substitutions through the lambda to refine that to:

$$(W, \lambda x. \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+))), \lambda x. \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))) \in \mathcal{E}[\tau_1 \rightarrow \tau_2]_{\rho}$$

Since these are already values, from Lemma B.1.1 it suffices to show that they are in $\mathcal{V}[\tau_1 \rightarrow \tau_2]_{\rho}$. Consider arbitrary v_1, v_2, W' where $W \sqsubseteq W'$ and $(W', v_1, v_2) \in \mathcal{V}[\tau_1]_{\rho}$. Then, we must show

$$(W', [x \rightarrow v_1] \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+))), [x \rightarrow v_2] \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))) \in \mathcal{E}[\tau_2]_{\rho}$$

Notice that $\gamma_{\Gamma}[x \rightarrow (v_1, v_2)] \in \mathcal{G}[\Gamma[x : \tau_1]]_{\rho}$ because $(W', \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho}$ (by $W \sqsubseteq W'$ and Lemma B.1.3) and $(W', v_1, v_2) \in \mathcal{V}[\tau_1]_{\rho}$. Then, we can instantiate the first induction hypothesis with $W', \gamma_{\Gamma}[x \rightarrow (v_1, v_2)], \gamma_{\Gamma}, \gamma_{\Omega}, \rho$

because $W \sqsubseteq W'$ and $\mathcal{G}[\Gamma]_\rho, \mathcal{G}[\Omega]_., \mathcal{G}[\Omega]_.$ are closed under world extension by Lemma B.1.3. Therefore,

$$(W', \gamma_{\Gamma}[x \rightarrow (v_1, v_2)]^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+))), \gamma_{\Gamma}[x \rightarrow (v_1, v_2)]^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))) \in \mathcal{E}[\tau_2]_\rho$$

Which, after rearranging substitutions is exactly what we needed to show. \square

Lemma B.1.24 (Compat app).

$$\begin{aligned} \Gamma; \Omega_1; \Delta; \Gamma \vdash e_1 \preceq e_1 : \tau_1 \rightarrow \tau_2 \wedge \Gamma; \Omega_2; \Delta; \Gamma \vdash e_2 \preceq e_2 : \tau_1 &\implies \\ \Gamma; \Omega_1 \uplus \Omega_2; \Delta; \Gamma \vdash e_1 e_2 \preceq e_1 e_2 : \tau_2 \end{aligned}$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \cdot \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega]_.. \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega_1 \uplus \Omega_2]..$$

then

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1 e_2^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1 e_2^+)))) \in \mathcal{E}[\tau_2]_\rho$$

We can push the compiler and substitutions through to refine that to:

$$\begin{aligned} (W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+))) \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_2^+))), \\ \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1^+))) \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+)))) \in \mathcal{E}[\tau_2]_\rho \end{aligned}$$

In order to proceed, first notice that, by Lemma B.1.2, $\gamma_{\Omega} = \gamma_1 \uplus \gamma_2$ where

$$(W, \gamma_1) \in \mathcal{G}[\Omega_1].. \text{ and } (W, \gamma_2) \in \mathcal{G}[\Omega_2]..$$

and, for any $i \in \{1, 2\}$

$$\gamma_{\Omega}^i(e_1^+) = \gamma_1^i(e_1^+) \text{ and } \gamma_{\Omega}^i(e_2^+) = \gamma_2^i(e_2^+)$$

Thus, we can rewrite our goal to:

$$\begin{aligned} (W, \gamma_{\Gamma}^1(\gamma_1^1(\gamma_1^1(e_1^+))) \gamma_{\Gamma}^1(\gamma_1^1(\gamma_2^1(e_2^+))), \\ \gamma_{\Gamma}^2(\gamma_2^2(\gamma_2^2(e_1^+))) \gamma_{\Gamma}^2(\gamma_2^2(\gamma_2^2(e_2^+)))) \in \mathcal{E}[\tau_2]_\rho \end{aligned}$$

We proceed by appealing to Lemma B.1.11, using our first hypothesis to reduce our obligation to:

$$\begin{aligned} (W_1, e_1^* \gamma_{\Gamma}^1(\gamma_1^1(\gamma_2^1(e_2^+))), \\ e_1^\dagger \gamma_{\Gamma}^2(\gamma_2^2(\gamma_2^2(e_2^+)))) \in \mathcal{E}[\tau_2]_\rho \end{aligned}$$

Where for $W \sqsubseteq W_1$, $(W_1, e_1^*, e_1^\dagger) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]_\rho$. And then again, using our second hypothesis to reduce our obligation to:

$$(W_2, e_1^* \ e_2^*, e_1^\dagger \ e_2^\dagger) \in \mathcal{E}[\![\tau_2]\!]_\rho$$

Where for $W_1 \sqsubseteq W_2$, $(W_2, e_2^*, e_2^\dagger) \in \mathcal{V}[\![\tau_1]\!]_\rho$.

Then, instantiate $(W_1, e_1^*, e_1^\dagger) \in \mathcal{V}[\![\tau_1 \rightarrow \tau_2]\!]_\rho$ with $e_2^*, e_2^\dagger, \triangleright W_2$. Because $W_1 \sqsubseteq W_2$ and $W_2 \sqsubset \triangleright W_2$, it follows that $W_1 \sqsubset \triangleright W_2$. Moreover, $(\triangleright W_2, e_2^*, e_2^\dagger) \in \mathcal{V}[\![\tau_1]\!]_\rho$ (because $(W_2, e_2^*, e_2^\dagger) \in \mathcal{V}[\![\tau_1]\!]_\rho$ and $W_2 \sqsubseteq \triangleright W_2$), so we find that there exist e_b^*, e_b^\dagger such that

$$e_1^* = \lambda x. e_b^*$$

and

$$e_1^\dagger = \lambda x. e_b^\dagger$$

and

$$(\triangleright W_2, [x \mapsto e_b^*]e_b^*, [x \rightarrow e_b^\dagger]e_b^\dagger) \in \mathcal{E}[\![\tau_2]\!]_\rho$$

By appeal to Lemma B.1.11, this is sufficient to complete the proof. \square

Lemma B.1.25 (Compat \vee).

$$\Gamma; \Omega; \Delta, \alpha; \Gamma \vdash e \preceq e : \tau \implies \Gamma; \Omega; \Delta; \Gamma \vdash \Lambda \alpha. e \preceq \Lambda \alpha. e : \forall \alpha. \tau$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Omega \cdot \rho \in \mathcal{D}[\![\Delta]\!] \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\![\Gamma]\!]_\rho \wedge (W, \gamma_\Omega) \in \mathcal{G}[\![\Omega]\!] \wedge (W, \gamma_\Omega) \in \mathcal{G}[\![\Omega]\!].$$

then

$$(W, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(\Lambda \alpha. e^+))), \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(\Lambda \alpha. e^+)))) \in \mathcal{E}[\![\forall \alpha. \tau]\!]_\rho$$

We can push the compiler and substitutions through the pair to refine that to:

$$(W, \lambda_- \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e^+))), \lambda_- \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e^+)))) \in \mathcal{E}[\![\forall \alpha. \tau]\!]_\rho$$

Since these are already values, by Lemma B.1.1, it suffices to show that

$$(W, \lambda_- \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e^+))), \lambda_- \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e^+)))) \in \mathcal{V}[\![\forall \alpha. \tau]\!]_\rho$$

Consider some arbitrary $R \in Typ$ and W' such that $W \sqsubset W'$. We must prove that

$$(W', \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e^+))), \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e^+)))) \in \mathcal{E}[\![\tau]\!]_{\rho[\alpha \mapsto R]}$$

Since $R \in Typ$ and $\rho \in \mathcal{D}[\![\Delta]\!]$, it follows that $\rho[\alpha \mapsto R] \in \mathcal{D}[\![\Delta, \alpha]\!]$. Thus, we can instantiate the first induction hypothesis with $W', \gamma_\Gamma, \gamma_\Omega, \gamma_\Omega, \rho[\alpha \mapsto R]$,

because $W \sqsubseteq W'$ and $\mathcal{G}[\Gamma]_\rho, \mathcal{G}[\Omega]_\rho$ is closed under world extension by Lemma B.1.3. This suffices to prove the above fact. \square

Lemma B.1.26 (Compat $[\tau/\alpha]$).

$$\Delta \vdash \tau' \wedge \textcolor{brown}{\Gamma}; \Omega; \Delta; \Gamma \vdash e \preceq e : \forall \alpha. \tau \implies \textcolor{brown}{\Gamma}; \Omega; \Delta; \Gamma \vdash e[\tau'] \preceq e[\tau'] : \tau[\tau'/\alpha]$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_\Gamma \gamma_{\textcolor{brown}{\Gamma}} \gamma_\Omega \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \gamma_{\textcolor{brown}{\Gamma}}) \in \mathcal{G}[\Omega] \wedge (W, \gamma_\Omega) \in \mathcal{G}[\Omega].$$

then

$$(W, \gamma_\Gamma^1(\gamma_\Omega^1(e[\tau']^+))), \gamma_\Gamma^2(\gamma_\Omega^2(e[\tau']^+))) \in \mathcal{E}[\tau[\tau'/\alpha]]_\rho$$

We can push the compiler and substitutions through the type application to refine this to:

$$(W, \gamma_\Gamma^1(\gamma_\Omega^1(e^+))), (\gamma_\Gamma^2(\gamma_\Omega^2(e^+))) \in \mathcal{E}[\tau[\tau'/\alpha]]_\rho$$

Expanding the expression relation definition, we find that given

$$\begin{aligned} & \forall H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \langle H_1, \gamma_\Gamma^1(\gamma_\Omega^1(e^+)) \rangle \xrightarrow{j} \langle H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

we must show either $e'_1 = \text{fail CONV}$ or there exist v_2, H'_2, W' such that:

$$\langle H_2, \gamma_\Gamma^2(\gamma_\Omega^2(e^+)) \rangle \xrightarrow{*} \langle H'_2, v_2 \rangle \wedge W \sqsubseteq W' \wedge H'_1, H'_2 : W' \wedge (W', e'_1, v_2) \in \mathcal{V}[\tau[\tau'/\alpha]]_\rho$$

To proceed, we must find what e'_1 is. From the operational semantics, we know the application will run its subexpression using H_1 until it reaches a target value or gets stuck. From the induction hypothesis instantiated with $W, \gamma_\Gamma, \gamma_{\textcolor{brown}{\Gamma}}, \gamma_\Omega, \rho$, we find that:

$$(W, \gamma_\Gamma^1(\gamma_\Omega^1(e^+))), \gamma_\Gamma^2(\gamma_\Omega^2(e^+)) \in \mathcal{E}[\forall \alpha. \tau]_\rho$$

By instantiating this fact with H_1, H_2 , we find that $\langle H_1, \gamma_\Gamma^1(\gamma_\Omega^1(e^+)) \rangle$ either reduces to fail CONV, in which case the entire term reduced to fail CONV, or it will reduce to some $\langle H_1^*, e_1^* \rangle$, in which case the other side with H_2 will reduce to some $\langle H_2^*, e_1^\dagger \rangle$ and

$$(W_1, e_1^*, e_1^\dagger) \in \mathcal{V}[\forall \alpha. \tau]_\rho$$

for some world W_1 where $W \sqsubseteq W_1$ and $H_1^*, H_2^* : W_1$.

Then, we can instantiate this fact with $\mathcal{V}[\tau']_\rho$ and $\triangleright W_1$. (Note that $\mathcal{V}[\tau']_\rho \in \text{Typ}$ by Lemma B.1.6.) Since $W \sqsubset \triangleright W_1$ (as $W \sqsubseteq W_1$ and $W_1 \sqsubset \triangleright W_1$), we find that there exist e_b^*, e_b^\dagger such that

$$e_1^* = \lambda _. e_b^*$$

$$e_1^\dagger = \lambda _. e_b^\dagger$$

and

$$(\triangleright W_1, e_b^*, e_b^\dagger) \in \mathcal{E}[\tau]_{\rho[\alpha \rightarrow \mathcal{V}[\tau']_\rho]}$$

Ergo, by the operational semantics, the original configuration with heap H_1 steps to $\langle H_1^*, \lambda _. e_b^* () \rangle$ and, on the other side, the configuration with H_2 steps to $\langle H_2^*, \lambda _. e_b^\dagger () \rangle$. Next, both of these configurations take a step to $\langle H_1^*, e_b^* \rangle$ and $\langle H_2^*, e_b^\dagger \rangle$, respectively. (Notice that $()$ is not substituted anywhere because the binding in the lambda values are unused.) Next, since $H_1^*, H_2^* : W_1$, by Lemma B.1.5, it follows that $H_1^*, H_2^* : \triangleright W_1$, so we can instantiate the above fact with H_1^*, H_2^* to deduce that either the first configuration steps to fail CONV, in which case the original configuration with H_1 steps to fail CONV, or the first configuration steps to some irreducible configuration $\langle H_1^{**}, e_f^* \rangle$, in which case the second configuration also steps to some irreducible configuration $\langle H_2^{**}, e_f^\dagger \rangle$, and there exists some W_2 where $\triangleright W_1 \sqsubseteq W_2$, $H_1^{**}, H_2^{**} : W_2$, and $(W_2, e_f^*, e_f^\dagger) \in \mathcal{V}[\tau]_{\rho[\alpha \rightarrow \mathcal{V}[\tau']_\rho]}$. Therefore, by Lemma B.1.7, $(W_2, e_f^*, e_f^\dagger) \in \mathcal{V}[\tau[\tau'/\alpha]]_\rho$. Ergo, $e'_1 = e_f^\dagger$, so this suffices to show e'_1 is in the value relation at $\tau[\tau'/\alpha]$ along with the value stepped to by the configuration with H_2 on the other side. Finally, since $W \sqsubseteq W_1$, $W_1 \sqsubseteq \triangleright W_1$, and $\triangleright W_1 \sqsubseteq W_2$, we have $W \sqsubseteq W_2$ (by Lemma B.1.4), which suffices to finish the proof. \square

Lemma B.1.27 (Compat ref).

$$\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau \implies \Gamma; \Omega; \Delta; \Gamma \vdash \text{ref } e \preceq \text{ref } e : \text{ref } \tau$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Delta. \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \gamma_\Omega) \in \mathcal{G}[\Omega]. \wedge (W, \gamma_\Delta) \in \mathcal{G}[\Delta].$$

then

$$(W, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Delta^1(\text{ref } e^+))), \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Delta^2(\text{ref } e^+)))) \in \mathcal{E}[\text{ref } \tau]_\rho$$

We can push the compiler and substitutions through the type application to refine this to:

$$(W, \text{ref } \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Delta^1(e^+))), \text{ref } \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Delta^2(e^+)))) \in \mathcal{E}[\text{ref } \tau]_\rho$$

We appeal to Lemma B.1.11 and our hypothesis to reduce our proof to:

$$(W', \text{ref } v_1, \text{ref } v_2) \in \mathcal{E}[\![\text{ref } \tau]\!]_\rho$$

Where $W \sqsubseteq W'$ and $(W', v_1, v_2) \in \mathcal{V}[\![\tau]\!]_\rho$. At this point, the result follows from another appeal to Lemma B.1.11 and Lemma B.1.1, since each will take a single step to fresh locations that point to values v_i . \square

Lemma B.1.28 (Compat !).

$$\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \text{ref } \tau \implies \Gamma; \Omega; \Delta; \Gamma \vdash !e \preceq !e : \tau$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Omega. \rho \in \mathcal{D}[\![\Delta]\!] \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\![\Gamma]\!]_\rho \wedge (W, \gamma_\Omega) \in \mathcal{G}[\![\Omega]\!] \wedge (W, \gamma_\Omega) \in \mathcal{G}[\![\Omega]\!].$$

then

$$(W, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(!e^+))), \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(!e^+)))) \in \mathcal{E}[\![\tau]\!]_\rho$$

We can push the compiler and substitutions through the dereference to refine this to:

$$(W, !\gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e^+))), !\gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e^+)))) \in \mathcal{E}[\![\tau]\!]_\rho$$

We appeal to Lemma B.1.11 and our hypothesis to reduce our obligation to:

$$(W', !v_1, !v_2) \in \mathcal{E}[\![\tau]\!]_\rho$$

Where $W \sqsubseteq W'$ and $(W', v_1, v_2) \in \mathcal{V}[\![\text{ref } \tau]\!]_\rho$. From the definition of $\mathcal{V}[\![\text{ref } \tau]\!]_\rho$, we know that v_i are locations that point to values in $\mathcal{V}[\![\tau]\!]_\rho$, and thus the result follows from Lemma B.1.11 and Lemma B.1.1. \square

Lemma B.1.29 (Compat $:=$).

$$\begin{aligned} \Gamma; \Omega_1; \Delta; \Gamma \vdash e_1 \preceq e_1 : \text{ref } \tau \wedge \Gamma; \Omega_2; \Delta; \Gamma \vdash e_2 \preceq e_2 : \tau \\ \implies \Gamma; \Omega_1 \uplus \Omega_2; \Delta; \Gamma \vdash e_1 := e_2 \preceq e_1 := e_2 : \text{unit} \end{aligned}$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Omega. \rho \in \mathcal{D}[\![\Delta]\!] \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\![\Gamma]\!]_\rho \wedge (W, \gamma_\Omega) \in \mathcal{G}[\![\Omega]\!] \wedge (W, \gamma_\Omega) \in \mathcal{G}[\![\Omega_1 \uplus \Omega_2]\!].$$

then

$$(W, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e_1 := e_2^+))), \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e_1 := e_2^+)))) \in \mathcal{E}[\![\text{unit}]\!]_\rho$$

We can push the compiler and substitutions through the assignment to refine that to:

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1^+)))) := \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_2^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+))) \in \mathcal{E}[\![\text{unit}]\!]_{\rho}$$

In order to proceed, first notice that, by Lemma B.1.2, $\gamma_{\Omega} = \gamma_1 \uplus \gamma_2$ where

$$(W, \gamma_1) \in \mathcal{G}[\![\Omega_1]\!], \text{ and } (W, \gamma_2) \in \mathcal{G}[\![\Omega_2]\!].$$

and, for any $i \in \{1, 2\}$

$$\gamma_{\Omega}^i(e_1^+) = \gamma_1^i(e_1^+) \text{ and } \gamma_{\Omega}^i(e_2^+) = \gamma_2^i(e_2^+)$$

This means we can rewrite our obligation to:

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_1^1(e_1^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_1^2(e_1^+)))) := \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_2^1(e_2^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_2^2(e_2^+))) \in \mathcal{E}[\![\text{unit}]\!]_{\rho}$$

We now appeal to Lemma B.1.11, using the first hypothesis to reduce our obligation to:

$$(W_1, e_1^* := \gamma_{\Gamma}^1(\gamma_2^1(e_2^+)), e_1^\dagger := \gamma_{\Gamma}^2(\gamma_2^2(e_2^+))) \in \mathcal{E}[\![\text{unit}]\!]_{\rho}$$

Where $W \sqsubseteq W_1$ and $(W_1, e_1^*, e_1^\dagger) \in \mathcal{V}[\![\text{ref } \tau]\!]_{\rho}$. We then appeal again to Lemma B.1.11, resulting in the following obligation:

$$(W_2, e_1^* := e_2^*, e_1^\dagger := e_2^\dagger) \in \mathcal{E}[\![\text{unit}]\!]_{\rho}$$

Where $W_1 \sqsubseteq W_2$ and $(W_2, e_2^*, e_2^\dagger) \in \mathcal{V}[\![\tau]\!]_{\rho}$. Since our value relations are closed under extension (Lemma B.1.3), we can reduce again using Lemma B.1.11 to step to a future world with unit values, which by Lemma B.1.1 are in $\mathcal{E}[\![\text{unit}]\!]_{\rho}$, thus completing the proof. \square

Lemma B.1.30 (Compat $\langle e \rangle_{\tau}$).

$$\Delta; \Gamma; \Omega \vdash e \preceq e : \tau \wedge \tau \sim \tau \implies \Gamma; \Omega; \Delta; \Gamma \vdash \langle e \rangle_{\tau} \preceq \langle e \rangle_{\tau} : \tau \wedge _ : \tau \sim \tau$$

Proof. We must show that given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \cdot \rho \in \mathcal{D}[\![\Delta]\!] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\![\Omega]\!] \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\![\Omega]\!].$$

then

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\langle e \rangle_{\tau}^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\langle e \rangle_{\tau}^+))) \in \mathcal{E}[\![\tau]\!]_{\rho}$$

We can push the compiler and substitutions through to refine that to:

$$(W, C_{\tau \mapsto \tau}(\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\langle e \rangle_{\tau}^+))), C_{\tau \mapsto \tau}(\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\langle e \rangle_{\tau}^+))))) \in \mathcal{E}[\![\tau]\!]_{\rho}$$

Now, by instantiating our induction hypothesis with $W, \gamma_{\Gamma}, \gamma_{\Omega}, \rho$, we find that:

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))) \in \mathcal{E}[\tau].$$

Therefore, by Theorem B.1.13, we have

$$(W, C_{\tau \mapsto \tau}(\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+)))), C_{\tau \mapsto \tau}(\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+))))) \in \mathcal{E}[\tau].$$

Finally, by Lemma B.1.8, we have

$$(W, C_{\tau \mapsto \tau}(\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+)))), C_{\tau \mapsto \tau}(\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+))))) \in \mathcal{E}[\tau]_{\rho}$$

as was to be proven. \square

Lemma B.1.31 (Compat **unit**).

$$\Delta; \Gamma; \Omega \vdash () \preceq () : \text{unit}$$

Proof. Expanding this definition, given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \cdot \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega].$$

we must show

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1((0)^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2((0)^+))) \in \mathcal{E}[\text{unit}].$$

$(0)^+ = 0$ is a closed term, so the closings have no effect. Ergo,

$$\gamma_{\Gamma}^1(\gamma_{\Omega}^1((0)^+)) = \gamma_{\Gamma}^2(\gamma_{\Omega}^2((0)^+)) = 0$$

One can easily see $(W, 0, 0) \in \mathcal{V}[\text{unit}]$, which suffices to show $(W, 0, 0) \in \mathcal{E}[\text{unit}]$. by Lemma B.1.1. This suffices to finish the proof. \square

Lemma B.1.32 (Compat **true**).

$$\Delta; \Gamma; \Omega \vdash \text{true} \preceq \text{true} : \text{bool}$$

Proof. Expanding this definition, given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \cdot \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega].$$

we must show

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\text{true}^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\text{true}^+))) \in \mathcal{E}[\text{bool}].$$

$\text{true}^+ = 0$ is a closed term, so the closings have no effect. Ergo,

$$\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\text{true}^+)) = \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\text{true}^+)) = 0$$

One can easily see $(W, 0, 0) \in \mathcal{V}[\![\text{bool}]\!]$, which suffices to show $(W, 0, 0) \in \mathcal{E}[\![\text{bool}]\!]$. by Lemma B.1.1. This suffices to finish the proof. \square

Lemma B.1.33 (Compat **false**).

$$\Delta; \Gamma; \Gamma; \Omega \vdash \text{false} \preceq \text{false} : \text{bool}$$

Proof. This is very similar to the proof for true, except $\text{false}^+ = 1$, and since $1 \neq 0$, $(W, 1, 1) \in \mathcal{V}[\![\text{bool}]\!]$. by the second clause. \square

Lemma B.1.34 (Compat **int**).

$$\Delta; \Gamma; \Gamma; \Omega \vdash n \preceq n : \text{int}$$

Proof. Expanding this definition, given

$$\forall W. \forall \rho \gamma_{\Gamma} \gamma_{\Gamma} \gamma_{\Omega}. \rho \in \mathcal{D}[\![\Delta]\!] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]_{\rho} \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]. \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\![\Omega]\!].$$

we must show

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(n^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(n^+)))) \in \mathcal{E}[\![\text{int}]\!].$$

$n^+ = n$ is a closed term, so the closings have no effect. Ergo,

$$\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(n^+))) = \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(n^+))) = n$$

Since $n \in \mathbb{Z}$, one can easily see $(W, n, n) \in \mathcal{V}[\![\text{int}]\!]$, which suffices to show $(W, n, n) \in \mathcal{E}[\![\text{int}]\!]$. by Lemma B.1.1. This suffices to finish the proof. \square

Lemma B.1.35 (Compat **a**).

$$a : \tau \in \Omega \implies \Delta; \Gamma; \Gamma; \Omega \vdash a \preceq a : \tau$$

Proof. Expanding the conclusion, given

$$\forall W. \forall \rho \gamma_{\Gamma} \gamma_{\Gamma} \gamma_{\Omega}. \rho \in \mathcal{D}[\![\Delta]\!] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]_{\rho} \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]. \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\![\Omega]\!].$$

we must show

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(a^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(a^+)))) \in \mathcal{E}[\![\tau]\!].$$

We can push the compiler and substitutions through this expression to refine this to:

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(a))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(a)))) \in \mathcal{E}[\![\tau]\!].$$

Since $(W, \gamma_{\Omega}) \in \mathcal{G}[\Omega]..$, there must exist $(\ell_1, \ell_2) \in W.\Theta$ and values v_1, v_2 such that

$$\gamma_{\Omega}^1(a) = \text{guard}(v_1, \ell_1) = \lambda_{\cdot}.\text{if } !\ell_1 \{ \text{fail CONV} \} \{\ell_1 := \text{USED}; v_1\}$$

and

$$\gamma_{\Omega}^2(a) = \text{guard}(v_2, \ell_2) = \lambda_{\cdot}.\text{if } !\ell_2 \{ \text{fail CONV} \} \{\ell_2 := \text{USED}; v_2\}$$

and $(W, v_1, v_2) \in \mathcal{V}[\tau]..$

Ergo, we must show

$$(W, \lambda_{\cdot}.\text{if } !\ell_1 \{ \text{fail CONV} \} \{\ell_1 := \text{USED}; v_1\} (), \lambda_{\cdot}.\text{if } !\ell_2 \{ \text{fail CONV} \} \{\ell_2 := \text{USED}; v_2\} ()) \in \mathcal{E}[\tau].$$

We appeal to Lemma B.1.11, noting that any heaps that satisfies W will map ℓ_i to either USED or UNUSED. If it is USED, then we reduce our obligation to $(W', \text{fail CONV}, \text{fail CONV}) \in \mathcal{E}[\tau]..$, which is trivially true. Otherwise, we reduce to heaps where $\ell_i := \text{USED}$ and reduce our obligation to $(W', v_1, v_2) \in \mathcal{E}[\tau]..$, but that follows from Lemma B.1.3 and Lemma B.1.1. \square

Lemma B.1.36 (Compat x).

$$x : \tau \in \Gamma \implies \Delta; \Gamma; \Omega \vdash x \preceq x : \tau$$

Proof. Expanding the conclusion, given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Omega} \cdot \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega] \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega].$$

we must show

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(x^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(x^+))) \in \mathcal{E}[\tau].$$

Notice that $x^+ = x$. Then, since $x \notin \Omega$ and $(W, \gamma_{\Omega}) \in \mathcal{G}[\Omega]..$, we have

$$\gamma_{\Omega}^1(x) = \gamma_{\Omega}^2(x) = x$$

Then, since $x : \tau \in \Gamma$ and $(W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]..$, there must exist v_1, v_2 such that

$$\gamma_{\Gamma}(x) = (v_1, v_2)$$

and $(W, v_1, v_2) \in \mathcal{V}[\tau]..$ Ergo,

$$\gamma_{\Gamma}^1(x) = v_1 \wedge \gamma_{\Gamma}^2(x) = v_2$$

Since $(W, v_1, v_2) \in \mathcal{V}[\tau]_.$, this suffices to show that

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(x))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(x)))) \in \mathcal{V}[\tau].$$

By Lemma B.1.1, $\mathcal{V}[\tau]_.\subseteq \mathcal{E}[\tau]_.$, so this suffices to finish the proof. \square

Lemma B.1.37 (Compat \multimap).

$$\Delta; \Gamma; \Omega, a : \tau_1 \vdash e \preceq e : \tau_2 \implies \Delta; \Gamma; \Omega \vdash \lambda a : \tau_1. e \preceq \lambda a : \tau_1. e : \tau_1 \multimap \tau_2$$

Proof. Expanding the conclusion, given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Omega}. \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega] \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega].$$

we must show

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\lambda a : \tau_1. e^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\lambda a : \tau_1. e^+))) \in \mathcal{E}[\tau_1 \multimap \tau_2].$$

We can push the compiler and the substitutions to refine that to:

$$(W, \lambda a. \gamma_{\Gamma}^1(\gamma_{\Omega}^1(e^+))), \lambda a. \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+))) \in \mathcal{E}[\tau_1 \multimap \tau_2].$$

Since these are clearly values, by Lemma B.1.1, it suffices to show

$$(W, \lambda a. \gamma_{\Gamma}^1(\gamma_{\Omega}^1(e^+))), \lambda a. \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+))) \in \mathcal{V}[\tau_1 \multimap \tau_2]_{\rho}$$

Expanding the value relation definition, given

$$\forall v_1 v_2. W' . W \sqsubseteq W' \wedge (W', v_1, v_2) \in \mathcal{V}[\tau_1].$$

we must show

$$\begin{aligned} & ((W'.k, W'.\Psi, W'.\Theta \uplus (\ell_1, \ell_2) \mapsto \text{UNUSED}), \\ & [a \mapsto \text{guard}(v_1, \ell_1)]\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+))), \\ & [a \mapsto \text{guard}(v_2, \ell_2)]\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))) \in \mathcal{E}[\tau_2]. \end{aligned}$$

Notice that $W'' = (W'.k, W'.\Psi, W'.\Theta \uplus (\ell_1, \ell_2) \mapsto \text{UNUSED})$ is a world extension of W' because it has the same heap typing as W' and has all the affine flags as W' plus one new affine flag which is disjoint from any affine flag in W' . Ergo, since $W \sqsubseteq W'$ and $W' \sqsubseteq W''$, we have $W \sqsubseteq W''$. Next, notice that:

$$(W'', \gamma_{\Omega}[a \mapsto (\text{guard}(v_1, \ell_1), \text{guard}(v_2, \ell_2))]) \in \mathcal{G}[\Omega, a : \tau_1].$$

because $(\ell_1, \ell_2) \in \text{dom}(W''.\Theta)$, $(W'', v_1, v_2) \in \mathcal{V}[\tau_1]$. because $(W, v_1, v_2) \in \mathcal{V}[\tau_1]$. and Lemma B.1.3, and $(W'', \gamma_{\Omega}) \in \mathcal{G}[\Omega]$. because $(W, \gamma_{\Omega}) \in \mathcal{G}[\Omega]$.

and Lemma B.1.3. Therefore, we can instantiate the first induction hypothesis with $W'', \gamma_{\Gamma}, \gamma_{\Omega}[\alpha \mapsto (\text{guard}(v_1, \ell_1), \text{guard}(v_2, \ell_2))], \rho$ to find

$$(W'', \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\alpha \mapsto (\text{guard}(v_1, \ell_1), \text{guard}(v_2, \ell_2)))^1(e^+))), \\ \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\alpha \mapsto (\text{guard}(v_1, \ell_1), \text{guard}(v_2, \ell_2)))^2(e^+))) \in \mathcal{E}[\tau_2].$$

which is equivalent to what was to be proven. \square

Lemma B.1.38 (Compat app).

$$\Delta_1; \Gamma_1; \Omega_1 \vdash e_1 \preceq e_1 : \tau_1 \multimap \tau_2 \wedge \Delta_2; \Gamma_2; \Omega_2 \vdash e_2 \preceq e_2 : \tau_1 \\ \implies \Delta_1; \Gamma_1; \Omega_1 \uplus \Omega_2 \vdash e_1 e_2 \preceq e_1 e_2 : \tau_2$$

Proof. Expanding this definition, given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega_1 \uplus \Omega_2].$$

we must show

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(e_1 e_2^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1 e_2^+))) \in \mathcal{E}[\tau_2].$$

We can push the compiler and substitutions through the application to refine this to:

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+)))) (\text{let } x = \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+))) \text{ in once}(x)), \\ \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+))) (\text{let } x = \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+))) \text{ in once}(x)) \in \mathcal{E}[\tau_2].$$

First, notice that, by Lemma B.1.2, $\gamma_{\Omega} = \gamma_1 \uplus \gamma_2$ where

$$(W, \gamma_1) \in \mathcal{G}[\Omega_1]. \text{ and } (W, \gamma_2) \in \mathcal{G}[\Omega_2].$$

and, for any $i \in \{1, 2\}$

$$\gamma_{\Omega}^i(e_1^+) = \gamma_1^i(e_1^+) \text{ and } \gamma_{\Omega}^i(e_2^+) = \gamma_2^i(e_2^+)$$

We proceed by appealing to Lemma B.1.11, using our first hypothesis to reduce our obligation to:

$$(W_1, e_1^* (\text{let } x = \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+))) \text{ in once}(x)), \\ e_1^\dagger (\text{let } x = \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+))) \text{ in once}(x))) \in \mathcal{E}[\tau_2].$$

For some W_1 where $W \sqsubseteq W_1$ and $(W_1, e_1^*, e_1^\dagger) \in \mathcal{V}[\tau_1 \multimap \tau_2]$. And then again, using our second hypotheses to reduce our obligation to:

$$(W_2, e_1^* \text{ once}(e_2^*), e_1^\dagger \text{ once}(e_2^\dagger)) \in \mathcal{E}[\tau_2].$$

For some W_2 where $W_1 \sqsubseteq W_2$ and $(W_2, e_2^*, e_2^\dagger) \in \mathcal{V}[\tau_1]$.

Then, instantiate $(W_1, e_1^*, e_1^\dagger) \in \mathcal{V}[\tau_1 \multimap \tau_2]$. with $e_2^*, e_2^\dagger, \triangleright W_2$. Because $W_1 \sqsubseteq W_2$ and $W_2 \sqsubset \triangleright W_2$, it follows that $W_1 \sqsubset \triangleright W_2$. Moreover, $(\triangleright W_2, e_2^*, e_2^\dagger) \in \mathcal{V}[\tau_1]$. (because $(W_2, e_2^*, e_2^\dagger) \in \mathcal{V}[\tau_1]$. and $W_2 \sqsubseteq \triangleright W_2$, so we can apply Lemma B.1.3). Ergo, there exist e_b^*, e_b^\dagger such that

$$e_1^* = \lambda a. e_b^*$$

and

$$e_1^\dagger = \lambda a. e_b^\dagger$$

and, for any $(\ell_1, \ell_2) \notin \text{dom}(\triangleright W_2. \Psi) \cup \text{dom}(\triangleright W_2. \Theta)$,

$$\begin{aligned} & ((\triangleright W_2. k, \triangleright W_2. \Psi, \triangleright W_2. \Theta \uplus (\ell_1, \ell_2) \mapsto \text{UNUSED}), \\ & [a \mapsto \text{guard}(e_2^*, \ell_2)] e_b^*, [a \mapsto \text{guard}(e_2^\dagger, \ell_1)] e_b^\dagger) \in \mathcal{E}[\tau_2]. \end{aligned}$$

In particular, we choose ℓ_1, ℓ_2 to match those that arise due to reducing the once()s, and thus complete the proof by appeal to Lemma B.1.11. \square

Lemma B.1.39 (Compat !).

$$\Delta; \Gamma; \Gamma; \cdot \vdash v \preceq v : \tau \implies \Delta; \Gamma; \Gamma; \cdot \vdash !v \preceq !v : !\tau$$

Proof. Expanding this definition, given

$$\forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Omega \cdot \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \gamma_\Omega) \in \mathcal{G}[\Omega] \wedge (W, \gamma_\Omega) \in \mathcal{G}[\Omega].$$

we must show

$$(W, \gamma_\Gamma^1(\gamma_\Omega^1(!v^+))), \gamma_\Gamma^2(\gamma_\Omega^2(!v^+))) \in \mathcal{E}[!\tau].$$

Note that $!v^+ = v^+$. Thus, the result follows immediately from our hypothesis. \square

Lemma B.1.40 (Compat let!).

$$\begin{aligned} & \Delta; \Gamma; \Gamma; \Omega_1 \vdash e_1 \preceq e_1 : !\tau \wedge \Delta; \Gamma; \Gamma; x : \tau; \Omega_2 \vdash e_2 \preceq e_2 : \tau' \\ & \implies \Delta; \Gamma; \Gamma; \Omega_1 \uplus \Omega_2 \vdash \text{let } !x = e_1 \text{ in } e_2 \preceq \text{let } !x = e_1 \text{ in } e_2 : \tau' \end{aligned}$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Omega \cdot \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \gamma_\Omega) \in \mathcal{G}[\Omega] \wedge (W, \gamma_\Omega) \in \mathcal{G}[\Omega_1 \uplus \Omega_2].$$

we must show

$$\begin{aligned} & (W, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(\text{let } !x = e_1 \text{ in } e_2^+)))), \\ & \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(\text{let } !x = e_1 \text{ in } e_2^+)))) \in \mathcal{E}[\tau']. \end{aligned}$$

We can push the compiler and substitutions through the **let** expression and refine this to:

$$(W, \text{let } x = \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+))) \text{ in } \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_2^+))), \\ \text{let } x = \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1^+))) \text{ in } \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+))) \in \mathcal{E}[\![\tau']\!].$$

Then, by Lemma B.1.2, we find that $\gamma_{\Omega} = \gamma_1 \uplus \gamma_2$ where

$$(W, \gamma_1) \in \mathcal{G}[\![\Omega_1]\!]. \text{ and } (W, \gamma_2) \in \mathcal{G}[\![\Omega_2]\!].$$

and, for any $i \in \{1, 2\}$

$$\gamma_{\Omega}^i(e_1^+) = \gamma_1^i(e_1^+) \text{ and } \gamma_{\Omega}^i(e_2) = \gamma_2^i(e_2^+)$$

Thus, we must show

$$(W, \text{let } x = \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_1^1(e_1^+))) \text{ in } \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_2^1(e_2^+))), \\ \text{let } x = \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_1^2(e_1^+))) \text{ in } \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_2^2(e_2^+))) \in \mathcal{E}[\![\tau']\!].$$

We appeal to Lemma B.1.11, using the first hypothesis to reduce this to:

$$(W_1, \text{let } x = e_1^* \text{ in } \gamma_{\Gamma}^1(\gamma_2^1(e_2^+))), \\ \text{let } x = e_1^\dagger \text{ in } \gamma_{\Gamma}^2(\gamma_2^2(e_2^+)) \in \mathcal{E}[\![\tau']\!].$$

For some W_1 where $W \sqsubseteq W_1$, $H_1^*, H_2^* : W_1$, and $(W_1, e_1^*, e_1^\dagger) \in \mathcal{V}[\![!\tau]\!]$. By expanding the value relation definition, we find $(W_1, e_1^*, e_1^\dagger) \in \mathcal{V}[\![\tau]\!]$. That means that we can appeal to Lemma B.1.11 again, reducing our problem to exactly what is provided by instantiating our second hypothesis with substitutions extended with $x \mapsto (e_1^*, e_1^\dagger)$. \square

Lemma B.1.41 (Compat &).

$$\Delta; \Gamma; \Omega \vdash e_1 \preceq e_1 : \tau_1 \wedge \Delta; \Gamma; \Omega \vdash e_2 \preceq e_2 : \tau_2 \\ \implies \Delta; \Gamma; \Omega \vdash \langle e_1, e_2 \rangle \preceq \langle e_1, e_2 \rangle : \tau_1 \& \tau_2$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Omega} \cdot \rho \in \mathcal{D}[\![\Delta]\!] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\![\Omega]\!] \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\![\Omega]\!].$$

we must show

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\langle e_1, e_2 \rangle^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\langle e_1, e_2 \rangle^+))) \in \mathcal{E}[\![\tau_1 \& \tau_2]\!].$$

We can push the compiler and substitutions through the product expression and refine this to:

$$(W, (\lambda_{-}. \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+)))), \lambda_{-}. \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_2^+)))), \\ (\lambda_{-}. \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1^+)))), \lambda_{-}. \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+)))) \in \mathcal{E}[\![\tau_1 \& \tau_2]\!].$$

Clearly, this is a target value. Thus, it suffices to show

$$(W, (\lambda_{-} \cdot \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+))), \lambda_{-} \cdot \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_2^+)))), \\ (\lambda_{-} \cdot \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1^+))), \lambda_{-} \cdot \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+))))) \in \mathcal{V}[\tau_1 \& \tau_2].$$

First, we can instantiate the first induction hypothesis with $W, \gamma_{\Gamma}, \gamma_{\Omega}, \rho$ to show that

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1^+)))) \in \mathcal{V}[\tau_1].$$

and we can instantiate the second induction hypothesis with $W, \gamma_{\Gamma}, \gamma_{\Omega}, \rho$ to show that

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_2^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+)))) \in \mathcal{V}[\tau_2].$$

This suffices to show that the pairs of lambdas are in the value relation at $\tau_1 \& \tau_2$, as was to be proven. \square

Lemma B.1.42 (Compat .1).

$$\Delta; \Gamma; \Omega \vdash e \preceq e : \tau_1 \& \tau_2 \implies \Delta; \Gamma; \Omega \vdash e.1 \preceq e.1 : \tau_1$$

Proof. Expanding this definition, given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \cdot \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega].$$

we must show

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e.1^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e.1^+)))) \in \mathcal{E}[\tau_1].$$

We can push the compiler and substitutions through the projection to refine this to:

$$(W, (\text{fst } \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+))))(), (\text{fst } \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+))))()) \in \mathcal{E}[\tau_1].$$

We can appeal to Lemma B.1.11 to reduce this to:

$$(W, (\text{fst } e_1^*)(), (\text{fst } e_1^\dagger)) \in \mathcal{E}[\tau_1].$$

For some world W_1 where $W \sqsubseteq W_1$ and $(W_1, e^*, e^\dagger) \in \mathcal{V}[\tau_1 \& \tau_2]$.

Ergo, there exists some $e_1^*, e_1^\dagger, e_2^*, e_2^\dagger$ such that

$$e^* = (\lambda_{-} \cdot e_1^*, \lambda_{-} \cdot e_2^*) \text{ and } e^\dagger = (\lambda_{-} \cdot e_1^\dagger, \lambda_{-} \cdot e_2^\dagger)$$

and

$$(W_1, e_1^*, e_1^\dagger) \in \mathcal{E}[\tau_1]. \text{ and } (W_1, e_2^*, e_2^\dagger) \in \mathcal{E}[\tau_2].$$

We can thus complete the proof by appeal to Lemma B.1.11, since the terms take two steps to the former, which suffice to complete the proof. \square

Lemma B.1.43 (Compat .2).

$$\Delta; \Gamma; \Omega \vdash e \preceq e : \tau_1 \& \tau_2 \implies \Delta; \Gamma; \Omega \vdash e.2 \preceq e.2 : \tau_2$$

Proof. This proof is essentially identical to that of .1. \square

Lemma B.1.44 (Compat \otimes).

$$\begin{aligned} \Delta; \Gamma; \Omega_1 \vdash e_1 \preceq e_1 : \tau_1 \wedge \Delta; \Gamma; \Omega_2 \vdash e_2 \preceq e_2 : \tau_2 \\ \implies \Delta; \Gamma; \Omega_1 \uplus \Omega_2 \vdash (e_1, e_2) \preceq (e_1, e_2) : \tau_1 \otimes \tau_2 \end{aligned}$$

Proof. Expanding the conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega}. \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega] \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega_1 \uplus \Omega_2].$$

we must show

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1((e_1, e_2)^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2((e_1, e_2)^+)) \in \mathcal{E}[\tau_1 \otimes \tau_2].$$

We can push the compiler and substitutions through the product expression and refine this to:

$$(W, \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_{\Omega}^1(e_1^+))), \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_{\Omega}^1(e_2^+))), \\ (\gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_{\Omega}^2(e_1^+))), \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_{\Omega}^2(e_2^+)))) \in \mathcal{E}[\tau_1 \otimes \tau_2].$$

Then, by Lemma B.1.2, we find that $\gamma_{\Omega} = \gamma_1 \uplus \gamma_2$ where

$$(W, \gamma_1) \in \mathcal{G}[\Omega_1] \text{ and } (W, \gamma_2) \in \mathcal{G}[\Omega_2].$$

and, for any $i \in \{1, 2\}$

$$\gamma_{\Omega}^i(e_1^+) = \gamma_1^i(e_1^+) \text{ and } \gamma_{\Omega}^i(e_2^+) = \gamma_2^i(e_2^+)$$

Thus, we must show

$$(W, (\gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_1^1(e_1^+))), \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_2^+)))), \\ (\gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_1^2(e_1^+))), \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_2^2(e_2^+)))) \in \mathcal{E}[\tau_1 \otimes \tau_2].$$

We appeal to Lemma B.1.11 twice, using both of our hypotheses, to reduce out obligation to:

$$(W_2, (e_1^*, e_2^*), \\ (e_1^{\dagger}, e_2^{\dagger})) \in \mathcal{E}[\tau_1 \otimes \tau_2].$$

Where for some W_1 where $W \sqsubseteq W_1$, $(W_1, e_1^*, e_1^\dagger) \in \mathcal{V}[\tau_1]$, and for some W_2 where $W_1 \sqsubseteq W_2$, $(W_2, e_2^*, e_2^\dagger) \in \mathcal{V}[\tau_2]$. By Lemma B.1.3, we have $(W_2, e_1^*, e_1^\dagger) \in \mathcal{V}[\tau_1]$. and thus $(W_2, (e_1^*, e_2^*), (e_1^\dagger, e_2^\dagger)) \in \mathcal{V}[\tau_1 \otimes \tau_2]$, which suffices to finish the proof. \square

Lemma B.1.45 (Compat let).

$$\begin{aligned} & \Delta; \Gamma; \Omega_1 \vdash e_1 \preceq e_1 : \tau_1 \otimes \tau_2 \wedge \Delta; \Gamma; \Omega_2, a : \tau_1, a' : \tau_2 \vdash e_2 \preceq e_2 : \tau \\ \implies & \Delta; \Gamma; \Omega_1 \uplus \Omega_2 \vdash \text{let } (a, a') = e_1 \text{ in } e_2 \preceq \text{let } (a, a') = e_1 \text{ in } e_2 : \tau \end{aligned}$$

Proof. Expanding the conclusion, we must show that given

$$\begin{aligned} & \forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \\ & \wedge (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]. \wedge (W, \gamma_\Omega) \in \mathcal{G}[\Omega_1 \uplus \Omega_2]. \end{aligned}$$

we must show

$$(W, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(\text{let } (a, a') = e_1 \text{ in } e_2^+))), \\ \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(\text{let } (a, a') = e_1 \text{ in } e_2^+)))) \in \mathcal{E}[\tau].$$

We can push the compiler and substitutions through the **let** expression and refine this to:

$$\begin{aligned} & (W, \text{let } x_{\text{fresh}} = \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e_1^+))) \text{ in let } x'_{\text{fresh}} = \text{fst } x_{\text{fresh}} \text{ in let } x''_{\text{fresh}} = \text{snd } x_{\text{fresh}} \text{ in} \\ & \quad \text{let } a = \text{once}(x'_{\text{fresh}}) \text{ in let } a' = \text{once}(x''_{\text{fresh}}) \text{ in } \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e_2^+))), \\ & \text{let } x_{\text{fresh}} = \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e_1^+))) \text{ in let } x'_{\text{fresh}} = \text{fst } x_{\text{fresh}} \text{ in let } x''_{\text{fresh}} = \text{snd } x_{\text{fresh}} \text{ in} \\ & \quad \text{let } a = \text{once}(x'_{\text{fresh}}) \text{ in let } a' = \text{once}(x''_{\text{fresh}}) \text{ in } \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e_2^+))), \in \mathcal{E}[\tau]. \end{aligned}$$

Then, by Lemma B.1.2, we find that $\gamma_\Omega = \gamma_1 \uplus \gamma_2$ where

$$(W, \gamma_1) \in \mathcal{G}[\Omega_1]. \text{ and } (W, \gamma_2) \in \mathcal{G}[\Omega_2].$$

and, for any $i \in \{1, 2\}$

$$\gamma_\Omega^i(e_1^+) = \gamma_1^i(e_1^+) \text{ and } \gamma_\Omega^i(e_2^+) = \gamma_2^i(e_2^+)$$

We appeal to Lemma B.1.11 using the first hypothesis, with the refined substitutions, to reduce our obligation to:

$$\begin{aligned} & (W_1, \text{let } x_{\text{fresh}} = e_1^* \text{ in let } x'_{\text{fresh}} = \text{fst } x_{\text{fresh}} \text{ in let } x''_{\text{fresh}} = \text{snd } x_{\text{fresh}} \text{ in} \\ & \quad \text{let } a = \text{once}(x'_{\text{fresh}}) \text{ in let } a' = \text{once}(x''_{\text{fresh}}) \text{ in } \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_2^1(e_2^+))), \\ & \quad \text{let } x_{\text{fresh}} = e_1^\dagger \text{ in let } x'_{\text{fresh}} = \text{fst } x_{\text{fresh}} \text{ in let } x''_{\text{fresh}} = \text{snd } x_{\text{fresh}} \text{ in} \\ & \quad \text{let } a = \text{once}(x'_{\text{fresh}}) \text{ in let } a' = \text{once}(x''_{\text{fresh}}) \text{ in } \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_2^2(e_2^+))), \in \mathcal{E}[\tau]. \end{aligned}$$

Where for some W_1 where $W \sqsubseteq W_1$ and $(W_1, e_1^*, e_1^\dagger) \in \mathcal{V}[\tau_1 \otimes \tau_2]$.

By expanding the value relation, we find that $e_1^* = (v_1^*, v_2^*)$ and $e_1^\dagger = (v_1^\dagger, v_2^\dagger)$ where $(W_1, v_1^*, v_1^\dagger) \in \mathcal{V}[\tau_1]$. and $(W_1, v_2^*, v_2^\dagger) \in \mathcal{V}[\tau_2]$.

Thus, we can appeal to Lemma B.1.11 again to reduce our obligation to:

$$(W_2, [e_1^* \mapsto a, e_2^* \mapsto a']\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_2^1(e_2^+))), [e_1^\dagger \mapsto a, e_2^\dagger \mapsto a']\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_2^2(e_2^+)))) \in \mathcal{E}[\tau].$$

Where W_2 only differs from W_1 by reduced step index. Now, note that we can combine our substitutions and instantiate our second hypothesis to complete the proof. \square

Lemma B.1.46 (Compat $(\langle e \rangle_{\tau})$).

$$\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau \wedge \tau \sim \tau \implies \Delta; \Gamma; \Omega \vdash (\langle e \rangle_{\tau}) \preceq (\langle e \rangle_{\tau}) : \tau$$

Proof. Expanding this conclusion, we must show that given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Omega} \cdot \rho \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega] \wedge (W, \gamma_{\Omega}) \in \mathcal{G}[\Omega].$$

then

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\langle e \rangle_{\tau}^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\langle e \rangle_{\tau}^+)))) \in \mathcal{E}[\tau].$$

We can push the compiler and substitutions through the pair to refine that to:

$$(W, C_{\tau \mapsto \tau}(\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+)))), C_{\tau \mapsto \tau}(\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+))))) \in \mathcal{E}[\tau].$$

Now, by instantiating our induction hypothesis with $W, \gamma_{\Gamma}, \gamma_{\Omega}, \gamma_{\Omega}, \rho$, we find that:

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))) \in \mathcal{E}[\tau]_{\rho}$$

By Lemma B.1.8, it follows that:

$$(W, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))) \in \mathcal{E}[\tau].$$

Therefore, by Theorem B.1.13, we have

$$(W, C_{\tau \mapsto \tau}(\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+)))), C_{\tau \mapsto \tau}(\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+))))) \in \mathcal{E}[\tau].$$

as was to be proven. \square

B.2 STATIC LOGICAL RELATION

As before, we first present supporting lemmas, and then the compatibility lemmas. Note that we omit many of the **Miniml** compatibility lemmas because the differences between the proofs from the **Miniml** compatibility

lemmas from the last case study and the corresponding compatibility lemmas in this case study are relatively straightforward; we do a selection of representative cases to demonstrate that this is the case.

Lemma B.2.1 (Expression Relation Contains Value Relation).

$$\mathcal{V}[\tau]_\rho \subseteq \mathcal{E}[\tau]_\rho$$

Proof. All terms in the value relation are irreducible, and thus are trivially in the expression relation. \square

Lemma B.2.2 (Values With No Flags Are In Expression Relation).

For all $\tau, \rho, W, \Phi_1, v_1, \Phi_2, v_2$, if $(W, (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau]_\rho$, then $(W, (\Phi_1, v_1), (\Phi_2, v_2)) \in \mathcal{E}[\tau]_\rho$.

Proof. Expanding the definition of the expression relation, given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \Phi_1 \wedge \Phi_{r2} \# \Phi_2 \wedge \Phi_{r1} \uplus \Phi_1, \Phi_{r2} \uplus \Phi_2 : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, v_1 \rangle \xrightarrow{-j} \langle \Phi'_1, H'_1, e'_1 \rangle \rightarrow \end{aligned}$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2^*, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, v_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2^* \rangle \rightarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2^*)) \in \mathcal{V}[\tau]_\rho \end{aligned}$$

Since v_1, v_2 are in the value relation, they are target values, so the configurations

$$\langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, v_1 \rangle$$

and

$$\langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, v_2 \rangle$$

are irreducible. Thus, Φ'_1 is simply equal to the set of static flags in the initial configuration, so $\Phi'_1 = \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1$. Then, we can take $\Phi_{f1} = \emptyset$, $\Phi_{g1} = \Phi_1$, $\Phi_{f2} = \emptyset$, $\Phi_{g2} = \Phi_2$, $v_2^* = v_2$, $H'_2 = H_2$, and $W' = W$.

Since $\Phi_{r1}, \Phi_{r2} : W$ by assumption, we have $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W$. Everything else in the expression relation is trivial by assumption, so this suffices to finish the proof. \square

Lemma B.2.3 (Expressions With No Flags Are In Expression Relation). For all $\tau, \rho, W, \Phi_1, e_1, \Phi_2, e_2$, if $(W, (\emptyset, e_1), (\emptyset, e_2)) \in \mathcal{E}[\tau]_\rho$, then $(W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\tau]_\rho$.

Proof. Expanding the definition of the expression relation, given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \Phi_1 \wedge \Phi_{r2} \# \Phi_2 \wedge \Phi_{r1} \uplus \Phi_1, \Phi_{r2} \uplus \Phi_2 : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_1, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau]_\rho \end{aligned}$$

Now, by expanding the expression relation in the assumption, we have that, if

$$\begin{aligned} & \forall \Phi_{r1}^*, \Phi_{r2}^*, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1}^* \# \emptyset \wedge \Phi_{r2}^* \# \emptyset \wedge \Phi_{r1}^* \uplus \emptyset, \Phi_{r2}^* \uplus \emptyset : W \wedge \\ & \langle \Phi_{r1}^* \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_1, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

then either e'_1 is fail CONV or there exist $\Phi_{f1}^*, \Phi_{g1}^*, \Phi_{f2}^*, \Phi_{g2}^*, v_2, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2}^* \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2}^* \uplus \text{flags}(W', 2) \uplus \Phi_{f2}^* \uplus \Phi_{g2}^*, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1}^* \uplus \text{flags}(W', 1) \uplus \Phi_{f1}^* \uplus \Phi_{g1}^* \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}^*, \Phi_{r2}^*} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}^*, e'_1), (\Phi_{f2}^*, v_2)) \in \mathcal{V}[\tau]_\rho \end{aligned}$$

Then, we can instantiate this fact with $\Phi_{r1}^* = \Phi_{r1} \uplus \Phi_1, \Phi_{r2}^* = \Phi_{r2} \uplus \Phi_2$. We then find that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \Phi_2 \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \Phi_2 \uplus \text{flags}(W', 2) \uplus \Phi_{f2}^* \uplus \Phi_{g2}^*, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \Phi_1 \uplus \text{flags}(W', 1) \uplus \Phi_{f1}^* \uplus \Phi_{g1}^* \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}^*, \Phi_{r2}^*} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}^*, e'_1), (\Phi_{f2}^*, v_2)) \in \mathcal{V}[\tau]_\rho \end{aligned}$$

Then, we can take $\Phi_{f1} = \Phi_{f1}^*, \Phi_{g1} = \Phi_{g1}^* \uplus \Phi_1, \Phi_{f2} = \Phi_{f2}^*, \Phi_{g2} = \Phi_{g2}^* \uplus \Phi_2$. Then, everything in the expression relation we have to prove trivially follows from the above, so the proof is finished. \square

Lemma B.2.4 (**Aff** Values Compile to Target Values).

Proof. By induction over the syntax: $()$ compiles to $()$, $\lambda a_e : \tau.e$ compiles to a target function, $\langle e, e' \rangle$ compiles to a pair of target functions, $!v$ compiles

to \mathbf{v}^+ (which is a target value by the induction hypothesis), and $(\mathbf{v}, \mathbf{v}')$ compiles to $(\mathbf{v}^+, \mathbf{v}'^+)$ (where both \mathbf{v}^+ and \mathbf{v}'^+ are target values by the induction hypothesis). \square

Lemma B.2.5 (Split Substitutions). *For any world W , flagsets Φ_1, Φ_2 , and substitution γ such that*

$$(W, \Phi_1, \Phi_2, \gamma) \in \mathcal{G}[\![\Omega_1 \uplus \Omega_2]\!]_\rho$$

there exist flagsets $\Phi_{1l}, \Phi_{1r}, \Phi_{2l}, \Phi_{2r}$ such that $\Phi_1 = \Phi_{1l} \uplus \Phi_{1r}$, $\Phi_2 = \Phi_{2l} \uplus \Phi_{2r}$, and substitutions γ_1, γ_2 such that $\gamma = \gamma_1 \uplus \gamma_2$ and

$$(W, \Phi_{1l}, \Phi_{2l}, \gamma_1) \in \mathcal{G}[\![\Omega_1]\!]_\rho$$

and

$$(W, \Phi_{1r}, \Phi_{2r}, \gamma_2) \in \mathcal{G}[\![\Omega_2]\!]_\rho$$

Moreover, for any $i, j \in \{1, 2\}$, for any $\Gamma; \Omega_j; \Delta; \Gamma \vdash e : \tau$,

$$\gamma^i(e^+) = \gamma_j^i(e^+)$$

and for any $\Gamma; \Omega_j; \Delta; \Gamma \vdash e : \tau$,

$$\gamma^i(e^+) = \gamma_j^i(e^+)$$

Proof. First, we need to show that there exist substitutions γ_1 and γ_2 . This follows from the inductive structure of $\mathcal{G}[\![\Omega]\!]_\rho$, where we can separate the parts that came from $\mathcal{G}[\![\Omega_1]\!]_\rho$ and $\mathcal{G}[\![\Omega_2]\!]_\rho$. The second follows from the fact that the statics means that the rest of the substitution must not occur in the term, and thus $\gamma^i(e^+) = \gamma_1^i(\gamma_2^i(e^+)) = \gamma_1^i(e^+)$ (for example). \square

Lemma B.2.6 (No Static Variables in **MinimL** Terms). *For any world W , flagsets Φ_1, Φ_2 , and substitution γ such that*

$$(W, \Phi_1, \Phi_2, \gamma) \in \mathcal{G}[\![\Omega]\!]_\rho$$

then there exists a substitution γ' such that

$$(W, \emptyset, \emptyset, \gamma') \in \mathcal{G}[\![\Omega_o]\!]_\rho$$

and, for all $\Gamma; \Omega; \Delta; \Gamma \vdash e : \tau$ and for all $i \in \{1, 2\}$,

$$\gamma^i(e^+) = \gamma'^i(e^+)$$

Proof. Let Ω_\bullet be the set of all static variables in Ω . Since Ω only contains dynamic or static variables, $\Omega = \Omega_o \uplus \Omega_\bullet$, so by Lemma B.2.5, there exist flagsets $f_{1l}, f_{1r}, f_{2l}, f_{2r}$ and substitutions γ_1, γ_2 such that $f_1 = f_{1l} \uplus f_{1r}$, $f_2 =$

$f_{2l} \uplus f_{2r}, \gamma = \gamma_1 \uplus \gamma_2, (W, f_{1l}, f_{2l}, \gamma_1) \in \mathcal{G}[\Omega_\circ]_\rho$ and $(W, f_{1r}, f_{2r}, \gamma_2) \in \mathcal{G}[\Omega_\bullet]_\rho$. Since Ω_\circ only contains dynamic variables, $f_{1l} = f_{2l} = \emptyset$. Thus, we can take $\gamma' = \gamma_1$.

Now, we must prove, for any $\Gamma; \Omega; \Delta; \Gamma \vdash e : \tau$ and for any $i \in \{1, 2\}$, it holds that $\gamma^i(e^+) = \gamma_1^i(e^+)$. Since $\gamma = \gamma_1 \uplus \gamma_2$, we have

$$\gamma^i(e^+) = \gamma_1^i(\gamma_2^i(e^+))$$

Notice that γ_2 only contains variables annotated with \bullet . However, e^+ contains no free variables annotated with \bullet because, if it did, then there would need to be a free static variable under a $(\cdot)_\tau$ boundary, as only static variables in **Affi** get compiled to variables annotated with \bullet in the target. However, the typing rule for $(\cdot)_\tau$ does not allow for free static variables, so this is impossible and thus e^+ contains no free variables annotated with \bullet . Ergo, closing e^+ with γ_2 has no impact, so

$$\gamma^i(e^+) = \gamma_1^i(\gamma_2^i(e^+)) = \gamma_1^i(e^+)$$

as was to be proven. \square

Lemma B.2.7 (Strengthening Logical Relation for **MiniML**). *For all $\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau$, if there exists some $(W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega]_.$, it holds that:*

$$\begin{aligned} & \forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \\ & \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \emptyset, \emptyset, \gamma_\Omega) \in \mathcal{G}[\Omega]. \wedge (W, \emptyset, \emptyset, \gamma_{\Omega_\circ}) \in \mathcal{G}[\Omega_\circ]. \\ & \implies (W, (\emptyset, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_{\Omega_\circ}^1(e^+)))), (\emptyset, \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_{\Omega_\circ}^2(e^+))))) \in \mathcal{E}[\tau]_\rho \end{aligned}$$

Proof. Since Ω only contains dynamic or static variables, $\Omega = \Omega_\circ \uplus \Omega_\bullet$, so by Lemma B.2.5, there exist flagsets $f_{1l}, f_{1r}, f_{2l}, f_{2r}$ and substitutions γ_1, γ_2 such that $f_1 = f_{1l} \uplus f_{1r}, f_2 = f_{2l} \uplus f_{2r}, \gamma = \gamma_1 \uplus \gamma_2, (W, f_{1l}, f_{2l}, \gamma_1) \in \mathcal{G}[\Omega_\circ]_\rho$ and $(W, f_{1r}, f_{2r}, \gamma_2) \in \mathcal{G}[\Omega_\bullet]_\rho$.

Now, consider the given hypothesis. Given

$$\begin{aligned} & \forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \\ & \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \emptyset, \emptyset, \gamma_\Omega) \in \mathcal{G}[\Omega]. \wedge (W, \emptyset, \emptyset, \gamma_{\Omega_\circ}) \in \mathcal{G}[\Omega_\circ]. \end{aligned}$$

we must show:

$$(W, (\emptyset, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_{\Omega_\circ}^1(e^+)))), (\emptyset, \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_{\Omega_\circ}^2(e^+))))) \in \mathcal{E}[\tau]_\rho$$

Since $(W, f_{1r}, f_{2r}, \gamma_2) \in \mathcal{G}[\Omega_\bullet]_\rho$, $(W, \emptyset, \emptyset, \gamma_{\Omega_\circ}) \in \mathcal{G}[\Omega_\circ]_.$, and $\Omega = \Omega_\bullet \uplus \Omega_\circ$, it holds that $(W, f_{1r}, f_{2r}, \gamma_2 \uplus \gamma_{\Omega_\circ}) \in \mathcal{G}[\Omega]_.$. Thus, by applying $\Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau$, we find

$$(W, (\emptyset, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_2 \uplus \gamma_{\Omega_\circ}^1(e^+)))), (\emptyset, \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_2 \uplus \gamma_{\Omega_\circ}^2(e^+))))) \in \mathcal{E}[\tau]_\rho$$

As explained in the proof for Lemma B.2.6, e^+ has no free variables annotated with \bullet , so closing e^+ over with γ_2 has no impact. Ergo,

$$(W, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+)))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+))))) \in \mathcal{E}[\tau]_{\rho}$$

which suffices to finish the proof. \square

Lemma B.2.8 (World Extension).

1. If $(W_1, (\Phi_1, v_1), (\Phi_2, v_2)) \in \mathcal{V}[\tau]_{\rho}$ and $W_1 \sqsubseteq_{\Phi_1, \Phi_2} W_2$, then $(W_2, (\Phi_1, v_1), (\Phi_2, v_2)) \in \mathcal{V}[\tau]_{\rho}$
2. If $(W_1, \Phi_1, \Phi_2, \gamma) \in \mathcal{G}[\Gamma]_{\rho}$ and $W_1 \sqsubseteq_{\Phi_1, \Phi_2} W_2$, then $(W_2, \Phi_1, \Phi_2, \gamma) \in \mathcal{G}[\Gamma]_{\rho}$

Proof. We note that world extension allows three things: the step index to decrease, the heap typing to add bindings (holding all existing bindings at same relation, module decreasing step index), and add flag references (ensuring existing flag references can go from pairs of sets of static flags to USED, but not the other way). In all cases, this is straightforward based on the definition (relying on Lemma B.1.4 in some cases). \square

Lemma B.2.9 (World Extension Transitive). *If $W_1 \sqsubseteq_{\Phi_1, \Phi_2} W_2$ and $W_2 \sqsubseteq_{\Phi'_1, \Phi'_2} W_3$ then $W_1 \sqsubseteq_{\Phi_1 \cap \Phi'_1, \Phi_2 \cap \Phi'_2} W_3$.*

Proof. This holds trivially for step indices, the heap typing, and the monotonicity of marking affine flags as USED. What remains is the side condition that the world satisfies Φ_1, Φ_2 . Since that is defined as being disjoint from the set of flags in W and W' , the set of flags that is disjoint from both W_1 and W_3 is the intersection. \square

Lemma B.2.10 (Heaps in Later World). *For any $W \in \text{World}$ and $H_1, H_2 : W$, it holds that $H_1, H_2 : \triangleright W$.*

Proof. For $H_1, H_2 : \triangleright W$, we need three things.

The first is that for any mapping $(\ell_1, \ell_2) \mapsto R$ in $\triangleright W.\Psi$, $(\triangleright \triangleright W, H_1(\ell_1), H_2(\ell_2)) \in R$. Since R is drawn from Typ , we know it is closed under world extension and thus the fact that $(\triangleright W, H_1(\ell_1), H_2(\ell_2)) \in R$ means this holds.

The other two conditions, which relate to $W.\Theta$, are unaffected by the shift of step index, and so hold trivially in $\triangleright W$. \square

Lemma B.2.11 (Heaps in Later World). *For any $W \in \text{World}$ and $H_1, H_2 : W$, it holds that $H_1, H_2 : \triangleright W$.*

Proof. Since heap typings map to relations that are by definition closed under world extension, and world extension cannot remove locations, only restrict them to future step indices, this holds by definition. \square

Lemma B.2.12 (Logical Relations for **MiniML** in *UnrTyp*). *For any Δ , $\rho \in \mathcal{D}[\Delta]$, and τ , if $\Delta \vdash \tau$, then $\mathcal{V}[\tau]_\rho \in \text{UnrTyp}$.*

Proof. First, we show $\mathcal{V}[\tau]_\rho \in \text{Typ}$. By the definition of *Typ*, it suffices to show, for all natural numbers n , $[\mathcal{V}[\tau]_\rho]_n \in \text{Typ}_n$, for which we must show two facts: first, that it is in 2^{AtomVal_n} , and second that it is closed under world extension. The latter holds by Lemma B.2.8. For the former, we note that we are required to show that the worlds are in World_n , which holds by definition, and that for any $(W, (\Phi_1, v_1), (\Phi_2, v_2))$ in the relation, $\Phi_1, \Phi_2 : W$. For the latter, note that $\Phi_1 = \Phi_2 = \emptyset$ as shown earlier, and \emptyset is trivially disjoint from $\text{flags}(W, 1)$ and $\text{flags}(W, 2)$.

Second, we show that for any $(W, (\Phi_1, v_1), (\Phi_2, v_2)) \in \mathcal{V}[\tau]_\rho$, $\Phi_1 = \Phi_2 = \emptyset$. This is trivial by the definition of $\mathcal{V}[\tau]_\rho$, aside from the case for α , where it holds because the relation is drawn from *UnrTyp*. \square

Lemma B.2.13 (Compositionality).

$$(W, (\Phi_1, v_1), (\Phi_2, v_2)) \in \mathcal{V}[\tau]_{\rho[\alpha \mapsto \mathcal{V}[\tau']_\rho]} \iff (W, (\Phi_1, v_1), (\Phi_2, v_2)) \in \mathcal{V}[\tau'/\alpha]_\rho$$

Proof. The proof for compositionality in this case study is essentially verbatim the proof for compositionality in the last case study. \square

Lemma B.2.14 (Expression Relation for Closed Types). *For any **MiniML** type τ where $\cdot \vdash \tau$ and any ρ ,*

$$\mathcal{E}[\tau]_\rho = \mathcal{E}[\tau].$$

Proof. Since $\mathcal{E}[\tau]_\rho$ is defined in terms of $\mathcal{V}[\tau]_\rho$, this proof is analogous to Lemma B.2.13, though since what we are substituting is not used, the interpretation can be arbitrary. \square

Lemma B.2.15 (Closing **MiniML** Terms). *For any **MiniML** term e where $\Gamma; \Omega; \Delta; \Gamma \vdash e : \tau$, for any $W, \gamma_\Gamma, \gamma_\Omega, \rho$ where $\rho \in \mathcal{D}[\Delta]$, $(W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho$, $(W, \emptyset, \emptyset, \gamma_\Omega) \in \mathcal{G}[\Omega]_\rho$, and $(W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega]_\rho$, it holds that*

$$\gamma_\Gamma^1(\gamma_\Gamma^1(\gamma_\Omega^1(e^+)))$$

and

$$\gamma_\Gamma^2(\gamma_\Gamma^2(\gamma_\Omega^2(e^+)))$$

are closed terms.

Proof. Since free variables are compiled to free variables, and no other free variables are introduced via compilation, this follows trivially from the structure of $\mathcal{G}[\Gamma]_\rho$. \square

Lemma B.2.16 (Closing **Affi** Terms). *For any **Affi** term e where $\Delta; \Gamma; \Omega \vdash e : \tau$, for any $W, \gamma_\Gamma, \gamma_\Omega, \rho$ where $\rho \in \mathcal{D}[\Delta]$, $(W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho$, $(W, \emptyset, \emptyset, \gamma_\Omega) \in \mathcal{G}[\Omega]_.$, and $(W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega]_.$, it holds that*

$$\gamma_\Gamma^1(\gamma_\Gamma^1(\gamma_\Omega^1(e^+)))$$

and

$$\gamma_\Gamma^2(\gamma_\Gamma^2(\gamma_\Omega^2(e^+)))$$

are closed terms.

Proof. Since free variables are compiled to free variables, and no other free variables are introduced via compilation, this follows trivially from the structure of $\mathcal{G}[\Gamma]_\rho$. \square

Lemma B.2.17 (**MiniML** Values Contain No Flags). *If $\Delta \vdash \tau$, $\rho \in \mathcal{D}[\Delta]$, and $(W, (\Phi_1, v_1), (\Phi_2, v_2)) \in \mathcal{V}[\tau]_\rho$, then $\Phi_1 = \Phi_2 = \emptyset$.*

Proof. If τ is not a type variable, then the theorem is trivially true because all non-type variable interpretations of **MiniML** types are defined to only contain tuples where the sets of static flags are \emptyset .

If τ is some type variable α , then, since $\Delta \vdash \tau$, $\alpha \in \Delta$. Thus, since $\rho \in \mathcal{D}[\Delta]$, it must be that $\rho(\alpha) \in UnrTyp$. Then, for any $(W, (\Phi_1, v_1), (\Phi_2, v_2)) \in \mathcal{V}[\tau]_\rho = \rho(\alpha)$, it must be that $\Phi_1 = \Phi_2 = \emptyset$ by the definition of *UnrTyp*. \square

Theorem B.2.18 (Convertibility Soundness). *If $\tau_A \sim \tau_B$ then*

$$\begin{aligned} \forall & (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\tau_A]. & \Rightarrow \\ (W, (\Phi_1, C_{\tau_A \mapsto \tau_B}(e_1)), (\Phi_2, C_{\tau_A \mapsto \tau_B}(e_2))) & \in \mathcal{E}[\tau_B]. \\ \wedge & \forall (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\tau_B]. & \Rightarrow \\ (W, (\Phi_1, C_{\tau_B \mapsto \tau_A}(e_1)), (\Phi_2, C_{\tau_B \mapsto \tau_A}(e_2))) & \in \mathcal{E}[\tau_A]. \end{aligned}$$

Proof. We prove this by simultaneous induction on the structure of the convertibility relation.

unit \sim unit There are two directions to this proof:

$$\begin{aligned} \forall & (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\text{unit}]. \\ \Rightarrow & (W, (\Phi_1, C_{\text{unit} \mapsto \text{unit}}(e_1)), (\Phi_2, C_{\text{unit} \mapsto \text{unit}}(e_2))) \in \mathcal{E}[\text{unit}]. \end{aligned}$$

and:

$$\begin{aligned} \forall & (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\text{unit}]. \\ \Rightarrow & (W, (\Phi_1, C_{\text{unit} \mapsto \text{unit}}(e_1)), (\Phi_2, C_{\text{unit} \mapsto \text{unit}}(e_2))) \in \mathcal{E}[\text{unit}]. \end{aligned}$$

Both directions are trivially similar to each other, so we will only prove the first direction.

Expanding the definition of the convertibility boundaries, we refine this to:

$$\forall (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\text{unit}]. \implies (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\text{unit}].$$

From the expression relation, we first need to show e_1, e_2 are closed. This follows directly from the fact the assumption that $(W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\text{unit}]$, and all terms in the expression relation are closed. Next, we need to show that given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \quad \Phi_{r1}, \Phi_{r2} : W \wedge \\ & \quad \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_{r1}, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

Then it holds that:

$$\begin{aligned} e'_1 = & \text{fail CONV} \vee (\exists \Phi_{f1} \Phi_{g1} \Phi_{f2} \Phi_{g2} v_2 H'_2 W'. \\ & \quad \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \quad \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \quad \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \quad \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau].) \} \end{aligned}$$

By instantiating the assumption $(W, (\Phi_1, e_1), (\Phi_1, e_2)) \in \mathcal{E}[\text{unit}]$. with $\Phi_{r1}, \Phi_{r2}, H_1, H_2$, etc, we find that

$$\begin{aligned} e'_1 = & \text{fail CONV} \vee (\exists \Phi_{f1} \Phi_{g1} \Phi_{f2} \Phi_{g2} v_2 H'_2 W'. \\ & \quad \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \quad \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \quad \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \quad \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau].) \} \end{aligned}$$

Ergo, it suffices to show that if $(W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\text{unit}]$, then $(W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\text{unit}]$. However, this is trivial because $\mathcal{V}[\text{unit}] = \mathcal{V}[\text{unit}] = \{(W, (\emptyset, ()), (\emptyset, ()))\}$.

int ~ bool There are two directions to this proof:

$$\begin{aligned} & \forall (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\text{int}]. \\ & \implies (W, (\Phi_1, C_{\text{int} \leftrightarrow \text{bool}}(e_1)), (\Phi_2, C_{\text{int} \leftrightarrow \text{bool}}(e_2))) \in \mathcal{E}[\text{bool}]. \end{aligned}$$

and:

$$\begin{aligned} & \forall (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\![\text{int}]\!]. \\ & \implies (W, (\Phi_1, C_{\text{bool} \leftrightarrow \text{int}}(e_1)), (\Phi_2, C_{\text{bool} \leftrightarrow \text{int}}(e_2))) \in \mathcal{E}[\![\text{int}]\!]. \end{aligned}$$

First, consider the first direction.

Expanding the definition of the convertibility boundaries, we refine this to:

$$\forall (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\![\text{int}]\!]. \implies (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\![\text{bool}]\!].$$

From the expression relation, we first need to show e_1, e_2 are closed. This follows directly from the fact the assumption that $(W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\![\text{int}]\!]$, and all terms in the expression relation are closed. Next, we need to show that given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1}, \Phi_{r2} : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_1, H'_1, e'_1 \rangle \not\rightarrow \end{aligned}$$

Then it holds that:

$$\begin{aligned} e'_1 = & \text{fail CONV} \vee (\exists \Phi_{f1} \Phi_{g1} \Phi_{f2} \Phi_{g2} v_2 H'_2 W'. \\ & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle \\ & \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \not\rightarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\![\text{bool}]\!].) \} \end{aligned}$$

By instantiating the assumption $(W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\![\text{int}]\!]$. with $\Phi_{r1}, \Phi_{r2}, H_1, H_2$, etc, we find that

$$\begin{aligned} e'_1 = & \text{fail CONV} \vee (\exists \Phi_{f1} \Phi_{g1} \Phi_{f2} \Phi_{g2} v_2 H'_2 W'. \\ & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle \\ & \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \not\rightarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\![\text{int}]\!].) \} \end{aligned}$$

Ergo, it suffices to show that if $(W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\![\text{int}]\!]$, then $(W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\![\text{bool}]\!]$. However, this is trivial because $\mathcal{V}[\![\text{int}]\!] \subseteq \mathcal{V}[\![\text{bool}]\!]$.

Next, consider the first direction.

Expanding the definition of the convertibility boundaries, we refine this to:

$$\forall (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\![\text{bool}]\!]. \implies (W, (\Phi_1, \text{if } e_1 0 1), (\Phi_2, \text{if } e_2 0 1)) \in \mathcal{E}[\![\text{int}]\!].$$

Expanding the expression relation, we must show that given

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \Phi_1 \wedge \Phi_{r2} \# \Phi_2 \wedge \Phi_{r1} \uplus \Phi_1, \Phi_{r2} \uplus \Phi_2 : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, \text{if } e_1 0 1 \rangle \xrightarrow{j} \langle \Phi'_1, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

it holds that:

$$\begin{aligned} e'_1 = \text{fail CONV} \vee (\exists \Phi_{f1} \Phi_{g1} \Phi_{f2} \Phi_{g2} v_2 H'_2 W'). \\ (\Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, \text{if } e_2 0 1) \\ \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\![\text{int}]\!]_\rho) \end{aligned}$$

By applying $(W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\![\text{int}]\!]$, we find that $\langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, e_1 \rangle$ either steps to fail CONV, in which case the original configuration with $\text{if } e_1 0 1$ takes another step to fail CONV, or steps to an irreducible configuration

$$\langle \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1}, H_1^*, e_1^* \rangle$$

in which case $\langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle$ steps to an irreducible configuration

$$\langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H_2^*, e_2^* \rangle$$

and there exists some world W' such that $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W', H_1^*, H_2^* : W'$, and $(W', (\Phi_{f1}, e_1^*), (\Phi_{f2}, e_2^*)) \in \mathcal{V}[\![\text{bool}]\!]_\rho$. By expanding the value relation, we find $\Phi_{f1} = \Phi_{f2} = \emptyset$ and there are two cases:

1. $e_1^* = e_2^* = 0$. In this scenario, we have

$$\begin{aligned} & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, \text{if } e_1 0 1 \rangle \xrightarrow{*} \\ & \langle \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1}, H_1^*, \text{if } 0 0 1 \rangle \rightsquigarrow \\ & \langle \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1}, H_1^*, 0 \rangle \end{aligned}$$

and

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, \text{if } e_2 0 1 \rangle \xrightarrow{*} \\ & \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H_2^*, \text{if } 0 0 1 \rangle \rightsquigarrow \\ & \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H_2^*, 0 \rangle \end{aligned}$$

Then, we have from before that $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W'$ and $H_1^*, H_2^* : W'$, and one can easily see that $(W', (\emptyset, 0), (\emptyset, 0)) \in \mathcal{V}[\![\text{int}]\!]$, which suffices to finish the proof.

2. $e_1^* = n_1$ and $e_2^* = n_2$ with $n_1, n_2 \neq 0$. In this scenario, we have

$$\begin{aligned} & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, \text{if } e_1 \neq 0 \rangle \xrightarrow{*} \\ & \langle \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1}, H_1^*, \text{if } n_1 \neq 0 \rangle \xrightarrow{*} \\ & \langle \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1}, H_1^*, 1 \rangle \end{aligned}$$

and

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, \text{if } e_2 \neq 0 \rangle \xrightarrow{*} \\ & \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H_2^*, \text{if } n_2 \neq 0 \rangle \xrightarrow{*} \\ & \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H_2^*, 1 \rangle \end{aligned}$$

Then, we have from before that $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W'$ and $H_1^*, H_2^* : W'$, and one can easily see that $(W', (\emptyset, 1), (\emptyset, 1)) \in \mathcal{V}[\![\text{int}]\!]$, which suffices to finish the proof.

$\tau_1 \otimes \tau_2 \sim \tau_1 \times \tau_2$ There are two directions to this proof:

$$\begin{aligned} & \forall (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\![\tau_1 \otimes \tau_2]\!]. \\ & \implies (W, (\Phi_1, C_{\tau_1 \otimes \tau_2 \mapsto \tau_1 \times \tau_2}(e_1)), (\Phi_2, C_{\tau_1 \otimes \tau_2 \mapsto \tau_1 \times \tau_2}(e_2))) \in \mathcal{E}[\![\tau_1 \times \tau_2]\!]. \end{aligned}$$

and:

$$\begin{aligned} & \forall (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\![\tau_1 \times \tau_2]\!]. \\ & \implies (W, (\Phi_1, C_{\tau_1 \times \tau_2 \mapsto \tau_1 \otimes \tau_2}(e_1)), (\Phi_2, C_{\tau_1 \times \tau_2 \mapsto \tau_1 \otimes \tau_2}(e_2))) \in \mathcal{E}[\![\tau_1 \otimes \tau_2]\!]. \end{aligned}$$

Both directions are trivially similar to each other, so we will only prove the first direction.

Expanding the definition of the convertibility boundaries, we refine this to:

$$\begin{aligned} & \forall (W, (\Phi_1, e_1), (\Phi_1, e_2)) \in \mathcal{E}[\![\tau_1 \otimes \tau_2]\!]. \implies \\ & (W, \\ & (\Phi_1, \text{let } x = e_1 \text{ in } (C_{\tau_1 \mapsto \tau_1}(\text{fst } x), C_{\tau_2 \mapsto \tau_2}(\text{snd } x))), \\ & (\Phi_2, \text{let } x = e_2 \text{ in } (C_{\tau_1 \mapsto \tau_1}(\text{fst } x), C_{\tau_2 \mapsto \tau_2}(\text{snd } x)))) \in \mathcal{E}[\![\tau_1 \times \tau_2]\!]. \end{aligned}$$

From the expression relation, we first need to show the two expressions in the conclusion are closed. This follows from the fact that e_1, e_2 are closed, by the assumption that $(W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\![\tau_1 \otimes \tau_2]\!]$, and that the new expressions do not introduce any new free variables. Next, we need to show that given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1}, \Phi_{r2} : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, \text{let } x = e_1 \text{ in } (C_{\tau_1 \mapsto \tau_1}(\text{fst } x), C_{\tau_2 \mapsto \tau_2}(\text{snd } x)) \rangle \\ & \xrightarrow{j} \langle \Phi'_1, H'_1, e'_1 \rangle \end{aligned}$$

Then it holds that:

$$\begin{aligned} e'_1 = \text{fail CONV} \vee (\exists \Phi_{f1} \Phi_{g1} \Phi_{f2} \Phi_{g2} v_2 H'_2 W'. \\ & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, \text{let } x = e_2 \text{ in } (C_{\tau_1 \mapsto \tau_1}(\text{fst } x), C_{\tau_2 \mapsto \tau_2}(\text{snd } x)) \rangle \\ & \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau_1 \times \tau_2]) \} \end{aligned}$$

First, since the let expression in the first configuration terminates to an irreducible configuration, by inspection on the operational semantic, it must be the case that $\langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, e_1 \rangle$ terminates to some irreducible configuration $\langle \Phi_1^*, H_1^*, e_1^* \rangle$. Then, by assumption, it follows that either $e_1^* = \text{fail CONV}$, in which case the whole let expression steps to fail CONV, or that e_1^* is a value, in which case $\langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle$ also steps to some irreducible configuration $\langle \Phi_2^*, H_2^*, e_2^* \rangle$ and there exists some world W_1 where $\Phi_i^* = \Phi_{ri} \uplus \text{flags}(W_1, i) \uplus \Phi_i^\dagger$, $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$, $H_1^*, H_2^* : W_1$, and $(W_1, (\Phi_1^*, e_1^*), (\Phi_2^*, e_2^*)) \in \mathcal{V}[\tau_1 \otimes \tau_2]$. By expanding the value relation definition, we find that $e_1^* = (v_1^*, v_2^*)$ and $e_2^* = (v_1^\dagger, v_2^\dagger)$ where $(W_1, (\Phi_{1a}, v_1^*), (\Phi_{2a}, v_1^\dagger)) \in \mathcal{V}[\tau_1]$ and $(W_1, (\Phi_{1b}, v_2^*), (\Phi_{2b}, v_2^\dagger)) \in \mathcal{V}[\tau_2]$, where $\Phi_1^\dagger = \Phi_{1a} \uplus \Phi_{1b}$ and $\Phi_2^\dagger = \Phi_{2a} \uplus \Phi_{2b}$.

Thus, the first configuration steps as follows:

$$\begin{aligned} & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, \text{let } x = e_1 \text{ in } (C_{\tau_1 \mapsto \tau_1}(\text{fst } x), C_{\tau_2 \mapsto \tau_2}(\text{snd } x)) \rangle \xrightarrow{*} \\ & \langle \Phi_{r1} \uplus \text{flags}(W_1, 1) \uplus \Phi_1^\dagger, H_1^*, \text{let } x = (v_1^*, v_2^*) \text{ in } (C_{\tau_1 \mapsto \tau_1}(\text{fst } x), C_{\tau_2 \mapsto \tau_2}(\text{snd } x)) \rangle \rightarrow \\ & \langle \Phi_{r1} \uplus \text{flags}(W_1, 1) \uplus \Phi_1^\dagger, H_1^*, (C_{\tau_1 \mapsto \tau_1}(\text{fst } (v_1^*, v_2^*)), C_{\tau_2 \mapsto \tau_2}(\text{snd } (v_1^*, v_2^*))) \rangle \rightarrow \\ & \langle \Phi_{r1} \uplus \text{flags}(W_1, 1) \uplus \Phi_1^\dagger, H_1^*, (C_{\tau_1 \mapsto \tau_1}(v_1^*), C_{\tau_2 \mapsto \tau_2}(v_2^*)) \rangle \end{aligned}$$

By a similar argument, the configuration on the other side with H_2 steps to

$$\langle \Phi_{r2} \uplus \text{flags}(W_1, 2) \uplus \Phi_2^\dagger, H_2^*, (C_{\tau_1 \mapsto \tau_1}(v_1^\dagger), C_{\tau_2 \mapsto \tau_2}(v_2^\dagger)) \rangle$$

Since $(W_1, (\Phi_{1a}, v_1^*), (\Phi_{2a}, v_1^\dagger)) \in \mathcal{V}[\tau_1] \subseteq \mathcal{E}[\tau_1]$ and $(W_1, (\Phi_{1b}, v_2^*), (\Phi_{2b}, v_2^\dagger)) \in \mathcal{V}[\tau_2] \subseteq \mathcal{E}[\tau_2]$, by the induction hypothesis, we have that

$$(W_1, (\Phi_{1a}, C_{\tau_1 \mapsto \tau_1}(v_1^*)), (\Phi_{2a}, C_{\tau_1 \mapsto \tau_1}(v_1^\dagger))) \in \mathcal{E}[\tau_1].$$

and

$$(W_1, (\Phi_{1b}, C_{\tau_2 \mapsto \tau_2}(v_2^*)), (\Phi_{2b}, C_{\tau_2 \mapsto \tau_2}(v_2^\dagger))) \in \mathcal{E}[\tau_2].$$

By the first fact, either $\langle \Phi_{r1} \uplus \Phi_{1b} \uplus \text{flags}(W_1, 1) \uplus \Phi_{1a}, H_1^*, C_{\tau_1 \mapsto \tau_1}(v_1^*) \rangle$ steps to fail CONV (note our choice of “rest” of flags includes those owned by the other half of the pair), in which case the original configuration with H_1 steps to fail CONV, or it steps to an irreducible configuration

$$\langle \Phi_{r1} \uplus \Phi_{1b} \uplus \text{flags}(W_2, 1) \uplus \Phi_{1a}^f, H_1^\dagger, v_1^{**} \rangle$$

in which case $\langle \Phi_{r2} \uplus \Phi_{2b} \uplus \text{flags}(W_1, 2) \uplus \Phi_{2a}, H_2^*, C_{\tau_1 \mapsto \tau_1}(v_1^\dagger) \rangle$ also steps to an irreducible configuration

$$\langle \Phi_{r2} \uplus \Phi_{2b} \uplus \text{flags}(W_2, 2) \uplus \Phi_{2a}^f, H_2^\dagger, v_1^{\dagger\dagger} \rangle$$

and there exists some world W_2 where $W_1 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{1b}, \Phi_{r2} \uplus \Phi_{2b}} W_2, H_1^\dagger, H_2^\dagger : W_2$, and $(W_2, (\Phi_{1a}^f, v_1^{**}), (\Phi_{2a}^f, v_1^{\dagger\dagger})) \in \mathcal{V}[\tau_1]\dots$

Once the first component of the pair in the configurations above have stepped to values v_1^{**} and $v_1^{\dagger\dagger}$, the pair will continue reducing on the second component. Then, by Lemma B.2.8, since $W_1 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{1b}, \Phi_{r2} \uplus \Phi_{2b}} W_2$ (which includes Φ_{1b} and Φ_{2b}),

$$(W_2, (\Phi_{1b}, C_{\tau_2 \mapsto \tau_2}(v_2^*)), (\Phi_{2b}, C_{\tau_2 \mapsto \tau_2}(v_2^\dagger))) \in \mathcal{E}[\tau_2].$$

Thus, either $\langle \Phi_{r1} \uplus \Phi_{1a}^f \uplus \text{flags}(W_2, 1) \uplus \Phi_{1b}, H_1^\dagger, C_{\tau_2 \mapsto \tau_2}(v_2^*) \rangle$ steps to fail CONV, in which case the original configuration also takes a step to fail CONV, or it steps to an irreducible configuration

$$\langle \Phi_{r1} \uplus \Phi_{1a}^f \uplus \text{flags}(W_3, 1) \uplus \Phi_{1b}^f, H_1^f, v_2^{**} \rangle$$

in which case $\langle \Phi_{r2} \uplus \Phi_{2a}^f \uplus \text{flags}(W_2, 1) \uplus \Phi_{2b}, H_2^\dagger, C_{\tau_2 \mapsto \tau_2}(v_2^\dagger) \rangle$ also steps to an irreducible configuration $\langle \Phi_{r2} \uplus \Phi_{2a}^f \uplus \text{flags}(W_3, 1) \uplus \Phi_{2b}^f, H_2^f, v_2^{\dagger\dagger} \rangle$ and there exists some world W_3 where

$$W_2 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{1a}^f, \Phi_{r2} \uplus \Phi_{2a}^f} W_3, H_1^f, H_2^f : W_3, \text{ and } (W_3, (\Phi_{1b}^f, v_2^{**}), (\Phi_{2b}^f, v_2^{\dagger\dagger})) \in \mathcal{V}[\tau_2]\dots$$

Thus, the original configuration with H_1 and $\Phi_1 \uplus \Phi_2$ steps to $\langle \Phi_r \uplus \text{flags}(W_3, 1) \uplus \Phi_{1a}^f \uplus \Phi_{1b}^f, H_1^f, (v_1^{**}, v_2^{**}) \rangle$ and the original configuration with H_2 steps to $\langle \Phi_r \uplus \text{flags}(W_3, 2) \uplus \Phi_{2a}^f \uplus \Phi_{2b}^f, H_2^f, (v_1^{\dagger\dagger}, v_2^{\dagger\dagger}) \rangle$. We have $H_1^f, H_2^f : W_3$ and, since $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$, $W_1 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{1b}^\dagger, \Phi_{r2} \uplus \Phi_{2b}^\dagger} W_2$, and $W_2 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{1a}^f, \Phi_{r2} \uplus \Phi_{2a}^f} W_3$, it follows from Lemma B.2.9 that $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_3$. Moreover, since $W_2 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{1a}^f, \Phi_{r2} \uplus \Phi_{2a}^f} W_3$ and $(W_2, (\Phi_{1a}^f, v_1^{**}), (\Phi_{2a}^f, v_1^{\dagger\dagger})) \in \mathcal{E}[\tau_2]$,

$\mathcal{V}[\tau_1]..$, we have $(W_3, (\Phi_{1a}^f, v_1^{**}), (\Phi_{2a}^f, v_1^{\dagger\dagger})) \in \mathcal{V}[\tau_1]..$. Finally, we also have $(W_3, (\Phi_{1b}^f, v_2^{**}), (\Phi_{2b}^f, v_2^{\dagger\dagger})) \in \mathcal{V}[\tau_2]..$ Ergo,

$$(W_3, (\Phi_{1a}^f \uplus \Phi_{1b}^f, (v_1^{**}, v_2^{**})), (\Phi_{2a}^f \uplus \Phi_{2b}^f, (v_1^{\dagger\dagger}, v_2^{\dagger\dagger}))) \in \mathcal{V}[\tau_1 \times \tau_2].$$

which suffices to finish the proof.

$$\boxed{\tau_1 \multimap \tau_2 \sim (\text{unit} \rightarrow \tau_1) \rightarrow \tau_2}$$

There are two directions, we first prove the former implication, that is, that:

$$\begin{aligned} \forall (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[\tau_1 \multimap \tau_2]. \implies \\ (W, (\Phi_1, C_{\tau_1 \multimap \tau_2 \mapsto (\text{unit} \rightarrow \tau_1) \rightarrow \tau_2}(e_1)), \\ (\Phi_2, C_{\tau_1 \multimap \tau_2 \mapsto (\text{unit} \rightarrow \tau_1) \rightarrow \tau_2}(e_2))) \in \mathcal{E}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]. \end{aligned}$$

Expanding the definition of the convertibility boundaries, we refine our goal to:

$$\begin{aligned} (W, (\Phi_1, \text{let } x = e_1 \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} ()) \\ \text{in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(x x_{\text{access}})), \\ (\Phi_2, \text{let } x = e_2 \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} ()) \\ \text{in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(x x_{\text{access}})) \\ \in \mathcal{E}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]. \end{aligned}$$

From the expression relation, we must show first that the terms are closed, which follows from our hypothesis given we did not introduce any new free variables. Then, we need to show that given:

$$\begin{aligned} \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ \Phi_{r1}, \Phi_{r2} : W \wedge \\ \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, \text{let } x = e_1 \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} ()) \text{ in } \\ \text{let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(x x_{\text{access}}) \\ \stackrel{j}{\dashrightarrow} \langle \Phi'_1, H'_1, e'_1 \rangle \not\rightarrow \end{aligned}$$

Then it holds that:

$$\begin{aligned}
e'_1 = & \text{fail CONV} \vee (\exists \Phi_{f1} \Phi_{g1} \Phi_{f2} \Phi_{g2} v_2 H'_2 W' \\
& \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, \text{let } x = e_2 \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}}()) \text{ in } \\
& \quad \text{let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(x x_{\text{access}}) \\
& \quad \dashv^* \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \dashv \\
& \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\
& \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\
& \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]) \}
\end{aligned}$$

To figure out what e'_1 is, we know from the operational semantics that first we will evaluate e_1 until it is a value and then will substitute. From our hypothesis, which we can instantiate with $\Phi_{r1}, \Phi_{r2}, H_1, H_2$, etc, we know that either e_1 will run forever, in which case the entire term will and we are done (trivially). Otherwise, we have that:

$$\langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, e_1 \rangle \dashv^j \langle \overline{\Phi_1}, H_1^\dagger, e_1^\dagger \rangle \dashv$$

And that:

$$\begin{aligned}
e_1^\dagger = & \text{fail CONV} \vee (\exists \Phi_{f1} \Phi_{g1} \Phi_{f2} \Phi_{g2} v_2 H'_2 W' \\
& \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle \\
& \quad \dashv^* \langle \Phi_{r2} \uplus \text{flags}(W^\dagger, 2) \uplus \Phi_2^\dagger \uplus \Phi_{g2}, H_2^\dagger, e_2^\dagger \rangle \dashv \\
& \wedge \overline{\Phi_1} = \Phi_{r1} \uplus \text{flags}(W^\dagger, 1) \uplus \Phi_1^\dagger \uplus \Phi_{g1} \wedge \\
& \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W^\dagger \wedge H_1^\dagger, H_2^\dagger : W^\dagger \\
& \wedge (W^\dagger, (\Phi_1^\dagger, e_1^\dagger), (\Phi_2^\dagger, e_2^\dagger)) \in \mathcal{V}[\tau_1 \multimap \tau_2]) \}
\end{aligned}$$

Where if e_1^\dagger is fail CONV then the operational semantics will lift that to the entire term and we will be done. Note also that from the definition of $\mathcal{V}[\tau_1 \multimap \tau_2]$, we know $\Phi_i^\dagger = \emptyset$.

Now, returning to our original reduction, we will take another step and substitute e_1^\dagger for x , which results in the following term:

$$\lambda x_{\text{thnk}}. \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}}()) \text{ in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \text{ in } C_{\tau_2 \mapsto \tau_2}(e_1^\dagger x_{\text{access}})$$

This is clearly irreducible (it is a value), so we now need to show that the other side similarly reduces to a value, which follows in the same way from our hypothesis, and thus what remains to show is that these two values are related at W^\dagger in $\mathcal{V}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]$. (we choose W^\dagger because no changes to heap or flags happened in the substitution).

The definition of $\mathcal{V}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]$ says that we need to take any world W' , where $W^\dagger \sqsubset_{\emptyset, \emptyset} W'$, $(W', (\emptyset, v'_1), (\emptyset, v'_2)) \in \mathcal{V}[\text{unit} \rightarrow \tau_1]$. and show that

$$(W', (\emptyset, [x_{\text{thnk}} \mapsto v'_1] \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} ()) \text{ in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \\ \text{in } C_{\tau_2 \mapsto \tau_2}(e_1^\dagger x_{\text{access}})), \\ (\emptyset, [x_{\text{thnk}} \mapsto v'_2] \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}} ()) \text{ in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \\ \text{in } C_{\tau_2 \mapsto \tau_2}(e_2^\dagger x_{\text{access}})) \in \mathcal{E}[\tau_2].$$

Where if we substitute, we get:

$$(W', (\emptyset, \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(v'_1 ()) \text{ in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \\ \text{in } C_{\tau_2 \mapsto \tau_2}(e_1^\dagger x_{\text{access}})), \\ (\emptyset, \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(v'_2 ()) \text{ in let } x_{\text{access}} = \text{once}(x_{\text{conv}}) \\ \text{in } C_{\tau_2 \mapsto \tau_2}(e_2^\dagger x_{\text{access}})) \in \mathcal{E}[\tau_2].$$

Now we can expand the definition of once(\cdot), to get:

$$(W', (\emptyset, \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(v'_1 ()) \text{ in let } x_{\text{access}} = \\ (\text{let } r_{\text{fresh}} = \text{ref } 1 \text{ in } \lambda_{\text{--}}\{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := 0; x_{\text{conv}}\}\}) \\ \text{in } C_{\tau_2 \mapsto \tau_2}(e_1^\dagger x_{\text{access}})), \\ (\emptyset, \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(v'_2 ()) \text{ in let } x_{\text{access}} = \\ (\text{let } r_{\text{fresh}} = \text{ref } 1 \text{ in } \lambda_{\text{--}}\{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := 0; x_{\text{conv}}\}\}) \\ \text{in } C_{\tau_2 \mapsto \tau_2}(e_2^\dagger x_{\text{access}}))) \\ \in \mathcal{E}[\tau_2].$$

From our induction hypothesis, instantiated with $\triangleright W'$ we know $(\triangleright W', (\emptyset, C_{\tau_1 \mapsto \tau_1}(v'_1 ())), (\emptyset, C_{\tau_1 \mapsto \tau_1}(v'_2 ())))$ will be in $\mathcal{E}[\tau_1]$. if $(\triangleright W', (\emptyset, v'_1()), (\emptyset, v'_2()))$ is in $\mathcal{E}[\tau_1]$. But, since $(W', (\emptyset, v'_1), (\emptyset, v'_2)) \in \mathcal{V}[\text{unit} \rightarrow \tau_1]$, by definition the latter holds, since the only values in $\mathcal{V}[\text{unit}]$ are ().

This means we can unfold the definition of $\mathcal{E}[\tau_1]$. and know that for any $\Phi_{r1}, \Phi_{r2} : \triangleright W', H_1, H_2 : \triangleright W'$:

$$\langle \Phi_{r1} \uplus \text{flags}(\triangleright W', 1) \uplus \emptyset, H_1, C_{\tau_1 \mapsto \tau_1}(v'_1 ()) \rangle \xrightarrow{j} \langle \overline{\Phi_1}, H_{c1}, v_{c1} \rangle \rightarrow$$

Assuming v_{c1} is not fail CONV:

$$\exists \Phi_{c1} \Phi_{g1} \Phi_{c2} \Phi_{g2} v_{c2} H_{c2} W''. \\ \langle \Phi_{r2} \uplus \text{flags}(\triangleright W', 2) \uplus \emptyset, H_2, C_{\tau_1 \mapsto \tau_1}(v'_2 ()) \rangle \\ \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W^c, 2) \uplus \Phi_{c2} \uplus \Phi_{g2}, H_{c2}, v_{c2} \rangle \rightarrow \\ \wedge \overline{\Phi_1} = \Phi_{r1} \uplus \text{flags}(W'', 1) \uplus \Phi_{c1} \uplus \Phi_{g1} \wedge \\ \wedge \triangleright W' \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W'' \wedge H_{c1}, H_{c2} : W'' \\ \wedge (W'', (\Phi_{c1}, v_{c1}), (\Phi_{c2}, v_{c2})) \in \mathcal{V}[\tau_1 \multimap \tau_2]) \}$$

If we return to our original obligation, we need to show that for some $\Phi'_{r1}, \Phi'_{r2}, H'_1, H'_2 : W'$ that if:

$$\langle \Phi'_{r1} \uplus \text{flags}(W', 1) \uplus \emptyset, H_1, \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(v'_1 ()) \text{ in let } x_{\text{access}} = \\ (\text{let } r_{\text{fresh}} = \text{ref } 1 \text{ in } \lambda_{-}.\{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \\ \{r_{\text{fresh}} := 0; x_{\text{conv}}\}\}) \text{ in} \\ C_{\tau_2 \mapsto \tau_2}(e_1^{\dagger} x_{\text{access}}) \\ \dashrightarrow^j \langle \overline{\Phi_1}, H''_1, e'_1 \rangle \dashrightarrow$$

Then:

$$\exists \Phi''_1 \Phi_{g1} \Phi''_2 \Phi_{g2} e'_2 H''_2 W'''.$$

$$\langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \emptyset, H_2, \text{let } x_{\text{conv}} = C_{\tau_1 \mapsto \tau_1}(v'_2 ()) \text{ in} \\ \text{let } x_{\text{access}} = (\text{let } r_{\text{fresh}} = \text{ref } 1 \text{ in} \\ \lambda_{-}.\{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := 0; x_{\text{conv}}\}\}) \text{ in} \\ C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}}) \\ \dashrightarrow^{*} \langle \Phi_{r2} \uplus \text{flags}(W^c, 2) \uplus \Phi_{c2} \uplus \Phi_{g2}, H_{c2}, v_{c2} \rangle \dashrightarrow \\ \wedge \overline{\Phi_1} = \Phi_{r1} \uplus \text{flags}(W'', 1) \uplus \Phi_{c1} \uplus \Phi_{g1} \wedge \\ \wedge \triangleright W' \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W'' \wedge H_{c1}, H_{c2} : W'' \\ \wedge (W'', (\Phi_{c1}, v_{c1}), (\Phi_{c2}, v_{c2})) \in \mathcal{V}[\tau_1 \multimap \tau_2] \rangle \}$$

If we choose Φ'_{ri} to be that chosen above, we know $C_{\tau_1 \mapsto \tau_1}(v'_1 ())$ reduces to v_{c2} with Φ_{c1} , and thus the entire term takes a step to:

$$\langle \Phi_{r1} \uplus \text{flags}(W'', 1) \uplus \Phi_{c1}, H_{c1}, \\ \text{let } x_{\text{conv}} = v_{c1} \text{ in let } x_{\text{access}} = \\ (\text{let } r_{\text{fresh}} = \text{ref } 1 \text{ in } \lambda_{-}.\{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := 0; x_{\text{conv}}\}\}) \text{ in} \\ C_{\tau_2 \mapsto \tau_2}(e_1^{\dagger} x_{\text{access}})$$

Which then takes two more steps to:

$$\langle \Phi_{r1} \uplus \text{flags}(W'', 1) \uplus \Phi_{c1}, H_{c1}, \\ C_{\tau_2 \mapsto \tau_2}(e_1^{\dagger} \\ (\text{let } r_{\text{fresh}} = \text{ref } 1 \text{ in } \lambda_{-}.\{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := 0; v_{c1}\}\})) \rangle$$

To figure out where that steps next, we need to appeal to our induction hypothesis. In particular, we instantiate it with W'' , which then tells us that:

$$(W'', (\Phi_{c1}, C_{\tau_2 \mapsto \tau_2}(e_1^\dagger (\text{let } r_{\text{fresh}} = \text{ref 1 in } \lambda_{-} \{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := 0; v_{c1}\}\}))), \\ (\Phi_{c2}, C_{\tau_2 \mapsto \tau_2}(e_2^\dagger (\text{let } r_{\text{fresh}} = \text{ref 1 in } \lambda_{-} \{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := 0; v_{c2}\}\})))) \in \mathcal{E}[\tau_2].$$

If we can show:

$$(W'', (\Phi_{c1}, (e_1^\dagger (\text{let } r_{\text{fresh}} = \text{ref 1 in } \lambda_{-} \{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := 0; v_{c1}\}\}))), \\ (\Phi_{c2}, (e_2^\dagger (\text{let } r_{\text{fresh}} = \text{ref 1 in } \lambda_{-} \{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := 0; v_{c2}\}\})))) \in \mathcal{E}[\tau_2].$$

To show the latter, recall that $(W^\dagger, (\emptyset, e_1^\dagger), (\emptyset, e_2^\dagger)) \in \mathcal{V}[\tau_1 \multimap \tau_1]..$. We know that $W^\dagger \sqsubseteq_{\emptyset, \emptyset} W'$, $W' \sqsubseteq_{\emptyset, \emptyset} \triangleright W'$, and $\triangleright W' \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W''$, so via Lemma B.2.9, $W^\dagger \sqsubseteq_{\emptyset, \emptyset} W''$ and thus via Lemma B.2.8, $(W'', (\emptyset, e_1^\dagger), (\emptyset, e_2^\dagger)) \in \mathcal{V}[\tau_1 \multimap \tau_1]..$. In particular, we know that each have the form $\lambda x. e_i^*$.

That means, if we can show, for some Φ_{c1}, Φ_{c2} and some world W''' where $W'' \sqsubseteq_{\emptyset, \emptyset} W'''$, that

$$(W''', (\Phi_{c1}, v_{c1}), (\Phi_{c2}, v_{c2})) \in \mathcal{V}[\tau_2]. \text{ (which we have from before) then}$$

$$(W^A, (\emptyset, [x \mapsto \text{guard}(v_{c1}, \ell_1)]e_1^*), (\emptyset, [x \mapsto \text{guard}(v_{c2}, \ell_2)]e_2^*)) \in \mathcal{E}[\tau_1].$$

Where $W^A = (W'''.k, W'''.\Psi, W'''.\Theta \uplus (\ell_1, \ell_2) \mapsto (\Phi_{c1}, \Phi_{c2}))$.

In particular, we let $W''' = W''$.

To connect these two together, we first unfold the former: the definition means that for any $\Phi''_{r1}, \Phi''_{r1} : W''$ and $H''_1, H''_2 : W''$, we need to show:

$$\langle \Phi''_{r1} \uplus \text{flags}(W'', 1) \uplus \Phi_{c1}, H''_1, (\lambda x. e_1^*) \left(\begin{array}{l} \text{let } r_{\text{fresh}} = \text{ref 1} \\ \text{in } \lambda_{-} \{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := 0; v_{c1}\}\} \end{array} \right) \rangle \\ \dashrightarrow^j \langle \overline{\Phi'''_1}, H'''_1, e'''_1 \rangle \dashrightarrow$$

The latter will give us the reduction, for $\Phi^A_{r1}, \Phi^A_{r2} : W^A$ and $H^A_1, H^A_2 : W^A$:

$$\langle \Phi^A_{r1} \uplus \text{flags}(W^A, 1) \uplus \emptyset, H^A_1, [x \mapsto \text{guard}(v_{c1}, \ell_1)]e_1^* \rangle \dashrightarrow^j \langle \overline{\Phi^A_1}, H^B_1, e_1^B \rangle \dashrightarrow$$

In particular, since W^A is identical to W'' aside from gaining Φ_{c1}, Φ_{c2} , we can use Φ''_{ri} as Φ^A_{ri} and $\text{flags}(W'') \uplus \Phi_{c1} = \text{flags}(W^A) \uplus \emptyset$.

Thus, the former takes one step to the latter, and the rest of what we need follows.

We now return to our original goal, that is, showing how this reduces:

$$\langle \Phi_{r1} \uplus \text{flags}(W'', 1) \uplus \Phi_{c1}, H_{c1}, C_{\tau_2 \mapsto \tau_2}(e_1^\dagger \\ (\text{let } r_{\text{fresh}} = \text{ref } 1 \\ \text{in } \lambda_. \{ \text{if } !r_{\text{fresh}} \{ \text{fail CONV} \} \{ r_{\text{fresh}} := 0; v_{c1} \} \})) \rangle$$

Since we now know:

$$(W'', \\ (\Phi_{c1}, C_{\tau_2 \mapsto \tau_2}(e_1^\dagger (\text{let } r_{\text{fresh}} = \text{ref } 1 \text{ in } \lambda_. \{ \text{if } !r_{\text{fresh}} \{ \text{fail CONV} \} \{ r_{\text{fresh}} := 0; v_{c1} \} \}))), \\ (\Phi_{c2}, C_{\tau_2 \mapsto \tau_2}(e_2^\dagger (\text{let } r_{\text{fresh}} = \text{ref } 1 \text{ in } \lambda_. \{ \text{if } !r_{\text{fresh}} \{ \text{fail CONV} \} \{ r_{\text{fresh}} := 0; v_{c2} \} \})))) \\ \in \mathcal{E}[\tau_2].$$

We can unfold the definition and get exactly what we need, as what we were originally showing was that the term in question was in $\mathcal{E}[\tau_2]$.

Thus, we are done with the first direction.

Now we have to prove the other direction, that is, that:

$$\forall (W, (\Phi_1, e_1), (\Phi_2, e_2)) \in \mathcal{E}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]. \implies \\ (W, (\Phi_1, C_{(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2 \mapsto \tau_1 \multimap \tau_2}(e_1)), \\ (\Phi_2, C_{(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2 \mapsto \tau_1 \multimap \tau_2}(e_2))) \in \mathcal{E}[\tau_1 \multimap \tau_2].$$

Expanding the definition of the convertibility boundaries, we refine our goal to:

$$(W, \\ (\Phi_1, \text{let } x = e_1 \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}})) \text{ in } C_{\tau_2 \mapsto \tau_2}(x x_{\text{access}})), \\ (\Phi_2, \text{let } x = e_2 \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}})) \text{ in } C_{\tau_2 \mapsto \tau_2}(x x_{\text{access}})) \\ \in \mathcal{E}[\tau_1 \multimap \tau_2].$$

From the expression relation, we must show first that the terms are closed, which follows from our hypothesis given we did not introduce any new free variables. Then, we need to show that given:

$$\forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ \Phi_{r1}, \Phi_{r2} : W \wedge \\ \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, \\ \text{let } x = e_1 \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}})) \text{ in } \\ C_{\tau_2 \mapsto \tau_2}(x x_{\text{access}}) \\ \dashv^j \langle \Phi'_1, H'_1, e'_1 \rangle \dashv \rangle$$

Then it holds that:

$$\begin{aligned}
 e'_1 = & \text{fail CONV} \vee (\exists \Phi_{f1} \Phi_{g1} \Phi_{f2} \Phi_{g2} v_2 H'_2 W' \\
 & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, \frac{\text{let } x = e_2 \text{ in } \lambda x_{\text{thnk}}. \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}})) \text{ in } }{C_{\tau_2 \mapsto \tau_2}(x x_{\text{access}})} \rangle \\
 & \dashrightarrow^* \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\
 & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\
 & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\
 & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau_1 \multimap \tau_2])
 \end{aligned}$$

To figure out what e'_1 is, we know from the operational semantics that first we will evaluate e_1 until it is a value and then will substitute. From our hypothesis, which we can instantiate with $\Phi_{r1}, \Phi_{r2}, H_1, H_2$, etc, we know that e_1 will run with either fail CONV (in which case this will lift into the entire term running to fail CONV) or will run to a value v_1 related in $\mathcal{V}[(\text{unit} \rightarrow \tau_1) \rightarrow \tau_2]$. at a future world W^\dagger where $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W^\dagger$ to another value v_2 that e_2 will run to, where the heaps have evolved to $H'_1, H'_2 : W^\dagger$, and empty flag stores.

Now, our original term will take another step and substitute v_1 for x (note that the operational semantics lifts steps on the subterm to steps on the whole term), which results in the following term:

$$\lambda x_{\text{thnk}}. \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}})) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}})$$

This is clearly irreducible (it is a value), so we now need to show that the other side similarly reduces to a value v_2 , which follows in the same way from our hypothesis, and thus what remains to show is that:

$$\begin{aligned}
 & (W^\dagger, (\emptyset, \lambda x_{\text{thnk}}. \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}})) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}}))), \\
 & (\emptyset, \lambda x_{\text{thnk}}. \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}})) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}}))) \\
 & \in \mathcal{V}[\tau_1 \multimap \tau_2].
 \end{aligned}$$

The definition of $\mathcal{V}[\tau_1 \multimap \tau_2]$. says that we need to take any $W^\dagger \sqsubset W'$, $v'_1, v'_2, \ell_1, \ell_2$ where $(W^\dagger, (\Phi'_1, v'_1), (\Phi'_2, v'_2))$ are in $\mathcal{V}[\tau_1]$. and (ℓ_1, ℓ_2) are not in either $W'.\Psi$ or $W'.\Theta$ and show that

$$\begin{aligned}
& ((W'.k, W'.\Psi, W'.\Theta \uplus (\ell_1, \ell_2) \mapsto (\Phi'_1, \Phi'_2)), \\
& (\emptyset, [x_{\text{thnk}} \mapsto \text{guard}(v'_1, \ell_1)] \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}}())), \\
& \quad \text{in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}})), \\
& (\emptyset, [x_{\text{thnk}} \mapsto \text{guard}(v'_2, \ell_2)] \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(x_{\text{thnk}}())), \\
& \quad \text{in } C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}}))) \\
& \in \mathcal{E}[\![\tau_2]\!].
\end{aligned}$$

Where if we substitute (letting $W^* = (W'.k, W'.\Psi, W'.\Theta \uplus (\ell_1, \ell_2) \mapsto (\Phi'_1, \Phi'_2))$), we get:

$$\begin{aligned}
& (W^*, (\emptyset, \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ()) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}}))), \\
& (\emptyset, \text{let } x_{\text{access}} = \text{once}(C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_2, \ell_2) ()) \text{ in } C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}}))) \\
& \in \mathcal{E}[\![\tau_2]\!].
\end{aligned}$$

First, let's expand the definition of $\text{once}(\cdot)$:

$$\begin{aligned}
& (W^*, (\emptyset, \text{let } x_{\text{access}} = \text{let } r_{\text{fresh}} = \text{ref UNUSED in} \\
& \quad \lambda_{-}\{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ())\} \\
& \quad \text{in } C_{\tau_2 \mapsto \tau_2}(v_1 x_{\text{access}})) \\
& (\emptyset, \text{let } x_{\text{access}} = \text{let } r_{\text{fresh}} = \text{ref UNUSED in} \\
& \quad \lambda_{-}\{\text{if } !r_{\text{fresh}} \{\text{fail CONV}\} \{r_{\text{fresh}} := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_2, \ell_2) ())\} \\
& \quad \text{in } C_{\tau_2 \mapsto \tau_2}(v_2 x_{\text{access}})) \\
& \in \mathcal{E}[\![\tau_2]\!].
\end{aligned}$$

To understand what happens, consider the operational reductions: allocating a new reference (ℓ'_i) , substituting it for r_{fresh} , and then substituting all of x_{access} , and thus suffices to show that:

$$\begin{aligned}
& (W^\dagger, (\emptyset, C_{\tau_2 \mapsto \tau_2}(v_1 (\lambda_{-}\{\text{if } !\ell'_1 \{\text{fail CONV}\} \{\ell'_1 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ())\}\}))), \\
& (\emptyset, C_{\tau_2 \mapsto \tau_2}(v_2 (\lambda_{-}\{\text{if } !\ell'_2 \{\text{fail CONV}\} \{\ell'_2 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_2, \ell_2) ())\}\}))) \\
& \in \mathcal{E}[\![\tau_2]\!].
\end{aligned}$$

Where W^\dagger has a new pair of references in $W^\dagger.\Theta$ (set to (\emptyset, \emptyset)) but otherwise is identical to W^* .

For this, we can appeal to our induction hypothesis, which requires us to show that:

$$\begin{aligned}
& (W^\dagger, (\emptyset, v_1 (\lambda_{-}\{\text{if } !\ell'_1 \{\text{fail CONV}\} \{\ell'_1 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ())\}\}), \\
& (\emptyset, v_2 (\lambda_{-}\{\text{if } !\ell'_2 \{\text{fail CONV}\} \{\ell'_2 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_2, \ell_2) ())\}\})) \\
& \in \mathcal{E}[\![\tau_2]\!].
\end{aligned}$$

Recalling that v_1 and v_2 came from $\mathcal{V}[\llbracket \text{unit} \rightarrow \tau_1 \rrbracket \rightarrow \tau_2]$, we can proceed by appealing to the definition of that relation, which tells us that for any arguments in $\mathcal{V}[\llbracket \text{unit} \rightarrow \tau_1 \rrbracket]$, the result of substituting will be in $\mathcal{E}[\llbracket \tau_2 \rrbracket]$. It thus remains to show that:

$$\begin{aligned} & (W^*, (\emptyset, \lambda_{-}\{\text{if } !\ell'_1 \{\text{fail CONV}\} \{\ell'_1 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ())\}\}), \\ & \quad (\emptyset, \lambda_{-}\{\text{if } !\ell'_2 \{\text{fail CONV}\} \{\ell'_2 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_2, \ell_2) ())\}\})) \\ & \in \mathcal{V}[\llbracket \text{unit} \rightarrow \tau_1 \rrbracket]. \end{aligned}$$

Where W^* is some future world of W^\dagger . From the definition of $\mathcal{V}[\llbracket \text{unit} \rightarrow \tau_1 \rrbracket]$, we have to show that substituting () for the unused argument results in terms in $\mathcal{E}[\llbracket \tau_1 \rrbracket]$, at some arbitrary future world W^{**} .

We proceed first by case analysis on whether the affine flags (ℓ'_1, ℓ'_2) have been set to USED, which they can be in a future world. If they have been, we can expand the definition of the expression relation, choose Φ_{r1} and heaps $H_1^{**}, H_2^{**} : W^{**}$, and show that

$$\begin{aligned} & \langle \Phi_{r1} \uplus \text{flags}(W^{**}, 1), H_1, \text{if } !\ell'_1 \{\text{fail CONV}\} \{\ell'_1 := \text{USED}; C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ())\} \rangle \xrightarrow{-2} \\ & \langle \Phi_{r1} \uplus \text{flags}(W^{**}, 1), H_1, \text{fail CONV} \rangle \end{aligned}$$

At which point we are done.

Thus, we now consider if (ℓ'_1, ℓ'_2) are still set to a pair of flag sets (Φ_a, Φ'_a) . If that's the case, we instead take three steps to move into the else branches and update the affine flags to USED. That means we reduce our task to showing that in a world W^{***} , which now has those locations marked used in Θ , we need to show:

$$(W^{***}, (\emptyset, C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_1, \ell_1) ()))), (\emptyset, C_{\tau_1 \mapsto \tau_1}(\text{guard}(v'_2, \ell_2) ()))) \in \mathcal{E}[\llbracket \tau_1 \rrbracket].$$

We now again appeal to our induction hypothesis, expanding the definition of $\text{guard}(\cdot)$ at the same time to yield the following obligation:

$$\begin{aligned} & (W^{***}, (\emptyset, (\lambda_{-}\{\text{if } !\ell_1 \{\text{fail CONV}\} \{\ell_1 := \text{USED}; v'_1\}\})()), \\ & \quad (\emptyset, (\lambda_{-}\{\text{if } !\ell_2 \{\text{fail CONV}\} \{\ell_2 := \text{USED}; v'_2\}\})()) \in \mathcal{E}[\llbracket \tau_1 \rrbracket]. \end{aligned}$$

We can then take one step, eliminating the pointless beta-reduction (for simplicity, we use the same name for the world, even though it is a future world):

$$\begin{aligned} & (W^{***}, (\emptyset, \text{if } !\ell_1 \{\text{fail CONV}\} \{\ell_1 := \text{USED}; v'_1\})), \\ & \quad (\emptyset, \text{if } !\ell_2 \{\text{fail CONV}\} \{\ell_2 := \text{USED}; v'_2\}) \in \mathcal{E}[\llbracket \tau_1 \rrbracket]. \end{aligned}$$

Now we again do case analysis on whether (ℓ_1, ℓ_2) is USED in $W^{***}.\Theta$. If it is, then, as before, we trivially reduce the left side to failure and are

done. If it is not, then we update those affine flags and reduce both sides to the values v'_1 and v'_2 , at a future world W^{final} . Now we knew, originally, that those values were in $\mathcal{V}[\tau_1]$. at world W^\dagger , but since, through many applications of Lemma B.2.9 and Lemma B.2.8, that also means that they are related at W^{final} , we are done.

□

Lemma B.2.19 (Phantom Steps Translate to Actual Steps). *For any expression e in the phantom LCVM language, let $Z(e)$ be an expression in the original LCVM language where every subexpression of the form $\text{protect}(e', f)$ is replaced with e' .*

For any heap H in the phantom LCVM language, let $Z_H(H) = \{\ell \mapsto Z(v) \mid \ell \mapsto v \in H\}$.

For any sets of flags Φ, Φ' , heaps H, H' , and expressions e, e' , if

$$\langle \Phi, H, e \rangle \xrightarrow{-m} \langle \Phi', H', e' \rangle$$

then

$$\langle Z_H(H), Z(e) \rangle \xrightarrow{n} \langle Z_H(H'), Z(e') \rangle$$

where n is the number of steps in the first reduction sequence which are not invoked by the following reduction rule

$$\langle \Phi \uplus \{f\}, H, \text{protect}(e, f) \rangle \dashrightarrow \langle \Phi, H, e \rangle \quad (15)$$

Proof. There exists some natural number j such that $\langle \Phi, H, e \rangle \xrightarrow{j} \langle \Phi', H', e' \rangle$. We will prove the theorem by induction on j .

If $j = 0$, then $\Phi = \Phi'$, $H = H'$, and $e = e'$. It is then trivial to show that $\langle Z_H(H), Z(e) \rangle \xrightarrow{0} \langle Z_H(H), Z(e) \rangle$, which finishes the proof for this case.

If $j > 0$, then there exist Φ_j, H_j, e_j such that

$$\langle \Phi, H, e \rangle \xrightarrow{j-1} \langle \Phi_j, H_j, e_j \rangle$$

and

$$\langle \Phi_j, H_j, e_j \rangle \dashrightarrow \langle \Phi', H', e' \rangle$$

By the induction hypothesis, we have

$$\langle Z_H(H), Z(e) \rangle \xrightarrow{n_j} \langle Z_H(H_j), Z(e_j) \rangle$$

where n_j is the number of steps in the sequence $\langle \Phi, H, e \rangle \xrightarrow{j-1} \langle \Phi_j, H_j, e_j \rangle$ not invoked by (15).

Thus, by transitivity of $\xrightarrow{*}$, it suffices to show

$$\langle Z_H(H_j), Z(e_j) \rangle \xrightarrow{k} \langle Z_H(H'), Z(e') \rangle$$

where $k = 0$ if $\langle \Phi_j, H_j, e_j \rangle \dashrightarrow \langle \Phi', H', e' \rangle$ is invoked by (15), and $k = 1$ otherwise.

We will prove the above by induction over the derivation of $\langle \Phi_j, H_j, e_j \rangle \dashrightarrow \langle \Phi', H', e' \rangle$. Most cases of this proof by induction are trivial because most reduction rules in \dashrightarrow come from the original \rightarrow . Thus, we prove the three non-trivial cases where the reduction rule is not derived from \rightarrow and then show three of the trivial cases which comes from \rightarrow .

1. Consider the reduction rule

$$\langle \Phi \uplus \{f\}, H, \text{protect}(e, f) \rangle \dashrightarrow \langle \Phi, H, e \rangle$$

Then, we must show

$$\langle Z_H(H), Z(\text{protect}(e, f)) \rangle \xrightarrow{0} \langle Z_H(H), Z(e) \rangle$$

However, notice that $Z(\text{protect}(e, f)) = Z(e)$. Then, we trivially have

$$\langle Z_H(H), Z(e) \rangle \xrightarrow{0} \langle Z_H(H), Z(e) \rangle$$

which finishes the proof for this case.

2. Consider the reduction rule

$$\frac{f \text{ fresh}}{\langle \Phi, H, \text{let } a_\bullet = v \text{ in } e \rangle \dashrightarrow \langle \Phi \uplus \{f\}, H, [a_\bullet \mapsto \text{protect}(v, f)]e \rangle}$$

Then, we must show

$$\langle Z_H(H), Z(\text{let } a_\bullet = v \text{ in } e) \rangle \xrightarrow{1} \langle Z_H(H), Z([a_\bullet \mapsto \text{protect}(v, f)]e) \rangle$$

Factor the Z function through the expressions:

$$\langle Z_H(H), \text{let } a_\bullet = Z(v) \text{ in } Z(e) \rangle \xrightarrow{1} \langle Z_H(H), [a_\bullet \mapsto Z(v)]Z(e) \rangle$$

Since $Z(v)$ is still a target value, the above reduction follows from the normal reduction rule on `let`.

3. Consider the reduction rule

$$\frac{f \text{ fresh}}{\langle \Phi, H, \lambda a_\bullet. e v \rangle \dashrightarrow \langle \Phi \uplus \{f\}, H, [a_\bullet \mapsto \text{protect}(v, f)]e \rangle}$$

Then, we must show

$$\langle Z_H(H), Z(\lambda a_\bullet. e v) \rangle \xrightarrow{1} \langle Z_H(H), Z([a_\bullet \mapsto \text{protect}(v, f)]e) \rangle$$

Factor the Z function through the expressions:

$$\langle Z_H(\mathsf{H}), \lambda a_\bullet. Z(e) Z(v) \rangle \xrightarrow{1} \langle Z_H(\mathsf{H}), [a_\bullet \mapsto Z(v)]Z(e) \rangle$$

Since $Z(v)$ is still a target value, the above reduction follows from the normal reduction rule on λ .

4. Consider the reduction rule

$$\frac{\text{fresh } \ell}{\langle \Phi, \mathsf{H}, \text{ref } v \rangle \dashrightarrow \langle \Phi, \mathsf{H}[\ell \mapsto v], \ell \rangle}$$

Thus, we must show

$$\langle Z_H(\mathsf{H}), Z(\text{ref } v) \rangle \xrightarrow{1} \langle Z_H(\mathsf{H}[\ell \mapsto v]), Z(\ell) \rangle$$

Factor through Z_H and Z :

$$\langle Z_H(\mathsf{H}), \text{ref } Z(v) \rangle \xrightarrow{1} \langle Z_H(\mathsf{H})[\ell \mapsto Z(v)], \ell \rangle$$

Since $Z(v)$ is a target value, the above reduction follows directly from the normal ref reduction rule.

5. Consider the reduction rule

$$\frac{\mathsf{H}[\ell] = v}{\langle \Phi, \mathsf{H}, !\ell \rangle \dashrightarrow \langle \Phi, \mathsf{H}, v \rangle}$$

Thus, we must show

$$\langle Z_H(\mathsf{H}), Z(!\ell) \rangle \xrightarrow{1} \langle Z_H(\mathsf{H}), Z(v) \rangle$$

Factor through Z on the left side:

$$\langle Z_H(\mathsf{H}), !\ell \rangle \xrightarrow{1} \langle Z_H(\mathsf{H}), Z(v) \rangle$$

By the definition of Z_H , if $\mathsf{H}[\ell] = v$, then $Z_H(\mathsf{H})[\ell] = Z(v)$. Thus, the above follows directly from the normal $!$ reduction rule.

6. Consider the reduction rule

$$\frac{\langle \Phi, \mathsf{H}, e \rangle \dashrightarrow \langle \Phi, \mathsf{H}', e' \rangle}{\langle \Phi, \mathsf{H}, K[e] \rangle \dashrightarrow \langle \Phi, \mathsf{H}', K[e'] \rangle}$$

By the induction hypothesis, we have $\langle Z_H(\mathsf{H}), Z(\mathbf{e}) \rangle \xrightarrow{k} \langle Z_H(\mathsf{H}'), Z(\mathbf{e}') \rangle$, where $k = 0$ if $\langle \Phi, \mathsf{H}, \mathbf{e} \rangle \dashrightarrow \langle \Phi, \mathsf{H}', \mathbf{e}' \rangle$ was invoked by (15) and $k = 1$ otherwise. Then, we must show

$$\langle Z_H(\mathsf{H}), Z(K[\mathbf{e}]) \rangle \xrightarrow{k} \langle Z_H(\mathsf{H}'), Z(K[\mathbf{e}']) \rangle$$

Factor Z through K :

$$\langle Z_H(\mathsf{H}), Z(K[Z(\mathbf{e})]) \rangle \xrightarrow{k} \langle Z_H(\mathsf{H}'), Z(K[Z(\mathbf{e}')]) \rangle$$

If $k = 0$, then by $\langle Z_H(\mathsf{H}), Z(\mathbf{e}) \rangle \xrightarrow{k} \langle Z_H(\mathsf{H}'), Z(\mathbf{e}') \rangle$, we must have $Z_H(\mathsf{H}) = Z_H(\mathsf{H}')$ and $Z(\mathbf{e}) = Z(\mathbf{e}')$, in which case the above is trivial. Otherwise, if $k = 1$, the above follows directly from the evaluation context reduction rule in the target.

□

Lemma B.2.20 (Phantom Steps Bounded). *If*

$$\langle \mathsf{H}, \mathbf{e}^+ \rangle \xrightarrow{n} \langle \mathsf{H}', \mathbf{e}' \rangle \not\rightarrow$$

then for any set of static flags Φ_{r1} , there exists some set of static flags Φ'_1 , $m \leq 2n$, and expression \mathbf{e}'_1 such that

$$\langle \Phi_{r1}, \mathsf{H}, \mathbf{e}^+ \rangle \xrightarrow{m} \langle \Phi'_1, \mathsf{H}'_1, \mathbf{e}'_1 \rangle \not\rightarrow$$

where, if \mathbf{e}'_1 is a value, then $\mathsf{H}' = Z(\mathsf{H}'_1)$ and $\mathbf{e}' = Z(\mathbf{e}'_1)$

where, as defined in the previous Lemma, let $Z(\mathbf{e})$ be an expression in the original LCVM language where every subexpression of the form $\text{protect}(\mathbf{e}', f)$ is replaced with \mathbf{e}'

and, for any heap H , let $Z_H(\mathsf{H}) = \{\ell \mapsto Z(v) \mid \ell \mapsto v \in \mathsf{H}\}$.

Note that we write \mathbf{e}^+ to indicate that we are proving this with respect to compiled terms. The only constraint we actually need is that H and \mathbf{e}^+ is a valid heap and expression, respectively, in the original LCVM language and thus does not include any subexpressions of the form $\text{protect}(\cdot)$, as it is not intended to be written by programmers (or compilers), but rather arise through reduction in the phantom operational semantics.

Proof. Suppose that $\langle \Phi_{r1}, \mathsf{H}, \mathbf{e}^+ \rangle \xrightarrow{m} \langle \Phi'_1, \mathsf{H}'_1, \mathbf{e}'_1 \rangle$ for some m . Then, by Lemma B.2.19,

$$\langle Z_H(\mathsf{H}), Z(\mathbf{e}^+) \rangle = \langle \mathsf{H}, \mathbf{e}^+ \rangle \xrightarrow{n'} \langle Z_H(\mathsf{H}'_1), Z(\mathbf{e}'_1) \rangle$$

where n' is the number of steps in the original reduction sequence not invoked by $\text{protect}(\cdot)$. Since $\langle \mathsf{H}, \mathbf{e}^+ \rangle$ terminates in n steps by assumption, $n' \leq n$.

Consider that, since $\text{protect}(\cdot)$ does not occur in H or e^+ , $\text{protect}(\cdot)$ instructions are only introduced by let and λ , and they are substituted for variable occurrences. Further, note that, for the reduction to have succeeded in the phantom semantics, out of each set of variable uses (that share a flag), only one $\text{protect}(\cdot)$ term could have been evaluated. This means that each reduction of $\text{protect}(\cdot)$ corresponds to a reduction of the let or λ that introduced it, so the number of reductions of $\text{protect}(\cdot)$ is at most the number of reductions not of $\text{protect}(\cdot)$, which means $m - n' \leq n'$. Ergo, $m \leq 2n' \leq 2n$.

This suffices to show that $\langle \Phi_{r1}, H, e^+ \rangle$ can not take more than $2n$ steps, so there is some $m \leq 2n$ such that $\langle \Phi_{r1}, H, e^+ \rangle \xrightarrow{m} \langle \Phi'_1, H'_1, e'_1 \rangle \not\rightarrow$.

To finish the proof, suppose that e'_1 is a value. Then, as shown above, $\langle H, e^+ \rangle \xrightarrow{n'} \langle Z_H(H'_1), Z(e'_1) \rangle$. If e'_1 is a value, then $Z(e'_1)$ is also a value, so $\langle Z_H(H'_1), Z(e'_1) \rangle$ is irreducible. Ergo, since $\langle H, e^+ \rangle \xrightarrow{n} \langle H', e' \rangle \not\rightarrow$ and $\langle H, e^+ \rangle$ can only possibly step to one irreducible configuration, $H' = Z_H(H'_1)$ and $e' = Z(e'_1)$. \square

Lemma B.2.21 (Compat \rightarrow).

$$\Gamma; \Omega; \Delta; \Gamma[x : \tau_1] \vdash e \preceq e : \tau_2 \implies \Gamma; \Omega; \Delta; \Gamma \vdash \lambda x : \tau_1. e \preceq \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2$$

Proof. Expanding the conclusion, given

$$\begin{aligned} & \forall W. \forall \rho. \gamma_\Gamma \gamma_\Gamma \gamma_\Omega. \\ & \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \emptyset, \emptyset, \gamma_\Omega) \in \mathcal{G}[\Omega]. \wedge (W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega]. \end{aligned}$$

we must show

$$(W, (\emptyset, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(\lambda x : \tau_1. e^+)))), (\emptyset, \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(\lambda x : \tau_1. e^+))))) \in \mathcal{E}[\tau_1 \rightarrow \tau_2].$$

Notice that both of the expressions have no free variables by Lemma B.2.15.

We can push the compiler and substitutions through the lambda to refine that to:

$$(W, (\emptyset, \lambda x. \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e^+)))), (\emptyset, \lambda x. \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e^+))))) \in \mathcal{E}[\tau_1 \rightarrow \tau_2]_\rho$$

Then, by Lemma B.2.6, there exists a γ' such that $(W, \emptyset, \emptyset, \gamma') \in \mathcal{G}[\Omega]$. and closing over e^+ with γ' is the same as closing with γ_Ω . Thus, we refine the above to:

$$(W, (\emptyset, \lambda x. \gamma_\Gamma^1(\gamma_\Omega^1(\gamma'^1(e^+)))), (\emptyset, \lambda x. \gamma_\Gamma^2(\gamma_\Omega^2(\gamma'^2(e^+))))) \in \mathcal{E}[\tau_1 \rightarrow \tau_2]_\rho$$

Since $\mathcal{V}[\tau_1 \rightarrow \tau_2]_\rho \subseteq \mathcal{E}[\tau_1 \rightarrow \tau_2]_\rho$ by Lemma B.2.1, it suffices to show that:

$$(W, (\emptyset, \lambda x. \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma'^1(e^+)))), (\emptyset, \lambda x. \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma'^2(e^+))))) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]_\rho$$

Expanding the value relation, given

$$\forall v_1 v_2. W \sqsubset_{\emptyset, \emptyset} W' \wedge (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau_1]_\rho$$

we must prove:

$$(W', (\emptyset, [x \mapsto v_1] \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma'^1(e^+)))), (\emptyset, [x \mapsto v_2] \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma'^2(e^+))))) \in \mathcal{E}[\tau_2]_\rho$$

By $W \sqsubseteq_{\emptyset, \emptyset} W'$ and Lemma B.2.8, we have

$$(W', \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_\rho$$

$$(W', \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega]_\rho$$

$$(W', \emptyset, \emptyset, \gamma') \in \mathcal{G}[\Omega_{\circ}]_\rho$$

Notice that

$$(W', \emptyset, \emptyset, \gamma_{\Gamma}[x \rightarrow (v_1, v_2)]) \in \mathcal{G}[\Gamma[x : \tau_1]]_\rho$$

because $(W', \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_\rho$ and $(W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau_1]_\rho$. Then, we can instantiate Lemma B.2.7 with the first induction hypothesis and $W', \gamma_{\Gamma}[x \rightarrow (v_1, v_2)], \gamma_{\Omega}, \gamma', \rho$. Therefore,

$$(W', (\emptyset, \gamma_{\Gamma}[x \rightarrow (v_1, v_2)]^1(\gamma_{\Omega}^1(\gamma'^1(e^+)))), (\emptyset, \gamma_{\Gamma}[x \rightarrow (v_1, v_2)]^2(\gamma_{\Omega}^2(\gamma'^2(e^+))))) \in \mathcal{E}[\tau_2]_\rho$$

We can simplify the above statement by bringing $x \rightarrow v_1$ out on the left side and bringing $x \rightarrow v_2$ out on the right side. This suffices to finish the proof. \square

Lemma B.2.22 (Compat app).

$$\begin{aligned} \Gamma; \Omega_1; \Delta; \Gamma \vdash e_1 \preceq e_1 : \tau_1 \rightarrow \tau_2 \wedge \Gamma; \Omega_2; \Delta; \Gamma \vdash e_2 \preceq e_2 : \tau_1 &\implies \\ \Gamma; \Omega_1 \uplus \Omega_2; \Delta; \Gamma \vdash e_1 e_2 \preceq e_1 e_2 : \tau_2 \end{aligned}$$

Proof. Expanding the conclusion, given

$$\begin{aligned} &\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Omega}. \\ &\rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \\ &\wedge (W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\Omega_1 \uplus \Omega_2]. \end{aligned}$$

we must show

$$(W, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1 e_2^+)))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1 e_2^+))))) \in \mathcal{E}[\tau_2].$$

Notice that both of the expressions have no free variables by Lemma B.2.15.

We can push the compiler and substitutions through the application to refine that to:

$$(W, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+)))) \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_2^+)))), \\ (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1^+)))) \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+)))) \in \mathcal{E}[\tau_2]_{\rho}$$

Next, by Lemma B.2.5, we have that $\gamma_{\Omega} = \gamma_1 \uplus \gamma_2$, $\Phi_1 = \Phi_{1l} \uplus \Phi_{1r}$, and $\Phi_2 = \Phi_{2l} \uplus \Phi_{2r}$ where

$$(W, \Phi_{1l}, \Phi_{2l}, \gamma_1) \in \mathcal{G}[\Omega_1].$$

and

$$(W, \Phi_{1r}, \Phi_{2r}, \gamma_2) \in \mathcal{G}[\Omega_2].$$

and for all $i \in \{1, 2\}$,

$$\gamma_{\Omega}^i(e_1^+) = \gamma_1^i(e_1^+)$$

and

$$\gamma_{\Omega}^i(e_2^+) = \gamma_1^i(e_2^+)$$

Thus, we refine the statement we need to prove to:

$$(W, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_1^1(e_1^+)))) \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_2^+)))), \\ (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_1^2(e_1^+)))) \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_2^2(e_2^+)))) \in \mathcal{E}[\tau_2]_{\rho}$$

Let e_1 and e_2 be the first and second expressions, respectively, in the above tuple. Expanding the definition of the expression relation, given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \emptyset \wedge \Phi_{r2} \# \emptyset \wedge \Phi_{r1} \uplus \emptyset, \Phi_{r2} \uplus \emptyset : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \emptyset, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_1, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \emptyset, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau]_{\rho} \end{aligned}$$

Next, we need to know what e'_1 is. From the operational semantic, the application will run the first subexpression using the heap H_1 until it reaches

a target value or gets stuck. By appealing to our first induction hypothesis, instantiated with $W, \gamma_{\Gamma}, \gamma_{\Gamma}, \gamma_1, \rho$, we get that:

$$(W, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_1^1(e_1^+)))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_1^2(e_1^+))))) \in \mathcal{E}[\tau_1]_{\rho}$$

Therefore, the configuration

$$\langle \Phi_{r1} \uplus \text{flags}(W, 1), H_1, \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_1^1(e_1^+))) \rangle$$

either steps to fail CONV, in which case the whole application expression steps to fail CONV, or steps to some irreducible configuration $\langle \Phi_{r1} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1l} \uplus \Phi_{g1l}, H_1^*, e_1^* \rangle$, in which case the configuration

$$\langle \Phi_{r2} \uplus \text{flags}(W, 2), H_2, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_1^2(e_1^+))) \rangle$$

steps to some irreducible configuration $\langle \Phi_{r2} \uplus \text{flags}(W_1, 2) \uplus \Phi_{f2l} \uplus \Phi_{g2l}, H_2^*, e_1^{\dagger} \rangle$ and there exists some world W_1 such that $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1, H_1^*, H_2^* : W_1$, and $(W_1, (\Phi_{f1l}, e_1^*), (\Phi_{f2l}, e_1^{\dagger}))$. By Lemma B.2.17, $\Phi_{f1l} = \Phi_{f2l} = \emptyset$.

Since terms in the value relation are target values, the original application will continue reducing on the second subexpression according to the operational semantics. Then, we can appeal to the second induction hypothesis instantiated with $W_1, \gamma_{\Gamma}, \gamma_{\Gamma}, \gamma_2, \rho$, because $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$ and $\mathcal{G}[\Gamma]_{\rho}, \mathcal{G}[\Gamma]_{\cdot}, \mathcal{G}[\Omega_1 \uplus \Omega_2]$ are closed under world extension by Lemma B.2.8. Thus,

$$(W_1, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_2^+)))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_2^2(e_2^+))))) \in \mathcal{E}[\tau_2]_{\rho}$$

Therefore, the configuration:

$$\langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \text{flags}(W_1, 1), H_1^*, \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_2^+))) \rangle$$

either reduces to fail CONV, in which case the whole expression steps to fail CONV, or to some irreducible configuration $\langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \text{flags}(W_2, 1) \uplus \Phi_{f1r} \uplus \Phi_{g1r}, H_1^{**}, e_2^* \rangle$, in which case on the other side, the configuration

$$\langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \text{flags}(W_1, 2), H_2^*, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_2^2(e_2^+))) \rangle$$

reduces to some irreducible configuration $\langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \text{flags}(W_2, 2) \uplus \Phi_{f2r} \uplus \Phi_{g2r}, H_2^{**}, e_2^{\dagger} \rangle$, and there exists some W_2 such that $W_1 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{g1l}, \Phi_{r2} \uplus \Phi_{g2l}} W_2, H_1^{**}, H_2^{**} : W_2$, and $(W_2, (\Phi_{f1r}, e_2^*), (\Phi_{f2r}, e_2^{\dagger})) \in \mathcal{V}[\tau_1]_{\rho}$. By Lemma B.2.17, $\Phi_{f1r} = \Phi_{f2r} = \emptyset$.

Then, instantiate $(W_1, (\emptyset, e_1^*), (\emptyset, e_1^{\dagger})) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]_{\rho}$ with $e_2^*, e_2^{\dagger}, \triangleright W_2$. Because $W_1 \sqsubseteq_{\emptyset, \emptyset} W_2$ and $W_2 \sqsubset_{\emptyset, \emptyset} \triangleright W_2$, it follows that $W_1 \sqsubset_{\emptyset, \emptyset} \triangleright W_2$.

Moreover, $(\triangleright W_2, (\emptyset, e_2^*), (\emptyset, e_2^\dagger)) \in \mathcal{V}[\llbracket \textcolor{blue}{T}_1 \rrbracket]_\rho$ (because $(W_2, (\emptyset, e_2^*), (\emptyset, e_2^\dagger)) \in \mathcal{V}[\llbracket \textcolor{blue}{T}_1 \rrbracket]_\rho$ and $W_2 \sqsubseteq_{\emptyset, \emptyset} \triangleright W_2$), so we find that there exist e_b^*, e_b^\dagger such that

$$e_1^* = \lambda x. e_b^*$$

and

$$e_1^\dagger = \lambda x. e_b^\dagger$$

and

$$(\triangleright W_2, (\emptyset, [x \mapsto e_2^*] e_b^*), (\emptyset, [x \rightarrow e_2^\dagger] e_b^\dagger)) \in \mathcal{E}[\llbracket \textcolor{blue}{T}_2 \rrbracket]_\rho$$

Now, by the operational semantic, the original configuration with heap H_1 steps to

$$\begin{aligned} & \langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \Phi_{g1r} \uplus \text{flags}(W_2, 1), H_1^{**}, \lambda x. e_b^* e_2^* \rangle \dashrightarrow \\ & \langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \Phi_{g1r} \uplus \text{flags}(W_2, 1), H_1^{**}, [x \mapsto e_2^*] e_b^* \rangle \end{aligned}$$

and, on the other side, the original configuration with H_2 steps to

$$\begin{aligned} & \langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \Phi_{g2r} \uplus \text{flags}(W_2, 2), H_2^{**}, \lambda x. e_b^\dagger e_2^\dagger \rangle \dashrightarrow \\ & \langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \Phi_{g2r} \uplus \text{flags}(W_2, 2), H_2^{**}, [x \mapsto e_2^\dagger] e_b^\dagger \rangle \end{aligned}$$

Then, since $H_1^{**}, H_2^{**} : W_2$, by Lemma B.2.11, it follows that $H_1^{**}, H_2^{**} : \triangleright W_2$. We also have $\text{flags}(W_2, 1) = \text{flags}(\triangleright W_2, 1)$ and $\text{flags}(W_2, 2) = \text{flags}(\triangleright W_2, 2)$, since \triangleright does not change the dynamic flags in the world. Thus, we can instantiate the above fact to deduce that either the first configuration steps to fail CONV, in which case the original configuration with H_1 steps to fail CONV, or the first configuration steps to some irreducible configuration

$$\langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \Phi_{g1r} \uplus \text{flags}(W_3, 1) \uplus \Phi_{f1f} \uplus \Phi_{g1f}, H_1^f, e_f^* \rangle$$

in which case the second configuration steps to some irreducible configuration

$$\langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \Phi_{g2r} \uplus \text{flags}(W_3, 2) \uplus \Phi_{f2f} \uplus \Phi_{g2f}, H_2^f, e_f^\dagger \rangle$$

and there exists some W_3 such that $\triangleright W_2 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{g1l} \uplus \Phi_{g1r}, \Phi_{r2} \uplus \Phi_{g2l} \uplus \Phi_{g2r}} W_3$, $H_1^f, H_2^f : W_3$, and

$$(W_3, (\Phi_{f1f}, e_f^*), (\Phi_{f2f}, e_f^\dagger)) \in \mathcal{V}[\llbracket \textcolor{blue}{T}_2 \rrbracket]_\rho$$

Then, since $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$, $W_1 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, $W_2 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} \triangleright W_2$, $\triangleright W_2 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_3$, we have $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_3$, which suffices to finish the proof. \square

Lemma B.2.23 (Compat $\textcolor{blue}{V}$).

$$\Gamma; \Omega; \Delta, \alpha; \Gamma \vdash e \preceq e : \tau \implies \Gamma; \Omega; \Delta; \Gamma \vdash \Lambda \alpha. e \preceq \Lambda \alpha. e : \forall \alpha. \tau$$

Proof. Expanding the conclusion, given

$$\begin{aligned} & \forall W. \forall \rho \gamma_{\Gamma} \gamma_{\Omega}. \\ & \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \end{aligned}$$

we must show

$$(W, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\Lambda\alpha.e^+)))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\Lambda\alpha.e^+))))) \in \mathcal{E}[\forall\alpha.\tau].$$

Notice that both of the expressions have no free variables by Lemma B.2.15.

We can push the compiler and substitutions through the pair to refine that to:

$$(W, (\emptyset, \lambda_{-}.\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+)))), (\emptyset, \lambda_{-}.\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+))))) \in \mathcal{E}[\forall\alpha.\tau]_{\rho}$$

Then, by Lemma B.2.6, there exists a γ' such that $(W, \emptyset, \emptyset, \gamma') \in \mathcal{G}[\Omega_o]$. and closing over e^+ with γ' is the same as closing with γ_{Ω} . Thus, we refine the above to:

$$(W, (\emptyset, \lambda_{-}.\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma'^1(e^+)))), (\emptyset, \lambda_{-}.\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma'^2(e^+))))) \in \mathcal{E}[\forall\alpha.\tau]_{\rho}$$

Then, since $\mathcal{V}[\forall\alpha.\tau]_{\rho} \subseteq \mathcal{E}[\forall\alpha.\tau]_{\rho}$, it suffices to prove:

$$(W, (\emptyset, \lambda_{-}.\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma'^1(e^+)))), (\emptyset, \lambda_{-}.\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma'^2(e^+))))) \in \mathcal{V}[\forall\alpha.\tau]_{\rho}$$

Consider some arbitrary $R \in UnrTyp$ and W' such that $W \sqsubset_{\emptyset, \emptyset} W'$. We must prove that

$$(W', (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma'^1(e^+)))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma'^2(e^+))))) \in \mathcal{E}[\tau]_{\rho[\alpha \mapsto R]}$$

Since $R \in UnrTyp$ and $\rho \in \mathcal{D}[\Delta]$, it follows that $\rho[\alpha \mapsto R] \in \mathcal{D}[\Delta, \alpha]$. Thus, we can instantiate Lemma B.2.7 with the first induction hypothesis and $W', \gamma_{\Gamma}, \gamma_{\Omega}, \gamma', \rho[\alpha \mapsto R]$, because $W \sqsubseteq_{\emptyset, \emptyset} W'$ and thus by Lemma B.2.8, the substitutions are still in the interpretation of $\mathcal{G}[\Gamma]_{\rho}, \mathcal{G}[\Omega], \mathcal{G}[\Omega_1 \uplus \Omega_2]$, respectively, with the world W' . This suffices to prove the above fact. \square

Lemma B.2.24 (Compat $[\tau/\alpha]$).

$$\Delta \vdash \tau' \wedge \Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \forall\alpha.\tau \implies \Gamma; \Omega; \Delta; \Gamma \vdash e[\tau'] \preceq e[\tau'] : \tau[\tau'/\alpha]$$

Proof. Expanding the conclusion, given

$$\begin{aligned} & \forall W. \forall \rho \gamma_{\Gamma} \gamma_{\Omega}. \\ & \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \\ & \wedge (W, \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \wedge (W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \end{aligned}$$

we must show

$$(W, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e[\tau']^+)))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e[\tau']^+))))) \in \mathcal{E}[\![\tau'/\alpha]\!].$$

Notice that both of the expressions have no free variables by Lemma B.2.15.

We can push the compiler and substitutions through the type application to refine this to:

$$(W, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+)))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))) \in \mathcal{E}[\![\tau'/\alpha]\!]_{\rho}$$

Let e_1 and e_2 be the first and second expressions, respectively, in the above tuple. Expanding the definition of the expression relation, given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \emptyset \wedge \Phi_{r2} \# \emptyset \wedge \Phi_{r1} \uplus \emptyset, \Phi_{r2} \uplus \emptyset : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \emptyset, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_{r1}, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \emptyset, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\![\tau]\!]_{\rho} \end{aligned}$$

To proceed, we must find what e'_1 is. From the operational semantic, we know the application will run its subexpression using H_1 until it reaches a target value or gets stuck. From the induction hypothesis instantiated with $W, \gamma_{\Gamma}, \gamma_{\Omega}, \rho$, we find that:

$$(W, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+)))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+))))) \in \mathcal{E}[\![\forall \alpha. \tau]\!]_{\rho}$$

Thus, the configuration

$$\langle \Phi_{r1} \uplus \text{flags}(W, 1), H_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+))) \rangle$$

either reduces to fail CONV, in which case the entire term reduced to fail CONV, or it will reduce to some $\langle \Phi_{r1} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1l} \uplus \Phi_{g1l}, H_1^*, e_1^* \rangle$, in which case the configuration

$$\langle \Phi_{r2} \uplus \text{flags}(W, 2), H_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+))) \rangle$$

will reduce to some $\langle \Phi_{r2} \uplus \text{flags}(W_1, 2) \uplus \Phi_{f2l} \uplus \Phi_{g2l}, H_2^*, e_1^\dagger \rangle$ and there exists some world W_1 where $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$, $H_1^*, H_2^* : W_1$, and

$$(W_1, (\Phi_{f1l}, e_1^*), (\Phi_{f2l}, e_1^\dagger)) \in \mathcal{V}[\![\forall \alpha. \tau]\!]_\rho$$

By expanding the value relation, we find $\Phi_{f1l} = \Phi_{f2l} = \emptyset$.

Then, we can instantiate the above fact with $\mathcal{V}[\![\tau']]\!_\rho$ and $\triangleright W_1$. (Note that $\mathcal{V}[\![\tau']]\!_\rho \in UnrTyp$ by Lemma B.2.12.) Since $W \sqsubset \triangleright W_1$ (as $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$ and $W_1 \sqsubset_{\Phi_{r1}, \Phi_{r2}} \triangleright W_1$ since W_1 and $\triangleright W_1$ have the same dynamic flags), we find that there exist e_b^* , e_b^\dagger such that

$$e_1^* = \lambda_- e_b^*$$

$$e_1^\dagger = \lambda_- e_b^\dagger$$

and

$$(\triangleright W_1, (\emptyset, e_b^*), (\emptyset, e_b^\dagger)) \in \mathcal{E}[\![\tau]\!]_{\rho[\alpha \rightarrow \mathcal{V}[\![\tau']]\!_\rho]}$$

Notice that $\text{flags}(W_1, 1) = \text{flags}(\triangleright W_1, 1)$ and $\text{flags}(W_1, 2) = \text{flags}(\triangleright W_1, 2)$ because \triangleright does not change the dynamic flags in the world.

Ergo, by the operational semantic, the original configuration with heap H_1 steps to

$$\begin{aligned} & \langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \text{flags}(\triangleright W_1, 1), H_1^*, \lambda_- e_b^* () \rangle \dashrightarrow \\ & \langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \text{flags}(\triangleright W_1, 1), H_1^*, e_b^* \rangle \end{aligned}$$

and, on the other side, the configuration with H_2 steps to

$$\begin{aligned} & \langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \text{flags}(\triangleright W_1, 2), H_2^*, \lambda_- e_b^\dagger () \rangle \dashrightarrow \\ & \langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \text{flags}(\triangleright W_1, 2), H_2^*, e_b^\dagger \rangle \end{aligned}$$

Next, since $H_1^*, H_2^* : W_1$, by Lemma B.2.11, it follows that $H_1^*, H_2^* : \triangleright W_1$, so we can instantiate the above fact with H_1^*, H_2^* to deduce that either the first configuration steps to fail CONV, in which case the original configuration with H_1 steps to fail CONV, or the first configuration steps to some irreducible configuration $\langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \text{flags}(W_2, 1) \uplus \Phi_{f1f} \uplus \Phi_{g1f}, H_1^{**}, e_f^* \rangle$, in which case the second configuration also steps to some irreducible configuration $\langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \text{flags}(W_2, 2) \uplus \Phi_{f2f} \uplus \Phi_{g2f}, H_2^{**}, e_f^\dagger \rangle$, and there exists some W_2 where $\triangleright W_1 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{g1l}, \Phi_{r2} \uplus \Phi_{g2l}} W_2$, $H_1^{**}, H_2^{**} : W_2$, and $(W_2, (\Phi_{f1f}, e_f^*), (\Phi_{f2f}, e_f^\dagger)) \in \mathcal{V}[\![\tau]\!]_{\rho[\alpha \rightarrow \mathcal{V}[\![\tau']]\!_\rho]}$. Therefore, by Lemma B.2.13, $(W_2, (\Phi_{f1f}, e_f^*), (\Phi_{f2f}, e_f^\dagger)) \in \mathcal{V}[\![\tau[\tau'/\alpha]]]\!_\rho$. Finally, since $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$, $W_1 \sqsubset_{\Phi_{r1}, \Phi_{r2}} \triangleright W_1$, and $\triangleright W_1 \sqsubset_{\Phi_{r1}, \Phi_{r2}} W_2$, we have $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, which suffices to finish the proof. \square

Lemma B.2.25 (Compat $(\text{e})_\tau$).

$$\begin{aligned} \Delta; \Gamma; \Omega \vdash e \preceq e : \tau \wedge \text{no}_\bullet(\Omega) \wedge _ : \tau \sim \tau \\ \implies \Gamma; \Omega; \Delta \vdash (\text{e})_\tau \preceq (\text{e})_\tau : \tau \wedge _ : \tau \sim \tau \end{aligned}$$

Proof. We must show that given

$$\begin{aligned} \forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Omega. \\ \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \emptyset, \emptyset, \gamma_\Omega) \in \mathcal{G}[\Omega]. \wedge (W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega]. \end{aligned}$$

we must show

$$(W, (\emptyset, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1((\text{e})_\tau^+)))), (\emptyset, \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2((\text{e})_\tau^+))))) \in \mathcal{E}[\tau]_\rho$$

We can push the compiler and substitutions through the pair to refine that to:

$$(W, (\emptyset, C_{\tau \mapsto \tau}(\gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e^+))))), (\emptyset, C_{\tau \mapsto \tau}(\gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e^+)))))) \in \mathcal{E}[\tau]_\rho$$

Now, by instantiating our induction hypothesis with $W, \gamma_\Gamma, \gamma_\Omega, \gamma_\Omega, \rho$, we find that:

$$(W, (\Phi_1, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e^+)))), (\Phi_2, \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e^+))))) \in \mathcal{E}[\tau].$$

However, since $\text{no}_\bullet(\Omega)$, there are no static affine variables in Ω , because $\Omega \subseteq \Omega$. Ergo, since $(W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega]$, it must be the case that $\Phi_1 = \Phi_2 = \emptyset$.

Therefore, by Theorem B.2.18, we have

$$(W, (\emptyset, C_{\tau \mapsto \tau}(\gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e^+))))), (\emptyset, C_{\tau \mapsto \tau}(\gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e^+)))))) \in \mathcal{E}[\tau].$$

Finally, by Lemma B.2.14, we have

$$(W, (\emptyset, C_{\tau \mapsto \tau}(\gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e^+)))), (\emptyset, C_{\tau \mapsto \tau}(\gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e^+)))))) \in \mathcal{E}[\tau]_\rho$$

as was to be proven. \square

Lemma B.2.26 (Compat **unit**).

$$\Delta; \Gamma; \Omega \vdash () \preceq () : \text{unit}$$

Proof. Expanding the conclusion, given

$$\begin{aligned} \forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Omega. \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \\ \wedge (W, \emptyset, \emptyset, \gamma_\Omega) \in \mathcal{G}[\Omega]. \wedge (W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega]. \end{aligned}$$

we must show

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1((\lambda)^+)))), (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2((\lambda)^+))))) \in \mathcal{E}[\text{unit}].$$

$(\lambda)^+ = \lambda$ is a closed term, so the closings have no effect. Ergo, we must show:

$$(W, (\Phi_1, \lambda), (\Phi_2, \lambda)) \in \mathcal{E}[\tau].$$

This trivially follows from $(W, (\emptyset, \lambda), (\emptyset, \lambda)) \in \mathcal{V}[\text{unit}]_{\rho}$ and Lemma B.2.2. \square

Lemma B.2.27 (Compat **true**).

$$\Delta; \Gamma; \Gamma; \Omega \vdash \text{true} \preceq \text{true} : \text{bool}$$

Proof. Expanding the conclusion, given

$$\begin{aligned} \forall W. \forall \rho \gamma_{\Gamma} \gamma_{\Omega}. \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega] \\ \wedge (W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \end{aligned}$$

we must show

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\text{true}^+)))), (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\text{true}^+))))) \in \mathcal{E}[\text{bool}].$$

$\text{true}^+ = 0$ is a closed term, so the closings have no effect. Ergo, we must show:

$$(W, (\Phi_1, 0), (\Phi_2, 0)) \in \mathcal{E}[\text{bool}].$$

This trivially follows from $(W, (\emptyset, 0), (\emptyset, 0)) \in \mathcal{V}[\text{bool}]_{\rho}$ and Lemma B.2.2. \square

Lemma B.2.28 (Compat **false**).

$$\Delta; \Gamma; \Gamma; \Omega \vdash \text{false} \preceq \text{false} : \text{bool}$$

Proof. This case is trivially similar to **true**, since $\text{false}^+ = 1$ and $(W, (\emptyset, 1), (\emptyset, 1)) \in \mathcal{V}[\text{bool}]_{\rho}$. \square

Lemma B.2.29 (Compat **int**).

$$\Delta; \Gamma; \Gamma; \Omega \vdash n \preceq n : \text{int}$$

Proof. Expanding the conclusion, given

$$\begin{aligned} \forall W. \forall \rho \gamma_{\Gamma} \gamma_{\Omega}. \\ \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega] \wedge (W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \end{aligned}$$

we must show

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\mathbf{n}^+)))), (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\mathbf{n}^+))))) \in \mathcal{E}[\![\text{int}]\!].$$

$\mathbf{n}^+ = \mathbf{n}$ is a closed term, so the closings have no effect. Ergo, we must show:

$$(W, (\Phi_1, \mathbf{n}), (\Phi_2, \mathbf{n})) \in \mathcal{E}[\![\tau]\!].$$

This trivially follows from $(W, (\emptyset, \mathbf{n}), (\emptyset, \mathbf{n})) \in \mathcal{V}[\![\text{int}]\!]_{\rho}$ and Lemma B.2.2.

□

Lemma B.2.30 (Compat \mathbf{x}).

$$\mathbf{x} : \tau \in \Gamma \implies \Delta; \Gamma; \Gamma \vdash \mathbf{x} \preceq \mathbf{x} : \tau$$

Proof. Expanding the conclusion, given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Omega}.$$

$$\rho \in \mathcal{D}[\![\Delta]\!] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]_{\rho} \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!] \wedge (W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\![\Omega]\!].$$

we must show

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\mathbf{x}^+)))), (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\mathbf{x}^+))))) \in \mathcal{E}[\![\tau]\!].$$

Notice that $\mathbf{x}^+ = \mathbf{x}$. Then, since $\mathbf{x} : \tau \in \Gamma$ and $(W, \emptyset, \emptyset, \gamma_{\Gamma})$, we have

$$\gamma_{\Gamma}(\mathbf{x}) = (v_1, v_2)$$

where $(W, (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\tau]\!]$.

Thus,

$$\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\mathbf{x}^+))) = v_1$$

and

$$\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\mathbf{x}^+))) = v_2$$

Ergo, we must show

$$(W, (\Phi_1, v_1), (\Phi_2, v_2)) \in \mathcal{E}[\![\tau]\!].$$

This trivially follows from $(W, (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\tau]\!]$. and Lemma B.2.2. □

Lemma B.2.31 (Compat \mathbf{a}_{\circ}).

$$\mathbf{a}_{\circ} : \tau \in \Omega \implies \Delta; \Gamma; \Gamma \vdash \mathbf{a}_{\circ} \preceq \mathbf{a}_{\circ} : \tau$$

Proof. Expanding the conclusion, given

$$\begin{aligned} \forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Omega}. \rho \in \mathcal{D}[\![\Delta]\!] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]_{\rho} \\ \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!] \wedge (W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\![\Omega]\!]. \end{aligned}$$

we must show:

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\mathbf{a}_{\circ}^+))), (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\mathbf{a}_{\circ}^+)))) \in \mathcal{E}[\tau].$$

We can push the compiler and the substitutions through this expression to refine this to:

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\mathbf{a}))), (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\mathbf{a})))) \in \mathcal{E}[\tau].$$

Since $(W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\Omega]$, there must exist $(\ell_1, \ell_2) \in W.\Theta$ and values v_1, v_2 such that:

$$\gamma_{\Omega}(\mathbf{a}) = (\text{guard}(v_1, \ell_1), \text{guard}(v_2, \ell_2))$$

where either $W.\Theta(\ell_1, \ell_2) = \text{USED}$ or $W.\Theta = \Theta' \uplus (\ell_1, \ell_2) \mapsto (\Phi_1^*, \Phi_2^*)$ and

$$((W.k, W.\Psi, \Theta'), (\Phi_1^*, v_1), (\Phi_2^*, v_2)) \in \mathcal{V}[\tau].$$

Ergo, we must show:

$$(W, (\Phi_1, \text{guard}(v_1, \ell_1)), (\Phi_2, \text{guard}(v_2, \ell_2))) \in \mathcal{E}[\tau].$$

which we can expand to:

$$(W, (\Phi_1, (\lambda_{_}. \text{if } !\ell_1 \{ \text{fail CONV} \} \{ \ell_1 := \text{USED}; v_1 \})), (\Phi_2, (\lambda_{_}. \text{if } !\ell_2 \{ \text{fail CONV} \} \{ \ell_2 := \text{USED}; v_2 \}))) \in \mathcal{E}[\tau].$$

Notice that both expressions have no free variables because v_1 and v_2 are closed, as they are in the value relation.

Let e_1 and e_2 be the first and second expressions, respectively, in the above tuple. Expanding the definition of the expression relation, given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \Phi_1 \wedge \Phi_{r2} \# \Phi_2 \wedge \Phi_{r1} \uplus \Phi_1, \Phi_{r2} \uplus \Phi_2 : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_1, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau]_{\rho} \end{aligned}$$

Then, by application, we have

$$\begin{aligned} & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, (\lambda _. \text{if } !\ell_1 \{ \text{fail CONV} \} \{ \ell_1 := \text{USED}; v_1 \}) () \rangle \dashrightarrow \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, \text{if } !\ell_1 \{ \text{fail CONV} \} \{ \ell_1 := \text{USED}; v_1 \} \rangle \end{aligned}$$

and

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, (\lambda _. \text{if } !\ell_2 \{ \text{fail CONV} \} \{ \ell_2 := \text{USED}; v_2 \}) () \rangle \dashrightarrow \\ & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, \text{if } !\ell_2 \{ \text{fail CONV} \} \{ \ell_2 := \text{USED}; v_2 \} \rangle \end{aligned}$$

Then, as mentioned before, we have two cases: either $W.\Theta(\ell_1, \ell_2) = \text{USED}$ or $W.\Theta(\ell_1, \ell_2) = (\Phi_1^*, \Phi_2^*)$.

If $W.\Theta(\ell_1, \ell_2) = \text{USED}$, then since $H_1, H_2 : W$, it follows that $H_1(\ell_1) = H_2(\ell_2) = \text{USED}$. In this case, the configuration steps to fail CONV, so we are done.

If $W.\Theta(\ell_1, \ell_2) = (\Phi_1^*, \Phi_2^*)$, then since $H_1, H_2 : W$, it follows that $H_1(\ell_1) = H_2(\ell_2) = \text{UNUSED}$.

$$\begin{aligned} & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, \text{if } !\ell_1 \{ \text{fail CONV} \} \{ \ell_1 := \text{USED}; v_1 \} \rangle \\ \dashrightarrow & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, \ell_1 := \text{USED}; v_1 \rangle \\ \dashrightarrow & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1[\ell_1 \mapsto \text{USED}], v_1 \rangle \end{aligned}$$

and

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, \text{if } !\ell_2 \{ \text{fail CONV} \} \{ \ell_2 := \text{USED}; v_2 \} \rangle \\ \dashrightarrow & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, \ell_2 := \text{USED}; v_2 \rangle \\ \dashrightarrow & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2[\ell_2 \mapsto \text{USED}], v_2 \rangle \end{aligned}$$

Now, consider

$$W' = (W.k, W.\Psi, W.\Theta[(\ell_1, \ell_2) \mapsto \text{USED}])$$

Notice that for all $i \in \{1, 2\}$, $\text{flags}(W, i) = \text{flags}(W', i) \uplus \Phi_i^*$. This is because the dynamic flags in W' are the exact same as W , except (ℓ_1, ℓ_2) has been switched to USED, meaning Φ_1^* has been removed from the left side and Φ_2^* has been removed from the right side. Ergo, since $\Phi_{r1}, \Phi_{r2} \subseteq W$ and $\text{flags}(W', i) \subseteq \text{flags}(W, i)$ for all $i \in \{1, 2\}$, it follows that $\Phi_{r1}, \Phi_{r2} : W'$. Since W and W' also have the same heap typing, we can then conclude that $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W'$.

Next, notice that $H_1[\ell_1 \mapsto \text{USED}], H_2[\ell_2 \mapsto \text{USED}] : W'$ because $H_1, H_2 : W$ and the only change from W to W' is that $W'.\Theta(\ell_1, \ell_2) = \text{USED}$, which is satisfied by both of these new heaps.

Moreover, let $W_b = (W.k, W.\Psi, \Theta')$. The only difference between W_b and W' is that the dynamic flag store in W_b does not contain the locations (ℓ_1, ℓ_2) whereas $W'.\Theta$ contains $(\ell_1, \ell_2) \mapsto \text{USED}$. Furthermore, since for all

$i \in \{1, 2\}$, $\text{flags}(W, i) = \text{flags}(W', i) \uplus \Phi_i^*$, we find that $\text{flags}(W', i) \# \Phi_i^*$ and thus $\Phi_1^*, \Phi_2^* : W'$. Ergo, $W_b \sqsubseteq_{\Phi_1^*, \Phi_2^*} W'$.

Finally, for all $i \in \{1, 2\}$, let $\Phi_{fi} = \Phi_i^*$ and let $\Phi_{gi} = \Phi_i$. We have by assumption that $(W_b, (\Phi_1^*, v_1), (\Phi_2^*, v_2)) \in \mathcal{V}[\tau]_.$, so since $W_b \sqsubseteq_{\Phi_1^*, \Phi_2^*} W'$, by Lemma B.2.8, we have $(W', (\Phi_1^*, v_1), (\Phi_2^*, v_2)) \in \mathcal{V}[\tau]_.$, which suffices to finish the proof. \square

Lemma B.2.32 (Compat a_\bullet).

$$a_\bullet : \tau \in \Omega \implies \Delta; \Gamma; \Omega \vdash a_\bullet \preceq a_\bullet : \tau$$

Proof. Expanding the conclusion, given

$$\forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Omega. \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \emptyset, \emptyset, \gamma_\Omega) \in \mathcal{G}[\Omega]. \wedge (W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega].$$

we must show

$$(W, (\Phi_1, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(a_\bullet^+)))), (\Phi_2, \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(a_\bullet^+)))) \in \mathcal{E}[\tau].$$

Notice that $a_\bullet^+ = a_\bullet$. Then, since $a_\bullet : \tau \in \Omega$ and $(W, \gamma_\Omega) \in \mathcal{G}[\Omega]_.$, then there exist $\Phi'_1, \Phi'_2, v_1, v_2, f_1, f_2$ such that

$$\gamma_\Omega(a_\bullet) = (\text{protect}(v_1, f_1), \text{protect}(v_2, f_2))$$

where $(W, (\Phi'_1, v_1), (\Phi'_2, v_2)) \in \mathcal{V}[\tau]_.$, $\Phi'_1 \cup \{f_1\} \subseteq \Phi_1$, $\Phi'_2 \cup \{f_2\} \subseteq \Phi_2$, $f_1 \notin \Phi'_1$, and $f_2 \notin \Phi'_2$. Thus, we must show

$$(W, (\Phi_1, \text{protect}(v_1, f_1)), (\Phi_2, \text{protect}(v_2, f_2))) \in \mathcal{E}[\tau].$$

Let $e_1 = \text{protect}(v_1, f_1)$ and $e_2 = \text{protect}(v_2, f_2)$. Expanding the definition of the expression relation, given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \Phi_1 \wedge \Phi_{r2} \# \Phi_2 \wedge \Phi_{r1} \uplus \Phi_1, \Phi_{r2} \uplus \Phi_2 : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_1, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau]_\rho \end{aligned}$$

Since $f_1 \in \Phi_1$,

$$\langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, \text{protect}(v_1, f_1) \rangle \dashrightarrow \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1 \setminus \{f_1\}, H_1, v_1 \rangle$$

and since $f_2 \in \Phi_2$,

$$\langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, \text{protect}(v_2, f_2) \rangle \dashrightarrow \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2 \setminus \{f_2\}, H_2, v_2 \rangle$$

Then, since $\Phi'_1 \subseteq \Phi_1$ and $f_1 \notin \Phi'_1$, we have $\Phi'_1 \subseteq \Phi_1 \setminus \{f_1\}$. Similarly, $\Phi'_2 \subseteq \Phi_2 \setminus \{f_2\}$. Ergo, for all $i \in \{1, 2\}$, let $\Phi_{fi} = \Phi'_i$ and let $\Phi_{gi} = \Phi_i \setminus \{f_i\} \setminus \Phi'_i$. Then, we can re-express the above configurations as

$$\langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1 \setminus \{f_1\}, H_1, v_1 \rangle = \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_{f1} \uplus \Phi_{g1}, H_1, v_1 \rangle$$

and

$$\langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2 \setminus \{f_2\}, H_2, v_2 \rangle = \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H_2, v_2 \rangle$$

Finally, we have $(W, (\Phi_{f1}, v_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau]$. because $\Phi_{fi} = \Phi'_i$ for all $i \in \{1, 2\}$, which suffices to finish the proof. \square

Lemma B.2.33 (Compat \multimap).

$$\begin{aligned} & \Delta; \Gamma; \Omega; \mathbf{a}_\circ : \tau_1 \vdash e \preceq e : \tau_2 \wedge \text{no}_\bullet(\Omega) \\ \implies & \Delta; \Gamma; \Omega \vdash \lambda \mathbf{a}_\circ : \tau_1. e \preceq \lambda \mathbf{a}_\circ : \tau_1. e : \tau_1 \multimap \tau_2 \end{aligned}$$

Proof. Expanding the conclusion, given

$$\begin{aligned} & \forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Omega. \\ & \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \emptyset, \emptyset, \gamma_\Omega) \in \mathcal{G}[\Omega]. \wedge (W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega]. \end{aligned}$$

we must show

$$(W, (\Phi_1, \gamma_\Gamma^1(\gamma_\Omega^1(\lambda \mathbf{a}_\circ : \tau_1. e^+))), (\Phi_2, \gamma_\Gamma^2(\gamma_\Omega^2(\lambda \mathbf{a}_\circ : \tau_1. e^+)))) \in \mathcal{E}[\tau].$$

Notice that both of these expressions have no free variables by Lemma B.2.16. Moreover, notice that since $\text{no}_\bullet(\Omega)$, $\Phi_1 = \Phi_2 = \emptyset$.

We can push the compiler and the substitutions to refine the above to:

$$(W, (\emptyset, \lambda \mathbf{a}. \gamma_\Gamma^1(\gamma_\Omega^1(e^+))), (\emptyset, \lambda \mathbf{a}. \gamma_\Gamma^2(\gamma_\Omega^2(e^+)))) \in \mathcal{E}[\tau_1 \multimap \tau_2].$$

Since $\mathcal{V}[\tau_1 \multimap \tau_2] \subseteq \mathcal{E}[\tau_1 \multimap \tau_2]$, it suffices to show:

$$(W, (\emptyset, \lambda \mathbf{a}. \gamma_\Gamma^1(\gamma_\Omega^1(e^+))), (\emptyset, \lambda \mathbf{a}. \gamma_\Gamma^2(\gamma_\Omega^2(e^+)))) \in \mathcal{V}[\tau_1 \multimap \tau_2].$$

Expanding the value relation, given:

$$\forall \Phi'_1 v_1 \Phi'_2 v_2 W'. W \sqsubset_{\emptyset, \emptyset} W' \wedge (W', (\Phi'_1, v_1), (\Phi'_2, v_2)) \in \mathcal{V}[\tau_1].$$

we must show that:

$$((W'.k, W'.\Psi, W'.\Theta \uplus (\ell_1, \ell_2) \mapsto (\Phi'_1, \Phi'_2)), \\ (\emptyset, [a \mapsto \text{guard}(v_1, \ell_1)]\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Gamma}^1(e^+)))), (\emptyset, [a \mapsto \text{guard}(v_2, \ell_2)]\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Gamma}^2(e^+)))) \in \mathcal{E}[\tau_2].$$

Notice that $W'' = (W'.k, W'.\Psi, W'.\Theta \uplus (\ell_1, \ell_2) \mapsto (\Phi'_1, \Phi'_2))$ is a world extension of W' because it has the same heap typing as W' and has all the affine flags as W' plus one new affine flag which is disjoint from any affine flag in W' . Ergo, since $W \sqsubseteq_{\emptyset, \emptyset} W'$ and $W' \sqsubseteq_{\emptyset, \emptyset} W''$, we have $W \sqsubseteq W''$. Next, notice that:

$$(W'', \emptyset, \emptyset, \gamma_{\Omega}[a \mapsto (\text{guard}(v_1, \ell_1), \text{guard}(v_2, \ell_2))]) \in \mathcal{G}[\Omega, a : \tau_1].$$

because $(\ell_1, \ell_2) \in \text{dom}(W''.\Theta)$, $(W'', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau_1]$. (by Lemma B.2.8 and $(W, (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau_1]$), and $(W'', \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega]$. (by Lemma B.2.8 and $(W, \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega]$). Therefore, we can instantiate the first induction hypothesis with

$$W'', \gamma_{\Gamma}, \gamma_{\Omega}, \gamma_{\Gamma}[a \mapsto (\text{guard}(v_1, \ell_1), \text{guard}(v_2, \ell_2))], \rho$$

to find

$$(W'', (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Gamma}^1[a \mapsto \text{guard}(v_1, \ell_1)]^1(e^+)))), \\ (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Gamma}^2[a \mapsto \text{guard}(v_2, \ell_2)]^2(e^+)))) \in \mathcal{E}[\tau_2].$$

which is equivalent to what was to be proven. \square

Lemma B.2.34 (Compat $\dashv \bullet$).

$$\Delta; \Gamma; \Omega, a_{\bullet} : \tau_1 \vdash e \preceq e : \tau_2 \implies \Delta; \Gamma; \Omega \vdash \lambda a_{\bullet} : \tau_1.e \preceq \lambda a_{\bullet} : \tau_1.e : \tau_1 \dashv \bullet \tau_2$$

Proof. Expanding the conclusion, given

$$\forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega}. \\ \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \wedge (W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\Omega].$$

we must show

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\lambda a_{\bullet} : \tau_1^+))), (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\lambda a_{\bullet} : \tau_1^+)))) \in \mathcal{E}[\tau_1 \dashv \bullet \tau_2].$$

By pushing the compiler and substitutions through the lambda expression, we can refine this to:

$$(W, (\Phi_1, \lambda a_{\bullet}. \gamma_{\Gamma}^1(\gamma_{\Omega}^1(e^+))), (\Phi_2, \lambda a_{\bullet}. \gamma_{\Gamma}^2(\gamma_{\Omega}^2(e^+)))) \in \mathcal{E}[\tau_1 \dashv \bullet \tau_2].$$

Since $\mathcal{V}[\tau_1 \multimap \tau_2] \subseteq \mathcal{E}[\tau_1 \multimap \tau_2]$, by Lemma B.2.1, it suffices to show:

$$(W, (\Phi_1, \lambda a_\bullet. \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e^+)))), (\Phi_2, \lambda a_\bullet. \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e^+))))) \in \mathcal{V}[\tau_1 \multimap \tau_2].$$

Expanding the value relation definition, we find that we need to show that given:

$$\begin{aligned} & \forall \Phi'_1 \Phi'_2 f_1 f_2 v_1 v_2 W'. W \sqsubseteq_{\Phi_1, \Phi_2} W' \\ & \wedge (W', (\Phi'_1, v_1), (\Phi'_2, v_2)) \in \mathcal{V}[\tau_1]. \wedge \Phi_1 \cap \Phi'_1 = \Phi_2 \cap \Phi'_2 = \emptyset \\ & \wedge f_1 \notin \Phi_1 \uplus \Phi'_1 \uplus \text{flags}(W', 1) \wedge f_2 \notin \Phi_2 \uplus \Phi'_2 \uplus \text{flags}(W', 2) \end{aligned}$$

it holds that:

$$(W', (\Phi_1 \uplus \Phi'_1 \uplus \{f_1\}, [a_\bullet \mapsto \text{protect}(v_1, f_1)]e_1), (\Phi_2 \uplus \Phi'_2 \uplus \{f_2\}, [a_\bullet \mapsto \text{protect}(v_2, f_2)]e_2)) \in \mathcal{E}[\tau_2].$$

By Lemma B.2.8, since $W \sqsubseteq_{\Phi_1, \Phi_2} W'$, we have $(W', \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega]$. Moreover, we have $(W', (\Phi'_1, v_1), (\Phi'_2, v_2)) \in \mathcal{V}[\tau_1]$, $\Phi_1 \cap \Phi'_1 = \Phi_2 \cap \Phi'_2 = \emptyset$, $f_1 \notin \Phi_1 \uplus \Phi'_1$, and $f_2 \notin \Phi_2 \uplus \Phi'_2$. Therefore,

$$(W', \Phi_1 \uplus \Phi'_1 \uplus \{f_1\}, \Phi_2 \uplus \Phi'_2 \uplus \{f_2\}, \gamma_\Omega[a_\bullet \mapsto (\text{protect}(v_1, f_1), \text{protect}(v_2, f_2))]) \in \mathcal{G}[\Omega, a_\bullet : \tau_1].$$

Then, we can instantiate the first induction hypothesis with

$$W', \gamma_\Gamma, \gamma_\Gamma, \gamma_\Omega[a_\bullet \mapsto (\text{protect}(v_1, f_1), \text{protect}(v_2, f_2))], \rho$$

to find that:

$$(W', (\Phi_1 \uplus \Phi'_1 \uplus \{f_1\}, \gamma_\Gamma^1(\gamma_\Gamma^1(\gamma_\Omega[a_\bullet \mapsto \text{protect}(v_1, f_1)]^1(e^+)))), (\Phi_2 \uplus \Phi'_2 \uplus \{f_2\}, \gamma_\Gamma^2(\gamma_\Gamma^2(\gamma_\Omega[a_\bullet \mapsto \text{protect}(v_2, f_2)]^2(e^+)))))) \in \mathcal{E}[\tau_2].$$

We can simplify this by bringing the $[a_\bullet \mapsto \text{protect}(v_1, f_1)]$ and $[a_\bullet \mapsto \text{protect}(v_2, f_2)]$ outside of the closings, which suffices to finish the proof. \square

Lemma B.2.35 (Compat $\text{app} : \multimap$).

$$\begin{aligned} & \Delta; \Gamma; \Omega_1 \vdash e_1 \preceq e_1 : \tau_1 \multimap \tau_2 \wedge \Delta; \Gamma; \Omega_2 \vdash e_2 \preceq e_2 : \tau_1 \\ & \implies \Delta; \Gamma; \Omega_1 \uplus \Omega_2 \vdash e_1 e_2 \preceq e_1 e_2 : \tau_2 \end{aligned}$$

Proof. Expanding the conclusion, given

$$\begin{aligned} & \forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \\ & \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Omega]. \wedge (W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega_1 \uplus \Omega_2]. \end{aligned}$$

we must show

$$(W, (\Phi_1, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e_1 e_2^+)))), (\Phi_2, \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e_1 e_2^+)))))) \in \mathcal{E}[\tau_2].$$

Notice that both of these expressions have no free variables by Lemma B.2.16.

We can push the compiler and substitutions through the application to refine this to:

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+)))) (\text{let } x = \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_2^+))) \text{ in once}(x))), \\ (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1^+)))) (\text{let } x = \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+))) \text{ in once}(x))) \in \mathcal{E}[\tau_2].$$

Next, by Lemma B.2.5, we have that $\gamma_{\Omega} = \gamma_1 \uplus \gamma_2$, $\Phi_1 = \Phi_{1l} \uplus \Phi_{1r}$, and $\Phi_2 = \Phi_{2l} \uplus \Phi_{2r}$ where

$$(W, \Phi_{1l}, \Phi_{2l}, \gamma_1) \in \mathcal{G}[\Omega_1].$$

and

$$(W, \Phi_{1r}, \Phi_{2r}, \gamma_2) \in \mathcal{G}[\Omega_2].$$

and for all $i \in \{1, 2\}$,

$$\gamma_{\Omega}^i(e_1^+) = \gamma_1^i(e_1^+)$$

and

$$\gamma_{\Omega}^i(e_2^+) = \gamma_1^i(e_2^+)$$

Thus, we refine the statement we need to prove to:

$$(W, (\Phi_{1l} \uplus \Phi_{1r}, \gamma_{\Gamma}^1(\gamma_1^1(\gamma_1^1(e_1^+)))) (\text{let } x = \gamma_{\Gamma}^1(\gamma_2^1(\gamma_2^1(e_2^+))) \text{ in once}(x))), \\ (\Phi_{2l} \uplus \Phi_{2r}, \gamma_{\Gamma}^2(\gamma_2^2(\gamma_2^2(e_1^+)))) (\text{let } x = \gamma_{\Gamma}^2(\gamma_2^2(\gamma_2^2(e_2^+))) \text{ in once}(x))) \in \mathcal{E}[\tau_2].$$

Let e_1 and e_2 be the first and second expressions, respectively, in the above tuple. Expanding the definition of the expression relation, given:

$$\forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ \Phi_{r1} \# \Phi_{1l} \uplus \Phi_{1r} \wedge \Phi_{r2} \# \Phi_{2l} \uplus \Phi_{2r} \wedge \Phi_{r1} \uplus \Phi_{1l} \uplus \Phi_{1r}, \Phi_{r2} \uplus \Phi_{2l} \uplus \Phi_{2r} : W \wedge \\ \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_{1l} \uplus \Phi_{1r}, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_{r1}, H'_1, e'_1 \rangle \rightsquigarrow$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2, H'_2, W'$ such that:

$$\langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_{2l} \uplus \Phi_{2r}, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau]_{\rho})$$

Next, we need to find e'_1 . From the operational semantic, the application will run the first subexpression using the heap H_1 until it reaches a

target value or gets stuck. By appealing to our first induction hypothesis, instantiated with $W, \gamma_{\text{T}}, \gamma_{\text{T}}, \gamma_1, \rho$, we find that:

$$(W, (\Phi_{1l}, \gamma_{\text{T}}^1(\gamma_1^1(\gamma_1^1(\mathbf{e}_1^+)))), (\Phi_{2l}, \gamma_{\text{T}}^2(\gamma_1^2(\gamma_1^2(\mathbf{e}_1^+))))) \in \mathcal{E}[\![\tau_1 \multimap \tau_2]\!].$$

Therefore,

$$\langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_{1r} \uplus \Phi_{1l}, \mathsf{H}_1, \gamma_{\text{T}}^1(\gamma_1^1(\gamma_1^1(\mathbf{e}_1^+))) \rangle$$

either reduces to fail CONV, in which case the original expression steps to fail CONV, or to some irreducible configuration $\langle \Phi_{r1} \uplus \text{flags}(W_1, 1) \uplus \Phi_{1r} \uplus \Phi_{f1l} \uplus \Phi_{g1l}, \mathsf{H}_1^*, \mathbf{e}_1^* \rangle$, in which case on the other side, the configuration

$$\langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_{2r} \uplus \Phi_{2l}, \mathsf{H}_2, \gamma_{\text{T}}^2(\gamma_1^2(\gamma_1^2(\mathbf{e}_1^+))) \rangle$$

reduces to some irreducible configuration $\langle \Phi_{r2} \uplus \text{flags}(W_1, 2) \uplus \Phi_{2r} \uplus \Phi_{f2l} \uplus \Phi_{g2l}, \mathsf{H}_2^*, \mathbf{e}_1^\dagger \rangle$ and there exists some W_1 where $W \sqsubseteq_{\Phi_{r1} \uplus \Phi_{1r}, \Phi_{r2} \uplus \Phi_{2r}} W_1$, $\mathsf{H}_1^*, \mathsf{H}_2^* : W_1$, and $(W_1, (\Phi_{f1l}, \mathbf{e}_1^*), (\Phi_{f2l}, \mathbf{e}_1^\dagger)) \in \mathcal{V}[\![\tau_1 \multimap \tau_2]\!]$. By expanding the value relation, we find that $\Phi_{f1l} = \Phi_{f2l} = \emptyset$.

Since terms in the value relation are target values, the original application will continue reducing on the second subexpression according to the operational semantics. Then, we can appeal to the second induction hypothesis instantiated with $W_1, \gamma_{\text{T}}, \gamma_{\text{T}}, \gamma_2, \rho$, by Lemma B.2.8 because $W \sqsubseteq_{\Phi_{1r}, \Phi_{2r}} W_1$. Ergo,

$$(W_1, (\Phi_{1r}, \gamma_{\text{T}}^1(\gamma_2^1(\gamma_2^1(\mathbf{e}_2^+)))), (\Phi_{2r}, \gamma_{\text{T}}^2(\gamma_2^2(\gamma_2^2(\mathbf{e}_2^+)))) \in \mathcal{E}[\![\tau_1]\!].$$

Therefore,

$$\langle \Phi_{r1} \uplus \text{flags}(W_1, 1) \uplus \Phi_{g1l} \uplus \Phi_{1r}, \mathsf{H}_1^*, \gamma_{\text{T}}^1(\gamma_2^1(\gamma_2^1(\mathbf{e}_2^+))) \rangle$$

either reduces to fail CONV, in which case the original expression steps to fail CONV, or to some irreducible configuration $\langle \Phi_{r1} \uplus \text{flags}(W_2, 1) \uplus \Phi_{g1l} \uplus \Phi_{f1r} \uplus \Phi_{g1r}, \mathsf{H}_1^{**}, \mathbf{e}_2^* \rangle$, in which case on the other side, the configuration

$$\langle \Phi_{r2} \uplus \text{flags}(W_1, 2) \uplus \Phi_{g2l} \uplus \Phi_{2r}, \mathsf{H}_1^*, \gamma_{\text{T}}^2(\gamma_2^2(\gamma_2^2(\mathbf{e}_2^+))) \rangle$$

reduces to some irreducible configuration $\langle \Phi_{r2} \uplus \text{flags}(W_2, 2) \uplus \Phi_{g2l} \uplus \Phi_{f2r} \uplus \Phi_{g2r}, \mathsf{H}_2^{**}, \mathbf{e}_2^\dagger \rangle$ and there exists some W_2 where $W_1 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{g1l}, \Phi_{r2} \uplus \Phi_{g2l}} W_2$, $\mathsf{H}_1^{**}, \mathsf{H}_2^{**} : W_2$, and $(W_2, (\Phi_{f1r}, \mathbf{e}_2^*), (\Phi_{f2r}, \mathbf{e}_2^\dagger)) \in \mathcal{V}[\![\tau_1]\!]$.

Then, instantiate $(W_1, e_1^*, e_1^\dagger) \in \mathcal{V}[\tau_1 \circ \tau_2]$. with $\Phi_{f1r}, e_2^*, \Phi_{f2r}, e_2^\dagger, \triangleright W_2$. Because $W_1 \sqsubseteq_{\emptyset, \emptyset} W_2$ and $W_2 \sqsubset_{\emptyset, \emptyset} \triangleright W_2$, it follows that $W_1 \sqsubset_{\emptyset, \emptyset} \triangleright W_2$. Moreover, $(\triangleright W_2, (\Phi_{f1r}, e_2^*), (\Phi_{f2r}, e_2^\dagger)) \in \mathcal{V}[\tau_1]$, because $\Phi_{f1r}, \Phi_{f2r} : W_2$ by Lemma B.2.8, which implies $\Phi_{f1r}, \Phi_{f2r} : W_2$ and thus $W_2 \sqsubseteq_{\Phi_{f1r}, \Phi_{f2r}} \triangleright W_2$. Ergo, there exist e_b^*, e_b^\dagger such that

$$e_1^* = \lambda a. e_b^*$$

and

$$e_1^\dagger = \lambda a. e_b^\dagger$$

and, for any $(\ell_1, \ell_2) \notin \text{dom}(\triangleright W_2. \Psi) \cup \text{dom}(\triangleright W_2. \Theta)$,

$$((\triangleright W_2. k, \triangleright W_2. \Psi, \triangleright W_2. \Theta \uplus (\ell_1, \ell_2) \mapsto (\Phi_{f1r}, \Phi_{f2r})), \\ (\emptyset, [a \mapsto \text{guard}(e_2^*, \ell_2)]e_b^*), (\emptyset, [a \mapsto \text{guard}(e_2^\dagger, \ell_1)]e_b^\dagger)) \in \mathcal{E}[\tau_2].$$

Let $W_3 = (\triangleright W_2. k, \triangleright W_2. \Psi, \triangleright W_2. \Theta \uplus (\ell_1, \ell_2) \mapsto (\Phi_{f1r}, \Phi_{f2r}))$.

Thus, the original configuration in H_1 steps as follows:

$$\begin{aligned} & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_{1l} \uplus \Phi_{1r} H_1, \\ & \quad \gamma_{\textcolor{blue}{T}}^1(\gamma_{\textcolor{red}{T}}^1(\gamma_{\Omega}^1(e_1^+))) (\text{let } x = \gamma_{\textcolor{blue}{T}}^1(\gamma_{\textcolor{red}{T}}^1(\gamma_{\Omega}^1(e_2^+))) \text{ in once}(x)) \dashrightarrow^* \\ & \langle \Phi_{r1} \uplus \text{flags}(W_1, 1) \uplus \Phi_{1r} \uplus \Phi_{g1l}, H_1^*, \lambda a. e_b^* (\text{let } x = \gamma_{\textcolor{blue}{T}}^1(\gamma_{\textcolor{red}{T}}^1(\gamma_{\Omega}^1(e_2^+))) \text{ in once}(x)) \rangle \dashrightarrow^* \\ & \langle \Phi_{r1} \uplus \text{flags}(W_2, 1) \uplus \Phi_{g1l} \uplus \Phi_{f1r} \uplus \Phi_{g1r}, H_1^{**}, \lambda a. e_b^* (\text{let } x = e_2^* \text{ in once}(x)) \rangle \dashrightarrow \\ & \langle \Phi_{r1} \uplus \text{flags}(W_2, 1) \uplus \Phi_{g1l} \uplus \Phi_{f1r} \uplus \Phi_{g1r}, H_1^{**}, \lambda a. e_b^* \text{ once}(e_2^*) \rangle \dashrightarrow^0 \\ & \langle \Phi_{r1} \uplus \text{flags}(W_2, 1) \uplus \Phi_{g1l} \uplus \Phi_{f1r} \uplus \Phi_{g1r}, H_1^{**}, \\ & \quad \lambda a. e_b^* \text{ let } r_{\text{fresh}} = \text{ref } 1 \text{ in } \lambda_. \{ \text{if } !r_{\text{fresh}} \{ \text{fail CONV} \} \{ r_{\text{fresh}} := \text{USED}; e_2^* \} \} \dashrightarrow \\ & \langle \Phi_{r1} \uplus \text{flags}(W_2, 1) \uplus \Phi_{g1l} \uplus \Phi_{f1r} \uplus \Phi_{g1r}, H_1^{**}[\ell_1 \rightarrow \text{UNUSED}], \\ & \quad \lambda a. e_b^* \lambda_. \{ \text{if } !\ell_1 \{ \text{fail CONV} \} \{ \ell_1 := \text{USED}; e_2^* \} \} \dashrightarrow^0 \\ & \langle \Phi_{r1} \uplus \text{flags}(W_2, 1) \uplus \Phi_{g1l} \uplus \Phi_{f1r} \uplus \Phi_{g1r}, H_1^{**}[\ell_1 \rightarrow \text{UNUSED}], \lambda a. e_b^* \text{ guard}(\ell_1, e_2^*) \rangle \dashrightarrow \\ & \langle \Phi_{r1} \uplus \text{flags}(W_2, 1) \uplus \Phi_{g1l} \uplus \Phi_{f1r} \uplus \Phi_{g1r}, H_1^{**}[\ell_1 \rightarrow \text{UNUSED}], [a \mapsto \text{guard}(\ell_1, e_2^*)]e_b^* \rangle \end{aligned}$$

for some $\ell_1 \notin H_1^{**}$. Similarly, the original configuration in H_2 steps to

$$\langle \Phi_{r2} \uplus \Phi(W_2, 1) \uplus \Phi_{g2l} \uplus \Phi_{f2r} \uplus \Phi_{g2r}, H_2^{**}[\ell_2 \rightarrow 1], [a \mapsto \text{guard}(\ell_2, e_2^\dagger)]e_b^\dagger \rangle$$

for some $\ell_2 \notin H_2^{**}$. Since $H_1^{**}, H_2^{**} : W_2$, this implies $(\ell_1, \ell_2) \notin \text{dom}(W_2. \Psi) \cup \text{dom}(W_2. \Theta)$, and thus $(\ell_1, \ell_2) \notin \text{dom}(\triangleright W_2. \Psi) \cup \text{dom}(\triangleright W_2. \Theta)$.

Therefore, by expanding the value relation for $\tau_1 \circ \tau_2$, we find:

$$(W_3, (\emptyset, [a \mapsto \text{guard}(\ell_1, e_2^*)]e_b^*), (\emptyset, [a \mapsto \text{guard}(\ell_2, e_2^\dagger)]e_b^\dagger)) \in \mathcal{E}[\tau_2].$$

Moreover, since $H_1^{**}, H_2^{**} : W_2$, we also have $H_1^{**}, H_2^{**} : \triangleright W_2$. Therefore, $H_1^{**}[\ell_1 \mapsto \text{UNUSED}], H_2^{**}[\ell_2 \mapsto \text{UNUSED}] : W_3$, because the only difference between $\triangleright W_2$ and W_3 is that W_3 has a new affine flag $(\ell_1, \ell_2) \rightarrow (\Phi_{f1l}, \Phi_{f1r})$,

and both of the above heaps indeed have ℓ_1 and ℓ_2 , respectively, set to UNUSED.

Finally, notice that, for all $i \in \{1, 2\}$, $\text{flags}(W_3, i) = \text{flags}(W_2, i) \uplus \Phi_{fir}$, because W_3 has the exact same dynamic flags as W_2 , except for $(\ell_1, \ell_2) \mapsto (\Phi_{f1r}, \Phi_{f2r})$, which has the effect of adding Φ_{f1r} on the left side and Φ_{f2r} on the right side. Thus, we can rewrite the above configurations as

$$\begin{aligned} & \langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \Phi_{g1r} \uplus \text{flags}(W_3, 1), H_1^{**}[\ell_1 \rightarrow \text{UNUSED}], [a \mapsto \text{guard}(\ell_1, e_2^*)]e_b^* \rangle \\ & \langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \Phi_{g2r} \uplus \text{flags}(W_3, 1), H_2^{**}[\ell_2 \rightarrow \text{UNUSED}], [a \mapsto \text{guard}(\ell_2, e_2^\dagger)]e_b^\dagger \rangle \end{aligned}$$

Ergo, we can instantiate the fact that the above expressions are in $\mathcal{E}[\tau_2]$. in the world W_3 to find that either the first configuration steps to fail CONV, in which case the original configuration with H_1 steps to fail CONV, or the first configuration steps to some irreducible configuration

$$\langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \Phi_{g1r} \uplus \text{flags}(W_4, 1) \uplus \Phi_{f1n} \uplus \Phi_{g1n}, H_1^{***}, e_f^* \rangle$$

in which case the second configuration steps to

$$\langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \Phi_{g2r} \uplus \text{flags}(W_4, 2) \uplus \Phi_{f2n} \uplus \Phi_{g2n}, H_2^{***}, e_f^\dagger \rangle$$

and there exists some W_4 such that $W_3 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{g1l} \uplus \Phi_{g1r}, \Phi_{r2} \uplus \Phi_{g2l} \uplus \Phi_{g2r}} W_4$, $H_1^{***}, H_2^{***} : W_4$, and $(W_4, (\Phi_{f1n}, e_f^*), (\Phi_{f2n}, e_f^\dagger)) \in \mathcal{V}[\tau_2]$.

This suffices to show that $e'_1 = e^{***}$, so e'_1 is indeed in the value relation at τ_2 along with the value stepped to by the original configuration on the right hand side. Ergo, since $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$, $W_1 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, $W_2 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, $\triangleright W_2 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_3$, and $W_3 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_4$ (note that these are weaker statements of what we learned above, but hold – and in particular, via transitivity, will be what hold), it follows that $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_4$, which suffices to finish the proof. \square

Lemma B.2.36 (Compat $\text{app} : \text{--}\bullet$).

$$\begin{aligned} & \Delta; \Gamma; \Omega_1 \vdash e_1 \preceq e_1 : \tau_1 \multimap \tau_2 \wedge \Delta; \Gamma; \Omega_2 \vdash e_2 \preceq e_2 : \tau_1 \\ & \implies \Delta; \Gamma; \Omega_1 \uplus \Omega_2 \vdash e_1 e_2 \preceq e_1 e_2 : \tau_2 \end{aligned}$$

Proof. Expanding the conclusion, given

$$\begin{aligned} & \forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega. \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \\ & \wedge (W, \emptyset, \emptyset, \gamma_\Omega) \in \mathcal{G}[\Omega]. \wedge (W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega_1 \uplus \Omega_2]. \end{aligned}$$

we must show

$$(W, (\Phi_1, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e_1 e_2^+))))), (\Phi_2, \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e_1 e_2^+)))) \in \mathcal{E}[\tau_2].$$

By pushing the compiler and substitutions through the application, we can refine this to:

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+)))) \ \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_2^+)))), (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1^+)))) \ \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+)))) \in \mathcal{E}[\tau_2].$$

Next, by Lemma B.2.5, we have that $\gamma_{\Omega} = \gamma_1 \uplus \gamma_2$, $\Phi_1 = \Phi_{1l} \uplus \Phi_{1r}$, and $\Phi_2 = \Phi_{2l} \uplus \Phi_{2r}$ where

$$(W, \Phi_{1l}, \Phi_{2l}, \gamma_1) \in \mathcal{G}[\Omega].$$

and

$$(W, \Phi_{1r}, \Phi_{2r}, \gamma_2) \in \mathcal{G}[\Omega].$$

and for all $i \in \{1, 2\}$,

$$\gamma_{\Omega}^i(e_1^+) = \gamma_1^i(e_1^+)$$

and

$$\gamma_{\Omega}^i(e_2^+) = \gamma_1^i(e_2^+)$$

Thus, we can refine the statement we need to prove as:

$$(W, (\Phi_{1l} \uplus \Phi_{1r}, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_1^1(e_1^+)))) \ \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_2^1(e_2^+)))), (\Phi_{2l} \uplus \Phi_{2r}, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_1^2(e_1^+)))) \ \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_2^2(e_2^+)))) \in \mathcal{E}[\tau_2].$$

Let e_1 and e_2 be the first and second expressions, respectively, in the tuple above. Expanding the definition of the expression relation, given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \Phi_{1l} \uplus \Phi_{1r} \wedge \Phi_{r2} \# \Phi_{2l} \uplus \Phi_{2r} \wedge \Phi_{r1} \uplus \Phi_{1l} \uplus \Phi_{1r}, \Phi_{r2} \uplus \Phi_{2l} \uplus \Phi_{2r} : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_{1l} \uplus \Phi_{1r}, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_{r1}, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_{2l} \uplus \Phi_{2r}, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau]_{\rho} \end{aligned}$$

By instantiating the first induction hypothesis with $W, \gamma_{\Gamma}, \gamma_{\Omega}, \gamma_1, \rho$, we find that:

$$(W, (\Phi_{1l}, \gamma_{\Gamma}^1(\gamma_1^1(\gamma_1^1(e_1^+)))), (\Phi_{1r}, \gamma_{\Gamma}^2(\gamma_1^2(\gamma_1^2(e_1^+)))) \in \mathcal{E}[\tau_1 \bullet \tau_2].$$

Thus,

$$\langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W, 1) \uplus \Phi_{1l}, H_1, \gamma_{\Gamma}^1(\gamma_1^1(\gamma_1^1(e_1^+))) \rangle$$

either steps to fail CONV, in which case the whole expression steps to fail CONV, or steps to an irreducible configuration $\langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1l} \uplus \Phi_{g1l}, H_1^*, e_1^* \rangle$, in which case

$$\langle \Phi_{r1} \uplus \Phi_{2r} \uplus \text{flags}(W, 2) \uplus \Phi_{2l}, H_2, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_1^2(e_1^+))) \rangle$$

also steps to an irreducible configuration $\langle \Phi_{r1} \uplus \Phi_{2r} \uplus \text{flags}(W_1, 2) \uplus \Phi_{f2l} \uplus \Phi_{g2l}, H_2^*, e_2^* \rangle$ and there exists some world W_1 where $W \sqsubseteq_{\Phi_{r1} \uplus \Phi_{1r}, \Phi_{r2} \uplus \Phi_{2r}} W_1$, $H_1^*, H_2^* : W_1$, and $(W_1, (\Phi_{f1l}, e_1^*), (\Phi_{f2l}, e_2^*)) \in \mathcal{V}[\tau_1 \bullet \tau_2]..$. By expanding the value relation, there exist expressions e_{b1}^* , e_{b2}^* such that $e_1^* = \lambda a_{\bullet}.e_{b1}^*$ and $e_2^* = \lambda a_{\bullet}.e_{b2}^*$.

Then, by the operational semantic, the original application expression continues reducing on the second subexpression. By instantiating the second induction hypothesis with $W_1, \gamma_{\Gamma}, \gamma_{\Gamma}, \gamma_2, \rho$, we find that:

$$(W_1, (\Phi_{2l}, \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_2^+)))), (\Phi_{2r}, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_2^2(e_2^+))))) \in \mathcal{E}[\tau_1].$$

Thus,

$$\langle \Phi_{r1} \uplus \Phi_{f1l} \uplus \Phi_{g1l} \uplus \text{flags}(W_1, 1) \uplus \Phi_{1r}, H_1^*, \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_2^+))) \rangle$$

either steps to fail CONV, in which case the whole expression steps to fail CONV, or steps to an irreducible configuration $\langle \Phi_{r1} \uplus \Phi_{f1l} \uplus \Phi_{g1l} \uplus \text{flags}(W_2, 1) \uplus \Phi_{f1r} \uplus \Phi_{g1r}, H_1^{\dagger}, e_1^{\dagger} \rangle$, in which case

$$\langle \Phi_{r2} \uplus \Phi_{f2l} \uplus \Phi_{g2l} \uplus \text{flags}(W_1, 2) \uplus \Phi_{2r}, H_2^*, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_2^2(e_2^+))) \rangle$$

also steps to an irreducible configuration $\langle \Phi_{r2} \uplus \Phi_{f2l} \uplus \Phi_{g2l} \uplus \text{flags}(W_2, 2) \uplus \Phi_{f2r} \uplus \Phi_{g2r}, H_2^{\dagger}, e_2^{\dagger} \rangle$ and there exists some world W_2 where $W_1 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{f1l} \uplus \Phi_{g1l}, \Phi_{r2} \uplus \Phi_{f2l}, \Phi_{g2l}} W_2$, $H_1^{\dagger}, H_2^{\dagger} : W_2$, and

$$(W_2, (\Phi_{f1r}, e_1^{\dagger}), (\Phi_{f2r}, e_2^{\dagger})) \in \mathcal{V}[\tau_1].$$

Thus, the original configuration with H_1 steps to

$$\langle \Phi_{r1} \uplus \Phi_{f1l} \uplus \Phi_{g1l} \uplus \text{flags}(W_2, 1) \uplus \Phi_{f1r} \uplus \Phi_{g1r}, H_1^{\dagger}, \lambda a_{\bullet}.e_{b1}^* e_1^{\dagger} \rangle$$

which steps to

$$\langle \Phi_{r1} \uplus \Phi_{f1l} \uplus \Phi_{g1l} \uplus \text{flags}(W_2, 1) \uplus \Phi_{f1r} \uplus \Phi_{g1r} \uplus \{f_1\}, H_1^{\dagger}, [a_{\bullet} \mapsto \text{protect}(e_1^{\dagger}, f_1)]e_{b1}^* \rangle$$

for some $f_1 \notin \Phi_{r1} \uplus \Phi_{f1l} \uplus \Phi_{g1l} \uplus \text{flags}(W_2, 1) \uplus \Phi_{f1r} \uplus \Phi_{g1r}$.

Similarly, the original configuration with H_2 steps to

$$\langle \Phi_{r2} \uplus \Phi_{f2l} \uplus \Phi_{g2l} \uplus \text{flags}(W_2, 2) \uplus \Phi_{f2r} \uplus \Phi_{g2r} \uplus \{f_2\}, H_2^{\dagger}, [a_{\bullet} \mapsto \text{protect}(e_2^{\dagger}, f_2)]e_{b2}^* \rangle$$

for some $f_2 \notin \Phi_{r2} \uplus \Phi_{f2l} \uplus \Phi_{g2l} \uplus \text{flags}(W_2, 2) \uplus \Phi_{f2r} \uplus \Phi_{g2r}$.

We can instantiate the fact that $(W_1, (\Phi_{f1l}, \lambda a_\bullet.e_{b1}^*), (\Phi_{f2l}, \lambda a_\bullet.e_{b2}^*)) \in \mathcal{V}[\tau_1 \bullet \tau_2]$. with $\Phi_{f1r}, \Phi_{f2r}, f_1, f_2, e_1^\dagger, e_2^\dagger, W_2$ to find that:

$$(W_2, (\Phi_{f1l} \uplus \Phi_{f1r} \uplus \{f_1\}, [a_\bullet \mapsto \text{protect}(e_1^\dagger, f_1)]e_{b1}^*), (\Phi_{f2l} \uplus \Phi_{f2r} \uplus \{f_2\}, [a_\bullet \mapsto \text{protect}(e_2^\dagger, f_2)]e_{b2}^*)) \in \mathcal{E}[\tau_2].$$

Given $H_1^\dagger, H_2^\dagger : W_2$, it follows that

$$\langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \Phi_{g1r} \uplus \text{flags}(W_2, 1) \uplus \Phi_{f1l} \uplus \Phi_{f1r} \uplus \{f_1\}, H_1^\dagger, [a_\bullet \mapsto \text{protect}(e_1^\dagger, f_1)]e_{b1}^* \rangle$$

either steps to fail CONV, in which case the original configuration with H_1 steps to fail CONV, or steps to an irreducible configuration

$$\langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \Phi_{g1r} \uplus \text{flags}(W_3, 1) \uplus \Phi_{f1f} \uplus \Phi_{g1f}, H_1^{**}, e_1^{**} \rangle$$

in which case the configuration

$$\langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \Phi_{g2r} \uplus \text{flags}(W_2, 2) \uplus \Phi_{f2l} \uplus \Phi_{f2r} \uplus \{f_2\}, H_2^\dagger, [a_\bullet \mapsto \text{protect}(e_2^\dagger, f_2)]e_{b2}^* \rangle$$

also steps to an irreducible configuration

$$\langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \Phi_{g2r} \uplus \text{flags}(W_3, 2) \uplus \Phi_{f2f} \uplus \Phi_{g2f}, H_2^{**}, e_2^{**} \rangle$$

and there exists a world W_3 such that $W_2 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{g1l} \uplus \Phi_{g1r}, \Phi_{r2} \uplus \Phi_{g2l} \uplus \Phi_{g2r}} W_3$, $H_1^{**}, H_2^{**} : W_3$, and $(W_3, (\Phi_{f1f}, e_1^{**}), (\Phi_{f2f}, e_2^{**})) \in \mathcal{V}[\tau_2]$. Finally, since $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$, $W_1 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, and $W_2 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_3$, it follows that $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_3$, which suffices to finish the proof. \square

Lemma B.2.37 (Compat !).

$$\Delta; \Gamma; \Gamma; \cdot \vdash v : \tau \implies \Delta; \Gamma; \Gamma; \cdot \vdash !v : !\tau$$

Proof. Expanding the conclusion, given

$$\begin{aligned} & \forall W. \forall \rho. \gamma_\Gamma \gamma_\Gamma \gamma_\Omega. \\ & \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]. \\ & \wedge (W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\cdot]. \end{aligned}$$

we must show

$$(W, (\Phi_1, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(!v^+)))), (\Phi_2, \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(!v^+)))))) \in \mathcal{E}[!\tau].$$

Notice that both of these expressions have no free variables by Lemma B.2.16. Moreover, since $(W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\cdot]_.$, we have $\Phi_1 = \Phi_2 = \emptyset$ and $\gamma_{\Omega} = \cdot$. Furthermore, $!\mathbf{v}^+ = \mathbf{v}^+$. Thus, we can refine the above to:

$$(W, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\mathbf{v}^+))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\mathbf{v}^+)))) \in \mathcal{E}[!\tau].$$

Let e_1 and e_2 be the first and second expressions, respectively, in the above tuple. Expanding the definition of the expression relation, given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \emptyset \wedge \Phi_{r2} \# \emptyset \wedge \Phi_{r1} \uplus \emptyset, \Phi_{r2} \uplus \emptyset : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \emptyset, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_1, H'_1, e'_1 \rangle \rightarrow \end{aligned}$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \emptyset, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[!\tau]_{\rho} \end{aligned}$$

Next, consider $W_1 = (W.k, W.\Psi, \Theta')$, where $\text{dom}(\Theta') = \text{dom}(W.\Theta)$ and for all $(\ell_1, \ell_2) \in \text{dom}(W.\Theta)$, $\Theta'(\ell_1, \ell_2) = \text{USED}$. Thus, since all dynamic flags in W_1 have been used $\text{flags}(W_1, 1) = \text{flags}(W_1, 2) = \emptyset$, so we trivially have $\Phi_{r1}, \Phi_{r2} : W_1$. It then follows that $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$ because W and W_1 have the exact same heap typing and all of the locations in W have been switched to USED in W_1 .

Thus, by Lemma B.2.8, we have $(W_1, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho}$ and $(W_1, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Omega]_.$ We also trivially have $(W_1, \emptyset, \emptyset, \cdot) \in \mathcal{G}[\cdot]_.$ Thus, by instantiating the first induction hypothesis with $W_1, \gamma_{\Gamma}, \gamma_{\Gamma}, \cdot, \rho$, we find:

$$(W_1, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\mathbf{v}^+))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\mathbf{v}^+)))) \in \mathcal{E}[\tau].$$

Thus, since $\text{flags}(W_1, 1) = \text{flags}(W_1, 2) = \emptyset$, the configuration

$$\langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \emptyset \uplus \emptyset, H_1, \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\mathbf{v}^+)) \rangle$$

must either step to fail CONV, in which case the proof is done, or steps to some irreducible configuration $\langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \emptyset \uplus \Phi_{f1} \uplus \Phi_{g1}, H_1^*, e_1^* \rangle$, in which case the configuration

$$\langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \emptyset \uplus \emptyset, H_2, (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\mathbf{v}^+))) \rangle$$

steps to an irreducible configuration $\langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \emptyset \uplus \Phi_{f2} \uplus \Phi_{g2}, H_1^*, e_2^* \rangle$, and there exists some world W_2 such that $W_1 \sqsubseteq_{\Phi_{r1} \uplus \text{flags}(W, 1), \Phi_{r2} \uplus \text{flags}(W, 2)} W_2$, $H_1^*, H_2^* : W_2$, and $(W_2, (\Phi_{f1}, e_1^*), (\Phi_{f2}, e_2^*)) \in \mathcal{V}[\tau]..$

However, from Lemma B.2.4, we know both the original configurations above are indeed irreducible and do not step. This means the set of static flags in the original configurations equal that in the final configurations. Thus,

$$\Phi_{r1} \uplus \text{flags}(W, 1) \uplus \emptyset \uplus \emptyset = \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \emptyset \uplus \Phi_{f1} \uplus \Phi_{g1}$$

and

$$\Phi_{r2} \uplus \text{flags}(W, 2) \uplus \emptyset \uplus \emptyset = \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \emptyset \uplus \Phi_{f2} \uplus \Phi_{g2}$$

This implies $\Phi_{f1} = \Phi_{g1} = \emptyset$ and $\Phi_{f2} = \Phi_{g2} = \emptyset$. Ergo, $(W_2, (\emptyset, e_1^*), (\emptyset, e_2^*)) \in \mathcal{V}[\tau]..$, from which it follows that $(W_2, (\emptyset, e_1^*), (\emptyset, e_2^*)) \in \mathcal{V}[\mathbf{!}\tau]..$

Finally, since $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$ and $W_1 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, we have $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, which suffices to finish the proof. \square

Lemma B.2.38 (Compat **let!**).

$$\begin{aligned} & \Delta; \Gamma; \Omega_1 \vdash e_1 \preceq e_1 : !\tau \wedge \Delta; \Gamma; \Omega_2 \vdash e_2 \preceq e_2 : \tau' \\ \implies & \Delta; \Gamma; \Omega_1 \uplus \Omega_2 \vdash \text{let } !x = e_1 \text{ in } e_2 \preceq \text{let } !x = e_1 \text{ in } e_2 : \tau' \end{aligned}$$

Proof. Expanding the conclusion, given

$$\begin{aligned} & \forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Omega}. \\ & \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \\ & \wedge (W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\Omega_1 \uplus \Omega_2]. \end{aligned}$$

we must show

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\text{let } !x = e_1 \text{ in } e_2^+)))), \\ (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\text{let } !x = e_1 \text{ in } e_2^+)))) \in \mathcal{E}[\tau'].$$

Notice that both of these expressions have no free variables by Lemma B.2.16.

We can push the compiler and substitutions through the **let** expression and refine this to:

$$(W, (\Phi_1, \text{let } x = \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+))) \text{ in } \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_2^+)))), \\ (\Phi_2, \text{let } x = \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1^+))) \text{ in } \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+)))) \in \mathcal{E}[\tau'].$$

Next, by Lemma B.2.5, we have that $\gamma_{\Omega} = \gamma_1 \uplus \gamma_2$, $\Phi_1 = \Phi_{1l} \uplus \Phi_{1r}$, and $\Phi_2 = \Phi_{2l} \uplus \Phi_{2r}$ where

$$(W, \Phi_{1l}, \Phi_{2l}, \gamma_1) \in \mathcal{G}[\![\Omega_1]\!].$$

and

$$(W, \Phi_{1r}, \Phi_{2r}, \gamma_2) \in \mathcal{G}[\![\Omega_2]\!].$$

and for all $i \in \{1, 2\}$,

$$\gamma_{\Omega}^i(\mathbf{e}_1^+) = \gamma_1^i(\mathbf{e}_1^+)$$

and

$$\gamma_{\Omega}^i(\mathbf{e}_2^+) = \gamma_1^i(\mathbf{e}_2^+)$$

Thus, we must show

$$(W, (\Phi_{1l} \uplus \Phi_{1r}, \text{let } x = \gamma_{\Gamma}^1(\gamma_1^1(\mathbf{e}_1^+)) \text{ in } \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_2^1(\mathbf{e}_2^+)))), (\Phi_{2l} \uplus \Phi_{2r}, \text{let } x = \gamma_{\Gamma}^2(\gamma_1^2(\mathbf{e}_1^+)) \text{ in } \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_2^2(\mathbf{e}_2^+)))) \in \mathcal{E}[\![\tau']\!].$$

Let \mathbf{e}_1 and \mathbf{e}_2 be the first and second expressions, respectively, in the above tuple. Expanding the definition of the expression relation, given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \Phi_{1l} \uplus \Phi_{1r} \wedge \Phi_{r2} \# \Phi_{2l} \uplus \Phi_{2r} \wedge \Phi_{r1} \uplus \Phi_{1l} \uplus \Phi_{1r}, \Phi_{r2} \uplus \Phi_{2l} \uplus \Phi_{2r} : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_{1l} \uplus \Phi_{1r}, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_{r1}, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_{2l} \uplus \Phi_{2r}, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\![\tau]\!]_{\rho} \end{aligned}$$

Next, we need to find e'_1 . From the operational semantic, the application will run the first subexpression using the heap H_1 until it reaches a target value or gets stuck. By appealing to our first induction hypothesis, instantiated with $W, \gamma_{\Gamma}, \gamma_{\Omega}, \gamma_1, \rho$, we find that:

$$(W, (\Phi_{1l}, \gamma_{\Gamma}^1(\gamma_1^1(\mathbf{e}_1^+))), (\Phi_{1r}, \gamma_{\Gamma}^2(\gamma_1^2(\mathbf{e}_1^+)))) \in \mathcal{E}[\![\!\tau]\!].$$

Therefore, the configuration

$$\langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W, 1) \uplus \Phi_{1l}, H_1, \gamma_{\Gamma}^1(\gamma_1^1(\mathbf{e}_1^+)) \rangle$$

either reduces to fail CONV, in which case the original expression steps to fail CONV, or to some irreducible configuration

$\langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1l} \uplus \Phi_{g1l}, H_1^*, e_1^* \rangle$, in which case the configuration

$$\langle \Phi_{r2} \uplus \Phi_{2r} \uplus \text{flags}(W, 2) \uplus \Phi_{2l}, H_2, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_1^2(e_1^+))) \rangle$$

also reduces to some irreducible configuration $\langle \Phi_{r2} \uplus \Phi_{2r} \uplus \text{flags}(W_1, 2) \uplus \Phi_{f2l} \uplus \Phi_{g2l}, H_2^*, e_1^\dagger \rangle$ and there exists some W_1 where $W \sqsubseteq_{\Phi_{r1} \uplus \Phi_{1r}, \Phi_{r2} \uplus \Phi_{2r}} W_1$, $H_1^*, H_2^* : W_1$, and $(W_1, (\Phi_{f1l}, e_1^*), (\Phi_{f1r}, e_1^\dagger)) \in \mathcal{V}[\![\tau]\!]$. By expanding the value relation definition, we find $\Phi_{f1l} = \Phi_{f1r} = \emptyset$ and $(W_1, (\emptyset, e_1^*), (\emptyset, e_1^\dagger)) \in \mathcal{V}[\![\tau]\!]$.

Since terms in the value relation are target values, the original configuration with H_1 steps as follows:

$$\begin{aligned} & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_{1l} \uplus \Phi_{1r}, H_1, \text{let } x = \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_1^1(e_1^+))) \text{ in } \rangle \xrightarrow{*} \\ & \quad \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_2^+))) \\ & \langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W_1, 1) \uplus \Phi_{g1l}, H_1^*, \text{let } x = e_1^* \text{ in } \gamma_{\Gamma}^1(\gamma_1^1(\gamma_2^1(e_2^+))) \rangle \rightarrow \\ & \langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W_1, 1) \uplus \Phi_{g1l}, H_1^*, [x \mapsto e_1^*] \gamma_{\Gamma}^1(\gamma_1^1(\gamma_2^1(e_2^+))) \rangle \end{aligned}$$

and similarly, the original configuration with H_2 steps to:

$$\langle \Phi_{r2} \uplus \Phi_{2r} \uplus \text{flags}(W_1, 2) \uplus \Phi_{g2l}, H_2^*, [x \mapsto e_1^\dagger] \gamma_{\Gamma}^2(\gamma_2^2(e_2^+)) \rangle$$

Next, notice that $(W_1, \emptyset, \emptyset, \gamma_{\Gamma}[x \mapsto (e_1^*, e_1^\dagger)]) \in \mathcal{G}[\![\Gamma, x : \tau]\!]$. because $(W_1, (\emptyset, e_1^*), (\emptyset, e_1^\dagger)) \in \mathcal{V}[\![\tau]\!]$ and $(W_1, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]$. (which follows from Lemma B.2.8 because $W \sqsubseteq_{\emptyset, \emptyset} W_1$ and $(W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]$). Therefore, by instantiating the second induction hypothesis with $W_1, \gamma_{\Gamma}, \gamma_{\Gamma}[x \mapsto (e_1^*, e_1^\dagger)], \gamma_2, \rho$, we find that

$$(W_1, (\Phi_{1r}, [x \mapsto e_1^*] \gamma_{\Gamma}^1(\gamma_1^1(\gamma_2^1(e_2^+)))), (\Phi_{2r}, [x \mapsto e_1^\dagger] \gamma_{\Gamma}^2(\gamma_2^2(e_2^+)))) \in \mathcal{E}[\![\tau']\!].$$

Then, since $H_1^*, H_2^* : W_1$, we can instantiate the above fact with H_1^* and H_2^* . Ergo, the configuration

$$\langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \text{flags}(W_1, 1) \uplus \Phi_{1r}, H_1^*, [x \mapsto e_1^*] \gamma_{\Gamma}^1(\gamma_1^1(\gamma_2^1(e_2^+))) \rangle$$

must either step to fail CONV, in which case the original expression steps to fail CONV, or it must step to some $\langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \text{flags}(W_2, 1) \uplus \Phi_{f1r} \uplus \Phi_{g1r}, H_1^\dagger, e_1^{**} \rangle$, in which case the configuration on the other side

$$\langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \text{flags}(W_1, 2) \uplus \Phi_{2r}, H_2^*, [x \mapsto e_1^\dagger] \gamma_{\Gamma}^2(\gamma_2^2(e_2^+)) \rangle$$

must step to $\langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \text{flags}(W_2, 2) \uplus \Phi_{f2r} \uplus \Phi_{g2r}, H_2^\dagger, e_1^{\dagger\dagger} \rangle$ for some heap H_2^\dagger and world W_2 where $W_1 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{g1l}, \Phi_{r2} \uplus \Phi_{g2l}} W_2$, $H_1^\dagger, H_2^\dagger : W_2$, and $(W_2, (\Phi_{f2r}, e_1^{**}), (\Phi_{f2r}, e_1^{\dagger\dagger})) \in \mathcal{V}[\![\tau']\!]$. Finally, since $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$ and

$W_1 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, we have $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, which suffices to finish the proof. \square

Lemma B.2.39 (Compat &).

$$\begin{aligned} \Delta; \Gamma; \Omega \vdash e_1 \preceq e_1 : \tau_1 \wedge \Delta; \Gamma; \Omega \vdash e_2 \preceq e_2 : \tau_2 \\ \implies \Delta; \Gamma; \Omega \vdash \langle e_1, e_2 \rangle \preceq \langle e_1, e_2 \rangle : \tau_1 \& \tau_2 \end{aligned}$$

Proof. Expanding the conclusion, given

$$\begin{aligned} \forall W. \forall \rho. \gamma_\Gamma \gamma_\Omega \gamma_\Omega. \\ \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \emptyset, \emptyset, \gamma_\Omega) \in \mathcal{G}[\Omega]. \wedge (W, \Phi_1, \Phi_2, \gamma_\Omega) \in \mathcal{G}[\Omega]. \end{aligned}$$

we must show

$$(W, (\Phi_1, \gamma_\Gamma^1(\gamma_\Omega^1(\langle e_1, e_2 \rangle^+))), (\Phi_2, \gamma_\Gamma^2(\gamma_\Omega^2(\langle e_1, e_2 \rangle^+)))) \in \mathcal{E}[\tau_1 \& \tau_2].$$

Note that both of these expressions are closed by Lemma B.2.16.

We can push the compiler and substitutions through the product expression and refine this to:

$$(W, (\Phi_1, (\lambda_- \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e_1^+)))), \lambda_- \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e_2^+)))), \\ (\Phi_2, (\lambda_- \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e_1^+)))), \lambda_- \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e_2^+))))) \in \mathcal{E}[\tau_1 \& \tau_2].$$

Since $\mathcal{V}[\tau_1 \& \tau_2] \subseteq \mathcal{E}[\tau_1 \& \tau_2]$. by Lemma B.2.1, it suffices to show

$$(W, (\Phi_1, (\lambda_- \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e_1^+)))), \lambda_- \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e_2^+)))), \\ (\Phi_2, (\lambda_- \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e_1^+)))), \lambda_- \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e_2^+))))) \in \mathcal{V}[\tau_1 \& \tau_2].$$

First, we can instantiate the first induction hypothesis with $W, \gamma_\Gamma, \gamma_\Omega, \rho$ to show that

$$(W, (\Phi_1, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e_1^+)))), (\Phi_2, \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e_1^+))))) \in \mathcal{V}[\tau_1].$$

and we can instantiate the second induction hypothesis with $W, \gamma_\Gamma, \gamma_\Omega, \rho$ to show that

$$(W, (\Phi_1, \gamma_\Gamma^1(\gamma_\Omega^1(\gamma_\Omega^1(e_2^+)))), (\Phi_2, \gamma_\Gamma^2(\gamma_\Omega^2(\gamma_\Omega^2(e_2^+))))) \in \mathcal{V}[\tau_2].$$

This suffices to show that the pairs of lambdas are in the value relation at $\tau_1 \& \tau_2$, as was to be proven. \square

Lemma B.2.40 (Compat .1).

$$\Delta; \Gamma; \Omega \vdash e \preceq e : \tau_1 \& \tau_2 \implies \Delta; \Gamma; \Omega \vdash e.1 \preceq e.1 : \tau_1$$

Proof. Expanding the conclusion, given

$$\begin{aligned} & \forall W. \forall \rho \gamma_{\Gamma} \gamma_{\Omega}. \\ & \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \end{aligned}$$

we must show

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\mathbf{e}.\mathbf{1}^+))))), (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\mathbf{e}.\mathbf{1}^+)))) \in \mathcal{E}[\tau_1].$$

Notice that both of these expressions have no free variables by Lemma B.2.16.

We can push the compiler and substitutions through the projection to refine this to:

$$(W, (\Phi_1, (\text{fst } \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\mathbf{e}^+))))()), (\Phi_2, (\text{fst } \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\mathbf{e}^+))))()) \in \mathcal{E}[\tau_1].$$

Expanding the definition of the expression relation, given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \Phi_1 \wedge \Phi_{r2} \# \Phi_2 \wedge \Phi_{r1} \uplus \Phi_1, \Phi_{r2} \uplus \Phi_2 : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, e_1 \rangle \xrightarrow{-j} \langle \Phi'_1, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau]_{\rho} \end{aligned}$$

To proceed, we must find out what e'_1 is. First, by instantiating the first induction hypothesis with $W, \gamma_{\Gamma}, \gamma_{\Omega}, \rho$, we find

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\mathbf{e}^+))))), (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\mathbf{e}^+)))) \in \mathcal{E}[\tau_1 \& \tau_2].$$

Therefore, the configuration

$$\langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\mathbf{e}^+))) \rangle$$

either steps to fail CONV, in which case the original expression steps to fail CONV, or steps to some irreducible configuration $\langle \Phi_{r1} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1} \uplus \Phi_{g1}, H_1^*, e^* \rangle$, in which case the configuration

$$\langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_2, H_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\mathbf{e}^+))) \rangle$$

also steps to some irreducible configuration $\langle \Phi_{r2} \uplus \text{flags}(W_1, 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H_2^*, e^\dagger \rangle$ and there exists some world W_1 where $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$, $H_1^*, H_2^* : W_1$, and $(W_1, (\Phi_{f1}, e^*), (\Phi_{f2}, e^\dagger)) \in \mathcal{V}[\tau_1 \& \tau_2]..$

Ergo, there exists some $e_1^*, e_1^\dagger, e_2^*, e_2^\dagger$ such that

$$e^* = (\lambda_- e_1^*, \lambda_- e_2^*)$$

and

$$e^\dagger = (\lambda_- e_1^\dagger, \lambda_- e_2^\dagger)$$

and

$$(W_1, (\Phi_{f1}, e_1^*), (\Phi_{f2}, e_1^\dagger)) \in \mathcal{E}[\tau_1].$$

and

$$(W_1, (\Phi_{f1}, e_2^*), (\Phi_{f2}, e_2^\dagger)) \in \mathcal{E}[\tau_2].$$

Thus, the original configuration with H_1 steps as follows:

$$\begin{aligned} & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_1, H_1, (\text{fst } \gamma_{\textcolor{blue}{T}}^1(\gamma_{\textcolor{brown}{O}}^1(\gamma_{\Omega}^1(e^+)))) () \rangle \xrightarrow{*} \\ & \langle \Phi_{r1} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1} \uplus \Phi_{g1}, H_1^*, (\text{fst } (\lambda_- e_1^*, \lambda_- e_2^*)) () \rangle \rightarrow \\ & \langle \Phi_{r1} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1} \uplus \Phi_{g1}, H_1^*, \lambda_- e_1^* () \rangle \rightarrow \\ & \langle \Phi_{r1} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1} \uplus \Phi_{g1}, H_1^*, e_1^* \rangle \end{aligned}$$

and on the other side, the original configuration with H_2 steps to:

$$\langle \Phi_{r2} \uplus \text{flags}(W_1, 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H_2^*, e_1^\dagger \rangle$$

Then, since $(W_1, (\Phi_{f1}, e_1^*), (\Phi_{f2}, e_1^\dagger)) \in \mathcal{E}[\tau_1]..$, we find that the first configuration either steps to fail CONV, in which case the original expression steps to fail CONV, or steps to some irreducible

$$\langle \Phi_{r1} \uplus \Phi_{g1} \uplus \text{flags}(W_2, 1) \uplus \Phi'_{f1} \uplus \Phi'_{g1}, H_1^\dagger, e_1^{**} \rangle$$

in which case the second configuration also steps to an irreducible

$$\langle \Phi_{r2} \uplus \Phi_{g2} \uplus \text{flags}(W_2, 2) \uplus \Phi'_{f2} \uplus \Phi'_{g2}, H_2^\dagger, e_1^{\dagger\dagger} \rangle$$

and there exists some world W_2 where $W_1 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{g1}, \Phi_{r2} \uplus \Phi_{g2}} W_2$, $H_1^\dagger, H_2^\dagger : W_2$, and $(W_1, (\Phi_{f1}, e_1^{**}), (\Phi_{f2}, e_1^{\dagger\dagger})) \in \mathcal{V}[\tau_1]..$ Finally, since $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$ and $W_1 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, it follows that $W_1 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, which suffices to finish the proof. \square

Lemma B.2.41 (Compat .2).

$$\Delta; \Gamma; \textcolor{blue}{T}; \textcolor{brown}{O} \vdash e \preceq e : \tau_1 \& \tau_2 \implies \Delta; \Gamma; \textcolor{blue}{T}; \textcolor{brown}{O} \vdash e.2 \preceq e.2 : \tau_2$$

Proof. This proof is essentially identical to that of .1. \square

Lemma B.2.42 (Compat \otimes).

$$\begin{aligned} \Delta; \Gamma; \Omega_1 \vdash e_1 \preceq e_1 : \tau_1 &\wedge \Delta; \Gamma; \Omega_2 \vdash e_2 \preceq e_2 : \tau_2 \\ \implies \Delta; \Gamma; \Omega_1 \uplus \Omega_2 \vdash (e_1, e_2) \preceq (e_1, e_2) : \tau_1 \otimes \tau_2 \end{aligned}$$

Proof. Expanding the conclusion, given

$$\begin{aligned} \forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega}. \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \\ \wedge (W, \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \wedge (W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\Omega_1 \uplus \Omega_2]. \end{aligned}$$

we must show

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1((e_1, e_2)^+))), (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2((e_1, e_2)^+)))) \in \mathcal{E}[\tau_1 \otimes \tau_2].$$

Notice that both of these expressions have no free variables by Lemma B.2.16.

We can push the compiler and substitutions through the product expression and refine this to:

$$\begin{aligned} (W, (\Phi_1, (\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_1^+)))), \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e_2^+))))), \\ (\Phi_2, (\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_1^+)))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e_2^+))))) \in \mathcal{E}[\tau_1 \otimes \tau_2]. \end{aligned}$$

Next, by Lemma B.2.5, we have that $\gamma_{\Omega} = \gamma_1 \uplus \gamma_2$, $\Phi_1 = \Phi_{1l} \uplus \Phi_{1r}$, and $\Phi_2 = \Phi_{2l} \uplus \Phi_{2r}$ where

$$(W, \Phi_{1l}, \Phi_{2l}, \gamma_1) \in \mathcal{G}[\Omega_1].$$

and

$$(W, \Phi_{1r}, \Phi_{2r}, \gamma_2) \in \mathcal{G}[\Omega_2].$$

and for all $i \in \{1, 2\}$,

$$\gamma_{\Omega}^i(e_1^+) = \gamma_1^i(e_1^+)$$

and

$$\gamma_{\Omega}^i(e_2^+) = \gamma_2^i(e_2^+)$$

Thus, we must show

$$\begin{aligned} (W, (\Phi_{1l} \uplus \Phi_{1r}, (\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_1^1(e_1^+)))), \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_2^1(e_2^+))))), \\ (\Phi_{2l} \uplus \Phi_{2r}, (\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_1^2(e_1^+)))), \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_2^2(e_2^+))))) \in \mathcal{E}[\tau_1 \otimes \tau_2]. \end{aligned}$$

Let e_1 and e_2 be the first and second expressions, respectively, in the above tuple. Expanding the definition of the expression relation, given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \Phi_{1l} \uplus \Phi_{1r} \wedge \Phi_{r2} \# \Phi_{2l} \uplus \Phi_{2r} \wedge \Phi_{r1} \uplus \Phi_{1l} \uplus \Phi_{1r}, \Phi_{r2} \uplus \Phi_{2l} \uplus \Phi_{2r} : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_{1l} \uplus \Phi_{1r}, H_1, e_1 \rangle \xrightarrow{j} \langle \Phi'_1, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_{2l} \uplus \Phi_{2r}, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\tau_1 \otimes \tau_2]_\rho \end{aligned}$$

Next, we need to find e'_1 . From the operational semantic, the tensor will run the first subexpression using the heap H_1 until it reaches a target value or gets stuck. By appealing to our first induction hypothesis, instantiated with $W, \gamma_\Gamma, \gamma_\Gamma, \gamma_1, \rho$, we find that:

$$(W, (\Phi_{1l}, \gamma_\Gamma^1(\gamma_1^1(\gamma_1^1(e_1^+))))), (\Phi_{2l}, \gamma_\Gamma^2(\gamma_1^2(\gamma_1^2(e_1^+)))) \in \mathcal{E}[\tau_1].$$

Thus, the configuration

$$\langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W, 1) \uplus \Phi_{1l}, H_1, \gamma_\Gamma^1(\gamma_1^1(\gamma_1^1(e_1^+))) \rangle$$

either reduces to fail CONV, in which case the original expression steps to fail CONV, or to some irreducible configuration $\langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1l} \uplus \Phi_{g1l}, H_1^*, e_1^* \rangle$, in which case on the other side, the configuration

$$\langle \Phi_{r2} \uplus \Phi_{2r} \uplus \text{flags}(W, 2) \uplus \Phi_{2l}, H_2, \gamma_\Gamma^2(\gamma_1^2(\gamma_1^2(e_1^+))) \rangle$$

reduces to some irreducible configuration $\langle \Phi_{r2} \uplus \Phi_{2r} \uplus \text{flags}(W_1, 2) \uplus \Phi_{f2l} \uplus \Phi_{g2l}, H_2^*, e_1^\dagger \rangle$ and there exists some W_1 where $W \sqsubseteq_{\Phi_{r1} \uplus \Phi_{1r}, \Phi_{r2} \uplus \Phi_{2r}} W_1, H_1^*, H_2^* : W_1$, and $(W_1, (\Phi_{f1l}, e_1^*), (\Phi_{f2l}, e_1^\dagger)) \in \mathcal{V}[\tau_1]..$

Since terms in the value relation are target values, the original pair will continue reducing on the second subexpression according to the operational semantics. Next, we can instantiate the second induction hypothesis with $W_1, \gamma_\Gamma, \gamma_\Gamma, \gamma_2, \rho$, which we can do because $\mathcal{G}[\Gamma]_\rho, \mathcal{G}[\Gamma].., \mathcal{G}[\Omega]..$ are closed under world extension (Lemma B.2.8). Thus:

$$(W, (\Phi_{1r}, \gamma_\Gamma^1(\gamma_1^1(\gamma_2^1(\gamma_2^1(e_2^+))))), (\Phi_{2r}, \gamma_\Gamma^2(\gamma_1^2(\gamma_2^2(\gamma_2^2(e_2^+)))))) \in \mathcal{E}[\tau_2].$$

Ergo, the configuration

$$\langle \Phi_{r1} \uplus \Phi_{f1l} \uplus \Phi_{g1l} \uplus \text{flags}(W_1, 1) \uplus \Phi_{1r}, H_1^*, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_1^1(e_2^+))) \rangle$$

either reduces to fail CONV, in which case the original pair steps to fail CONV, or to some irreducible configuration $\langle \Phi_{r1} \uplus \Phi_{f1l} \uplus \Phi_{g1l} \uplus \text{flags}(W_2, 1) \uplus \Phi_{f1r} \uplus \Phi_{g1r}, H_1^\dagger, e_2^* \rangle$, in which case on the other side, the configuration

$$\langle \Phi_{r2} \uplus \Phi_{f2l} \uplus \Phi_{g2l} \uplus \text{flags}(W_1, 2) \uplus \Phi_{2r}, H_2^*, e_1^\dagger \rangle$$

reduces to some irreducible configuration $\langle \Phi_{r2} \uplus \Phi_{f2l} \uplus \Phi_{g2l} \uplus \text{flags}(W_2, 2) \uplus \Phi_{f2r} \uplus \Phi_{g2r}, H_2^\dagger, e_2^\dagger \rangle$ and there exists some W_2 where $W_1 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{f1l} \uplus \Phi_{g1l}, \Phi_{r2} \uplus \Phi_{f2l} \uplus \Phi_{g2l}} W_2$, $H_1^\dagger, H_2^\dagger : W_2$ and

$$(W_2, (\Phi_{f1r}, e_2^*), (\Phi_{f2r}, e_2^\dagger)) \in \mathcal{V}[\tau_2].$$

Thus, the original pair with H_1 steps to $\langle \Phi_{r1} \uplus \text{flags}(W_2, 1) \uplus \Phi_{f1l} \uplus \Phi_{f1r} \uplus \Phi_{g1l} \uplus \Phi_{g1r}, H_1^\dagger, (e_1^*, e_2^*) \rangle$ which is a value and thus an irreducible configuration because both e_1^* and e_2^* are values. Similarly, the original pair with H_2 steps to $\langle \Phi_{r2} \uplus \text{flags}(W_2, 2) \uplus \Phi_{f2l} \uplus \Phi_{f2r} \uplus \Phi_{g2l} \uplus \Phi_{g2r}, H_2^\dagger, (e_1^\dagger, e_2^\dagger) \rangle \not\rightsquigarrow$. Ergo, since $(W_2, (\Phi_{f1l}, e_1^*), (\Phi_{f1r}, e_1^\dagger)) \in \mathcal{V}[\tau_1]$. (because $(W_1, (\Phi_{f1l}, e_1^*), (\Phi_{f2l}, e_1^\dagger)) \in \mathcal{V}[\tau_1]$. and $W_1 \sqsubseteq_{\Phi_{f1l}, \Phi_{f2l}} W_2$) and $(W_2, (\Phi_{f1r}, e_2^*), (\Phi_{f2r}, e_2^\dagger)) \in \mathcal{V}[\tau_2]$, so $(W_2, (\Phi_{f1l} \uplus \Phi_{f1r}, (e_1^*, e_2^*)), (\Phi_{f2l} \uplus \Phi_{f2r}, (e_1^\dagger, e_2^\dagger))) \in \mathcal{V}[\tau_1 \otimes \tau_2]$. Finally, since $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$ and $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, we have $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, which suffices to finish the proof. \square

Lemma B.2.43 (Compat let).

$$\begin{aligned} & \Delta; \Gamma; \Omega_1 \vdash e_1 \preceq e_1 : \tau_1 \otimes \tau_2 \wedge \Delta; \Gamma; \Omega_2, a_\bullet : \tau_1, a'_\bullet : \tau_2 \vdash e_2 \preceq e_2 : \tau \\ \implies & \Delta; \Gamma; \Omega_1 \uplus \Omega_2 \vdash \text{let } (a_\bullet, a'_\bullet) = e_1 \text{ in } e_2 \preceq \text{let } (a_\bullet, a'_\bullet) = e_1 \text{ in } e_2 : \tau \end{aligned}$$

Proof. Expanding the conclusion, given

$$\begin{aligned} & \forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Omega}. \\ & \rho \in \mathcal{D}[\Delta] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\Omega]. \\ & \wedge (W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\Omega_1 \uplus \Omega_2]. \end{aligned}$$

we must show

$$\begin{aligned} & (W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(\text{let } (a_\bullet, a'_\bullet) = e_1 \text{ in } e_2^+)))), \\ & (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(\text{let } (a_\bullet, a'_\bullet) = e_1 \text{ in } e_2^+)))) \in \mathcal{E}[\tau]. \end{aligned}$$

By pushing the compilers and substitutions through the **let**, we can refine this to:

$$\begin{aligned} & (W, (\Phi_1, \text{let } x_{\text{fresh}} = \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\mathbf{e}_1^+)) \text{ in let } a_{\bullet} = \text{fst } x_{\text{fresh}} \\ & \quad \text{in let } a'_{\bullet} = \text{snd } x_{\text{fresh}} \text{ in } \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\mathbf{e}_2^+))), \\ & (\Phi_2, \text{let } x_{\text{fresh}} = \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\mathbf{e}_1^+)) \text{ in let } a_{\bullet} = \text{fst } x_{\text{fresh}} \\ & \quad \text{in let } a'_{\bullet} = \text{snd } x_{\text{fresh}} \text{ in } \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\mathbf{e}_2^+))) \in \mathcal{E}[\![\tau]\!]. \end{aligned}$$

Next, by Lemma B.2.5, we have that $\gamma_{\Omega} = \gamma_1 \uplus \gamma_2$, $\Phi_1 = \Phi_{1l} \uplus \Phi_{1r}$, and $\Phi_2 = \Phi_{2l} \uplus \Phi_{2r}$ where

$$(W, \Phi_{1l}, \Phi_{2l}, \gamma_1) \in \mathcal{G}[\![\Omega]\!].$$

and

$$(W, \Phi_{1r}, \Phi_{2r}, \gamma_2) \in \mathcal{G}[\![\Omega]\!].$$

and for all $i \in \{1, 2\}$,

$$\gamma_{\Omega}^i(\mathbf{e}_1^+) = \gamma_1^i(\mathbf{e}_1^+)$$

and

$$\gamma_{\Omega}^i(\mathbf{e}_2^+) = \gamma_2^i(\mathbf{e}_2^+)$$

Thus, we refine the statement we need to prove to:

$$\begin{aligned} & (W, (\Phi_{1l} \uplus \Phi_{1r}, \text{let } x_{\text{fresh}} = \gamma_{\Gamma}^1(\gamma_1^1(\mathbf{e}_1^+)) \text{ in let } a_{\bullet} = \text{fst } x_{\text{fresh}} \\ & \quad \text{in let } a'_{\bullet} = \text{snd } x_{\text{fresh}} \text{ in } \gamma_{\Gamma}^1(\gamma_2^1(\mathbf{e}_2^+))), \\ & (\Phi_{2l} \uplus \Phi_{2r}, \text{let } x_{\text{fresh}} = \gamma_{\Gamma}^2(\gamma_1^2(\mathbf{e}_1^+)) \text{ in let } a_{\bullet} = \text{fst } x_{\text{fresh}} \\ & \quad \text{in let } a'_{\bullet} = \text{snd } x_{\text{fresh}} \text{ in } \gamma_{\Gamma}^2(\gamma_2^2(\mathbf{e}_2^+))) \in \mathcal{E}[\![\tau]\!]. \end{aligned}$$

Let \mathbf{e}_1 and \mathbf{e}_2 be the first and second expressions, respectively, in the tuple above. Expanding the definition of the expression relation, given:

$$\begin{aligned} & \forall \Phi_{r1}, \Phi_{r2}, H_1, H_2 : W, e'_1, H'_1, j < W.k. \\ & \Phi_{r1} \# \Phi_{1l} \uplus \Phi_{1r} \wedge \Phi_{r2} \# \Phi_{2l} \uplus \Phi_{2r} \wedge \Phi_{r1} \uplus \Phi_{1l} \uplus \Phi_{1r}, \Phi_{r2} \uplus \Phi_{2l} \uplus \Phi_{2r} : W \wedge \\ & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_{1l} \uplus \Phi_{1r}, H_1, e_1 \rangle \xrightarrow{-j} \langle \Phi'_1, H'_1, e'_1 \rangle \rightsquigarrow \end{aligned}$$

we must show that either e'_1 is fail CONV or there exist $\Phi_{f1}, \Phi_{g1}, \Phi_{f2}, \Phi_{g2}, v_2, H'_2, W'$ such that:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \text{flags}(W, 2) \uplus \Phi_{2l} \uplus \Phi_{2r}, H_2, e_2 \rangle \xrightarrow{*} \langle \Phi_{r2} \uplus \text{flags}(W', 2) \uplus \Phi_{f2} \uplus \Phi_{g2}, H'_2, v_2 \rangle \rightsquigarrow \\ & \wedge \Phi'_1 = \Phi_{r1} \uplus \text{flags}(W', 1) \uplus \Phi_{f1} \uplus \Phi_{g1} \wedge \\ & \wedge W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W' \wedge H'_1, H'_2 : W' \\ & \wedge (W', (\Phi_{f1}, e'_1), (\Phi_{f2}, v_2)) \in \mathcal{V}[\![\tau]\!]_{\rho} \end{aligned}$$

Therefore, we find that

$$\langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W, 1) \uplus \Phi_{1l}, H_1, \gamma_{\Gamma}^1(\gamma_1^1(\mathbf{e}_1^+)) \rangle$$

either reduces to fail CONV, in which case the original expression steps to fail CONV, or to some irreducible configuration $\langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1l} \uplus \Phi_{g1l}, H_1^*, e_1^* \rangle$, in which case

$$\langle \Phi_{r2} \uplus \Phi_{2r} \uplus \text{flags}(W, 2) \uplus \Phi_{2l}, H_2, \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_1^2(e_1^+))) \rangle$$

also reduces to an irreducible configuration $\langle \Phi_{r2} \uplus \Phi_{2r} \uplus \text{flags}(W_1, 2) \uplus \Phi_{f2l} \uplus \Phi_{g2l}, H_2^*, e_1^\dagger \rangle$ and there exists some world W_1 where $W \sqsubseteq_{\Phi_{r1} \uplus \Phi_{1r}, \Phi_{r2} \uplus \Phi_{2r}} W_1$, $H_1^*, H_2^* : W_1$, and $(W_1, (\Phi_{f1l}, e_1^*), (\Phi_{f2l}, e_1^\dagger)) \in \mathcal{V}[\![\tau_1 \otimes \tau_2]\!]$.

By expanding the value relation, we find that

$$\Phi_{f1l} = \Phi_{f1ll} \uplus \Phi_{f1lr}$$

$$e_1^* = (v_1^*, v_2^*)$$

$$\Phi_{f2l} = \Phi_{f2ll} \uplus \Phi_{f2lr}$$

$$e_1^\dagger = (v_1^\dagger, v_2^\dagger)$$

where

$$(W_1, (\Phi_{f1ll}, v_1^*), (\Phi_{f2ll}, v_1^\dagger)) \in \mathcal{V}[\![\tau_1]\!].$$

$$(W_1, (\Phi_{f1lr}, v_2^*), (\Phi_{f2lr}, v_2^\dagger)) \in \mathcal{V}[\![\tau_2]\!].$$

Thus, the original configuration with H_1 steps as follows:

$$\begin{aligned} & \langle \Phi_{r1} \uplus \text{flags}(W, 1) \uplus \Phi_{1l} \uplus \Phi_{1r}, H_1, \text{let } x_{\text{fresh}} = \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_1^1(e_1^+))) \text{ in} \\ & \quad \text{let } a_\bullet = \text{fst } x_{\text{fresh}} \text{ in let } a'_\bullet = \text{snd } x_{\text{fresh}} \text{ in } \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_2^+))) \rangle \xrightarrow{*} \\ & \langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1ll} \uplus \Phi_{f1lr} \uplus \Phi_{g1l}, H_1^*, \text{let } x_{\text{fresh}} = (v_1^*, v_2^*) \text{ in} \\ & \quad \text{let } a_\bullet = \text{fst } x_{\text{fresh}} \text{ in let } a'_\bullet = \text{snd } x_{\text{fresh}} \text{ in } \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_2^+))) \rangle \xrightarrow{*} \\ & \langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1ll} \uplus \Phi_{f1lr} \uplus \Phi_{g1l} \uplus \{f_{1l}, f_{2l}\}, H_1^*, \\ & \quad [a_\bullet \mapsto \text{protect}(v_1^*, f_{1l})][a'_\bullet \mapsto \text{protect}(v_2^*, f_{2l})] \gamma_{\Gamma}^1(\gamma_{\Gamma}^1(\gamma_2^1(e_2^+))) \rangle \end{aligned}$$

where $f_{1l} \neq f_{2l}$ and $f_{1l}, f_{2l} \notin \Phi_{r1} \uplus \Phi_{1r} \uplus \Phi_{f1ll} \uplus \Phi_{f1lr} \uplus \Phi_{g1l}$.

By similar reasoning, the configuration on the other side with H_2 steps to:

$$\begin{aligned} & \langle \Phi_{r2} \uplus \Phi_{2r} \uplus \text{flags}(W_1, 2) \uplus \Phi_{f2ll} \uplus \Phi_{f2lr} \uplus \Phi_{g2l} \uplus \{f_{1r}, f_{2r}\}, H_2^*, \\ & \quad [a_\bullet \mapsto \text{protect}(v_1^\dagger, f_{1r})][a'_\bullet \mapsto \text{protect}(v_2^\dagger, f_{2r})] \gamma_{\Gamma}^2(\gamma_{\Gamma}^2(\gamma_2^2(e_2^+))) \rangle \end{aligned}$$

where $f_{1r} \neq f_{2r}$ and $f_{1r}, f_{2r} \notin \Phi_{r2} \uplus \Phi_{2r} \uplus \Phi_{f2ll} \uplus \Phi_{f2lr} \uplus \Phi_{g2l}$.

Next, notice that:

$$\begin{aligned} & (W_1, \Phi_{1r} \uplus \Phi_{f1ll} \uplus \Phi_{f1lr} \uplus \{f_{1l}, f_{2l}\}, \Phi_{2r} \uplus \Phi_{f2ll} \uplus \Phi_{f2lr} \uplus \{f_{1r}, f_{2r}\}, \\ & \quad \gamma_2[a_\bullet \mapsto (\text{protect}(v_1^*, f_{1l}), \text{protect}(v_1^\dagger, f_{1r}))][a'_\bullet \mapsto (\text{protect}(v_2^*, f_{2l}), \text{protect}(v_2^\dagger, f_{2r}))]) \\ & \in \mathcal{G}[\![\Omega_2, a_\bullet : \tau_1, a'_\bullet : \tau_2]\!]. \end{aligned}$$

Thus, we can instantiate the second induction hypothesis with

$$W_1, \gamma_{\Gamma}, \gamma_{\Omega}, \\ \gamma_2[a_{\bullet} \mapsto (\text{protect}(v_1^*, f_{1l}), \text{protect}(v_1^{\dagger}, f_{1r}))][a'_{\bullet} \mapsto (\text{protect}(v_2^*, f_{2l}), \text{protect}(v_2^{\dagger}, f_{2r}))], \rho$$

to find that:

$$(W_1, \\ (\Phi_{1r} \uplus \Phi_{f1ll} \uplus \Phi_{f1lr} \uplus \{f_{1l}, f_{2l}\}, \\ [a_{\bullet} \mapsto \text{protect}(v_1^*, f_{1l})][a'_{\bullet} \mapsto \text{protect}(v_2^*, f_{2l})]\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_2^1(e_2^+)))), \\ (\Phi_{2r} \uplus \Phi_{f2ll} \uplus \Phi_{f2lr} \uplus \{f_{1r}, f_{2r}\}, \\ [a_{\bullet} \mapsto \text{protect}(v_1^{\dagger}, f_{1r})][a'_{\bullet} \mapsto \text{protect}(v_2^{\dagger}, f_{2r})]\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_2^2(e_2^+)))) \in \mathcal{E}[\![\tau]\!].$$

Then, consider again the above configurations:

$$\langle \Phi_{r1} \uplus \Phi_{1r} \uplus \text{flags}(W_1, 1) \uplus \Phi_{f1ll} \uplus \Phi_{f1lr} \uplus \Phi_{g1l} \uplus \{f_{1l}, f_{2l}\}, H_1^*, \\ [a_{\bullet} \mapsto \text{protect}(v_1^*, f_{1l})][a'_{\bullet} \mapsto \text{protect}(v_2^*, f_{2l})]\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_2^1(e_2^+))) \rangle$$

$$\langle \Phi_{r2} \uplus \Phi_{2r} \uplus \text{flags}(W_1, 2) \uplus \Phi_{f2ll} \uplus \Phi_{f2lr} \uplus \Phi_{g2l} \uplus \{f_{1r}, f_{2r}\}, H_2^*, \\ [a_{\bullet} \mapsto \text{protect}(v_1^{\dagger}, f_{1r})][a'_{\bullet} \mapsto \text{protect}(v_2^{\dagger}, f_{2r})]\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_2^2(e_2^+))) \rangle$$

By applying the above fact, we find that the first configuration either steps to fail CONV, in which case the original configuration steps to fail CONV, or steps to some irreducible configuration

$$\langle \Phi_{r1} \uplus \Phi_{g1l} \uplus \text{flags}(W_2, 1) \uplus \Phi_{f1r} \uplus \Phi_{g1r}, H_1^{\dagger}, e_2^* \rangle$$

in which case the second configuration also steps to some irreducible configuration

$$\langle \Phi_{r2} \uplus \Phi_{g2l} \uplus \text{flags}(W_2, 2) \uplus \Phi_{f2r} \uplus \Phi_{g2r}, H_2^{\dagger}, e_2^{\dagger} \rangle$$

and there exists some world W_2 such that $W_1 \sqsubseteq_{\Phi_{r1} \uplus \Phi_{g1l}, \Phi_{r2} \uplus \Phi_{g2l}} W_2, H_1^{\dagger}, H_2^{\dagger} : W_2$, and

$$(W_2, (\Phi_{f1r}, e_2^*), (\Phi_{f2r}, e_2^{\dagger})) \in \mathcal{V}[\![\tau]\!].$$

Finally, since $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_1$ and $W_1 \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, we have $W \sqsubseteq_{\Phi_{r1}, \Phi_{r2}} W_2$, which suffices to finish the proof. \square

Lemma B.2.44 (Compat $(e)\tau$).

$$\begin{aligned} & \Gamma; \Omega; \Delta; \Gamma \vdash e \preceq e : \tau \wedge _ : \tau \sim \tau \\ \implies & \Delta; \Gamma; \Omega \vdash (e)\tau \preceq (e)\tau : \tau \end{aligned}$$

Proof. Expanding the conclusion, given

$$\begin{aligned} & \forall W. \forall \rho. \gamma_{\Gamma} \gamma_{\Omega} \gamma_{\Delta}. \\ & \rho \in \mathcal{D}[\![\Delta]\!] \wedge (W, \emptyset, \emptyset, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]_{\rho} \wedge (W, \emptyset, \emptyset, \gamma_{\Omega}) \in \mathcal{G}[\![\Omega]\!] \wedge (W, \Phi_1, \Phi_2, \gamma_{\Omega}) \in \mathcal{G}[\![\Omega]\!]. \end{aligned}$$

we must show

$$(W, (\Phi_1, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\langle e \rangle_{\tau}^+))), (\Phi_2, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\langle e \rangle_{\tau}^+)))) \in \mathcal{E}[\tau].$$

We can push the compiler and substitutions through the pair to refine that to:

$$(W, (\Phi_1, C_{\tau \mapsto \tau}(\gamma_{\Gamma}^1(\gamma_{\Omega}^1(e^+)))), (\Phi_2, C_{\tau \mapsto \tau}(\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))))) \in \mathcal{E}[\tau].$$

By Lemma B.2.3, it suffices to show:

$$(W, (\emptyset, C_{\tau \mapsto \tau}(\gamma_{\Gamma}^1(\gamma_{\Omega}^1(e^+)))), (\emptyset, C_{\tau \mapsto \tau}(\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))))) \in \mathcal{E}[\tau].$$

Now, by instantiating our induction hypothesis with $W, \gamma_{\Gamma}, \gamma_{\Omega}, \rho$, we find that:

$$(W, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+)))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))))) \in \mathcal{E}[\tau]_{\rho}$$

By Lemma B.2.14, it follows that:

$$(W, (\emptyset, \gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+)))), (\emptyset, \gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))))) \in \mathcal{E}[\tau].$$

Therefore, by Theorem B.2.18, we have

$$\begin{aligned} & (W, (\emptyset, C_{\tau \mapsto \tau}(\gamma_{\Gamma}^1(\gamma_{\Omega}^1(\gamma_{\Omega}^1(e^+))))), \\ & (\emptyset, C_{\tau \mapsto \tau}(\gamma_{\Gamma}^2(\gamma_{\Omega}^2(\gamma_{\Omega}^2(e^+)))))) \in \mathcal{E}[\tau]. \end{aligned}$$

as was to be proven. \square

C

VALUE INTEROPERABILITY: MEMORY MANAGEMENT AND POLYMORPHISM

Lemma C.0.1 (World Extension Weakening). *If $W \sqsubseteq_{\mathbb{L}, \eta} W'$, then for any \mathbb{L}' such that $\mathbb{L}' \cdot j \subseteq \mathbb{L} \cdot j$ for all $j \in \{1, 2\}$ and for any $\eta' \subseteq \eta$, $W \sqsubseteq_{\mathbb{L}', \eta'} W'$.*

Proof. Let $W = (k, \Psi)$ and $W' = (j, \Psi')$. From $W \sqsubseteq_{\mathbb{L}, \eta} W'$, we have $j \leq k$. We also have $\mathbb{L}.1 \# \text{dom}((\Psi')^1)$ and $\mathbb{L}.2 \# \text{dom}((\Psi')^2)$. Since $\mathbb{L}' \cdot 1 \subseteq \mathbb{L}.1$ and $\mathbb{L}' \cdot 2 \subseteq \mathbb{L}.2$, this implies $\mathbb{L}' \cdot 1 \# \text{dom}((\Psi')^1)$ and $\mathbb{L}' \cdot 2 \# \text{dom}((\Psi')^2)$. Moreover, for all $(\ell_1, \ell_2) \in \eta$, $\Psi'(\ell_1, \ell_2) = \lfloor \Psi(\ell_1, \ell_2) \rfloor_j$. Since $\eta' \subseteq \eta$, it follows that for all $(\ell_1, \ell_2) \in \eta'$, $\Psi'(\ell_1, \ell_2) = \lfloor \Psi(\ell_1, \ell_2) \rfloor_j$. Ergo, $W \sqsubseteq_{\mathbb{L}', \eta'} W'$, as was to be proven. \square

Lemma C.0.2 (World Extension Transitive). *If $W_1 \sqsubseteq_{\mathbb{L}_1, \eta_1} W_2$ and $W_2 \sqsubseteq_{\mathbb{L}_2, \eta_2} W_3$ then*

$$W_1 \sqsubseteq_{(\mathbb{L}_1.1 \cap \mathbb{L}_2.1, \mathbb{L}_1.2 \cap \mathbb{L}_2.2), \eta_1 \cap \eta_2} W_3$$

Proof. Let $\mathbb{L} = (\mathbb{L}_1.1 \cap \mathbb{L}_2.1, \mathbb{L}_1.2 \cap \mathbb{L}_2.2)$ and $\eta = \eta_1 \cap \eta_2$. By Lemma C.0.1, $W_1 \sqsubseteq_{\mathbb{L}, \eta} W_2$ and $W_2 \sqsubseteq_{\mathbb{L}, \eta} W_3$. We would like to show $W_1 \sqsubseteq_{\mathbb{L}, \eta} W_3$.

Let $W_1 = (k_1, \Psi_1)$, $W_2 = (k_2, \Psi_2)$, and $W_3 = (k_3, \Psi_3)$.

We know from world extension that $k_1 \leq k_2$ and $k_2 \leq k_3$, so by transitivity, $k_1 \leq k_3$.

By $W_2 \sqsubseteq_{\mathbb{L}, \eta} W_3$, $\mathbb{L}.1 \# \text{dom}(\Psi_3^1)$ and $\mathbb{L}.2 \# \text{dom}(\Psi_3^2)$.

Finally, by both world extensions, for all $(\ell_1, \ell_2) \in \eta$,

$$\Psi_3(\ell_1, \ell_2) = \lfloor \Psi_2(\ell_1, \ell_2) \rfloor_{k_3} = \lfloor \lfloor \Psi_1(\ell_1, \ell_2) \rfloor_{k_2} \rfloor_{k_3}$$

Then, since $k_2 \leq k_3$, we find that $\Psi_3(\ell_1, \ell_2) = \lfloor \lfloor \Psi_1(\ell_1, \ell_2) \rfloor_{k_2} \rfloor_{k_3} = \lfloor \Psi_1(\ell_1, \ell_2) \rfloor_{k_3}$. This suffices to show that $W_1 \sqsubseteq_{\mathbb{L}, \eta} W_3$, as was to be proven. \square

Lemma C.0.3 (World Extension).

1. If $(W, (\mathsf{H}_1, v_1), (\mathsf{H}_2, v_2)) \in \mathcal{V}[\tau]_\rho$, and $W \sqsubseteq_{\mathsf{H}_1, \mathsf{H}_2, v_1, v_2} W'$, then $(W', (\mathsf{H}_1, v_1), (\mathsf{H}_2, v_2)) \in \mathcal{V}[\tau]_\rho$.
2. If $(W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}} \cdot \Gamma) \in \mathcal{G}[\Gamma]_\rho$ and $W \sqsubseteq_{\mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}} \cdot \Gamma^1(\cdot), \gamma_{\mathbf{L}} \cdot \Gamma^2(\cdot)} W'$, then $(W', \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}} \cdot \Gamma) \in \mathcal{G}[\Gamma]_\rho$.
3. If $(W, \gamma_{\mathbf{T}}) \in \mathcal{G}[\Gamma]_\rho$ and $W \sqsubseteq_{\emptyset, \emptyset, \gamma_{\mathbf{T}}^1(\cdot), \gamma_{\mathbf{T}}^2(\cdot)} W'$, then $(W', \gamma_{\mathbf{T}}) \in \mathcal{G}[\Gamma]_\rho$.

Proof. 1. By induction on τ . Most cases are trivial, relying on Lemma C.0.2 where appropriate. The only non-trivial cases are $\tau = \text{ref } \tau$ and $\tau = \text{cap } \zeta \tau$.

- $\tau = \text{ref } \tau$: Suppose that $(W, (\emptyset, \ell_1), (\emptyset, \ell_2)) \in \mathcal{V}[\text{ref } \tau]_\rho$ and $W \sqsubseteq_{\emptyset, \emptyset, \ell_1, \ell_2} W'$. We would like to show $(W', (\emptyset, \ell_1), (\emptyset, \ell_2)) \in \mathcal{V}[\text{ref } \tau]_\rho$. Expanding the premise, we have that $W.\Psi(\ell_1, \ell_2) = [\mathcal{V}[\tau]_\rho]_{W.k}$. This shows that $(\ell_1, \ell_2) \in \text{dom}(W.\Psi)$, so since ℓ_1 is free in the expression ℓ_1 and ℓ_2 is free in the expression ℓ_2 , it follows that $(\ell_1, \ell_2) \in \text{rchgclocs}(W, \text{FL}(\ell_1), \text{FL}(\ell_2))$. Ergo, by the definition of world extension,

$$W'.\Psi(\ell_1, \ell_2) = [W.\Psi(\ell_1, \ell_2)]_{W'.k} = [[\mathcal{V}[\tau]_\rho]_{W.k}]_{W'.k} = [\mathcal{V}[\tau]_\rho]_{W'.k},$$

which suffices to prove $(W', (\emptyset, \ell_1), (\emptyset, \ell_2)) \in \mathcal{V}[\text{ref } \tau]_\rho$.

(Note that $[[\mathcal{V}[\tau]_\rho]_{W.k}]_{W'.k} = [\mathcal{V}[\tau]_\rho]_{W'.k}$ follows from $W'.k \leq W.k$, which we get from world extension.)

- $\tau = \text{cap } \zeta \tau$: Suppose that $(W, (\mathbf{H}_1 \uplus \{\ell_1 \mapsto v_1\}, ()), (\mathbf{H}_2 \uplus \{\ell_2 \mapsto v_2\})) \in \mathcal{V}[\text{cap } \zeta \tau]_\rho$ where $\rho.\mathbf{L3}(\zeta) = (\ell_1, \ell_2)$ and $W \sqsubseteq_{\mathbf{H}_1 \uplus \{\ell_1 \mapsto v_1\}, \mathbf{H}_2 \uplus \{\ell_2 \mapsto v_2\}, (), ()} W'$. Expanding the definition of world extension, we find

$$W \sqsubseteq_{(\text{dom}(\mathbf{H}_1) \uplus \{\ell_1\}, \text{dom}(\mathbf{H}_2) \uplus \{\ell_2\}), \text{rchgclocs}(W, \text{FL}(\text{cod}(\mathbf{H}_1)) \cup \text{FL}(v_1), \text{FL}(\text{cod}(\mathbf{H}_2)) \cup \text{FL}(v_2))} W'$$

Thus, for $j \in \{1, 2\}$, $(\text{dom}(\mathbf{H}_j) \uplus \{\ell_j\}) \# \text{dom}(W'.\Psi^j)$, so $(W', (\mathbf{H}_1 \uplus \{\ell_1 \mapsto v_1\}, (), (\mathbf{H}_2 \uplus \{\ell_2 \mapsto v_2\}, ()))$ is still in *Atom*, which is required to show this tuple is in the value relation.

Moreover, by Lemma C.0.1, we find $W \sqsubseteq_{\mathbf{H}_1, \mathbf{H}_2, v_1, v_2} W'$.

By expanding the value relation, we find $(W, (\mathbf{H}_1, v_1), (\mathbf{H}_2, v_2)) \in \mathcal{V}[\tau]_\rho$. Since $W \sqsubseteq_{\mathbf{H}_1, \mathbf{H}_2, v_1, v_2} W'$, by the induction hypothesis, we find $(W', (\mathbf{H}_1, v_1), (\mathbf{H}_2, v_2)) \in \mathcal{V}[\tau]_\rho$, which suffices to prove $(W', (\mathbf{H}_1 \uplus \{\ell_1 \mapsto v_1\}, (), (\mathbf{H}_2 \uplus \{\ell_2 \mapsto v_2\}, ())) \in \mathcal{V}[\text{cap } \zeta \tau]_\rho$.

2. By induction on $\gamma_{\mathbf{L}}.\Gamma$, appealing to the previous case where appropriate.
3. By induction on $\gamma_{\mathbf{T}}$, appealing to the previous case where appropriate.

□

Lemma C.0.4 (World Extension and Garbage Collection). *Consider some world W and two sets of locations L_1, L_2 . Then, consider arbitrary heaps $\mathbf{H}_{1g}, \mathbf{H}_{2g} : W$ and $\mathbf{H}_{1m}, \mathbf{H}_{2m}$ such that $\mathbf{H}_{1m} : M\text{Heap}$, $\mathbf{H}_{2m} : M\text{Heap}$, $\text{dom}(\mathbf{H}_{1m}) \# \text{dom}(W.\Psi^1)$, and $\text{dom}(\mathbf{H}_{2m}) \# \text{dom}(W.\Psi^2)$. Let $L'_1 = \text{reachablelocs}(\mathbf{H}_{1g} \uplus \mathbf{H}_{1m}, \text{dom}(\mathbf{H}_{1m}) \uplus \text{FL}(\mathbf{K}_1[\cdot]) \cup L_1)$ and $L'_2 = \text{reachablelocs}(\mathbf{H}_{2g} \uplus \mathbf{H}_{2m}, \text{dom}(\mathbf{H}_{2m}) \uplus \text{FL}(\mathbf{K}_2[\cdot]) \cup L_2)$.*

Then, if

$$(\mathsf{H}_{1g} \uplus \mathsf{H}_{1m}, \mathsf{K}_1[\mathsf{callgc}]) \rightarrow_{L_1} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1m}, \mathsf{K}_1[()])$$

and

$$(\mathsf{H}_{2g} \uplus \mathsf{H}_{2m}, \mathsf{K}_2[\mathsf{callgc}]) \rightarrow_{L_2} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2m}, \mathsf{K}_2[()])$$

where $\mathsf{H}'_{1g} : \mathit{GCH} \mathit{Heap}$, $\mathsf{H}'_{2g} : \mathit{GCH} \mathit{Heap}$, then there exists some world W' such that $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ and

$$W \sqsubseteq_{(dom(\mathsf{H}_{1m}), dom(\mathsf{H}_{2m})), rchgclocs(W, L'_1, L'_2)} W' \quad (16)$$

Note: Remember that for all H, L , $L \subseteq \text{reachablelocs}(\mathsf{H}, L)$ and $FL(\text{cod}(\mathsf{H})) \subseteq \text{reachablelocs}(\mathsf{H}, L)$. Ergo, $FL(\text{cod}(\mathsf{H}_{im})) \cup FL(\mathsf{K}_j[\cdot]) \cup L_j \subseteq L'_j$ for $j \in \{1, 2\}$, which implies

$$\begin{aligned} & rchgclocs(W, FL(\text{cod}(\mathsf{H}_{1m})) \cup FL(\mathsf{K}_1[\cdot]) \cup L_1, FL(\text{cod}(\mathsf{H}_{2m})) \\ & \cup FL(\mathsf{K}_2[\cdot]) \cup L_2) \subseteq rchgclocs(W, L'_1, L'_2) \end{aligned}$$

so by Lemma C.0.1, it follows that

$$W \sqsubseteq_{(dom(\mathsf{H}_{1m}), dom(\mathsf{H}_{2m})), rchgclocs(W, FL(\text{cod}(\mathsf{H}_{1m})) \cup FL(\mathsf{K}_1[\cdot]) \cup L_1, FL(\text{cod}(\mathsf{H}_{2m})) \cup FL(\mathsf{K}_2[\cdot]) \cup L_2)} W'$$

Proof. Let $W' = (W.k, \Psi')$ where Ψ' is the subset of Ψ restricted to $\text{rchgclocs}(W, L'_1, L'_2)$.

First, it is clear that $W'.k \leq W.k$.

Second, since $W'.\Psi \subseteq W.\Psi$, $\text{dom}(\mathsf{H}_{1m}) \# \text{dom}(W.\Psi^1)$, and $\text{dom}(\mathsf{H}_{2m}) \# \text{dom}(W.\Psi^2)$, we find that $\text{dom}(\mathsf{H}_{1m}) \# \text{dom}(W'.\Psi^1)$ and $\text{dom}(\mathsf{H}_{2m}) \# \text{dom}(W'.\Psi^2)$.

Finally, to finish showing (16), we need to show that for all $(\ell_1, \ell_2) \in \text{rchgclocs}(W, L'_1, L'_2)$,

$$W'.\Psi(\ell_1, \ell_2) = \lfloor W.\Psi(\ell_1, \ell_2) \rfloor_{W'.k}$$

If $(\ell_1, \ell_2) \in \text{rchgclocs}(W, L'_1, L'_2)$, then by the definition of Ψ' above, $W'.\Psi(\ell_1, \ell_2) = W.\Psi(\ell_1, \ell_2)$. Thus, since $W'.k = W.k$,

$$W'.\Psi(\ell_1, \ell_2) = W.\Psi(\ell_1, \ell_2) = \lfloor W.\Psi(\ell_1, \ell_2) \rfloor_{W.k} = \lfloor W.\Psi(\ell_1, \ell_2) \rfloor_{W'.k}$$

as was to be demonstrated.

Next, we must show that $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$. First, since $\mathsf{H}'_{1g} \subseteq \mathsf{H}_{1g}$ and $\mathsf{H}'_{2g} \subseteq \mathsf{H}_{2g}$, it follows that $\mathsf{H}'_{1g} : \mathit{GCH} \mathit{Heap}$ and $\mathsf{H}'_{2g} : \mathit{GCH} \mathit{Heap}$.

Next, we must show that for all $(\ell_1, \ell_2) \in \text{dom}(W'.\Psi)$, we must show $\ell_1 \in \text{dom}(\mathsf{H}'_{1g})$, $\ell_2 \in \text{dom}(\mathsf{H}'_{2g})$, and

$$(\triangleright W', (\emptyset, \mathsf{H}'_{1g}(\ell_1)), (\emptyset, \mathsf{H}'_{2g}(\ell_2))) \in W'.\Psi(\ell_1, \ell_2) \quad (17)$$

By definition, $\text{dom}(W'.\Psi) = \text{rchgclocs}(W, L'_1, L'_2)$, so if $(\ell_1, \ell_2) \in \text{dom}(W'.\Psi)$, then $(\ell_1, \ell_2) \in \text{dom}(W.\Psi)$, $\ell_1 \in L'_1$ and $\ell_2 \in L'_2$. Since $(\ell_1, \ell_2) \in \text{dom}(W.\Psi)$ and $H_{1g}, H_{2g} : W$, we find that $\ell_1 \in \text{dom}(H_{1g})$, $\ell_2 \in \text{dom}(H_{2g})$, and $(\triangleright W, (\emptyset, H_{1g}(\ell_1)), (\emptyset, H_{2g}(\ell_2))) \in W.\Psi(\ell_1, \ell_2)$.

Then, since $(\ell_1, \ell_2) \in \text{dom}(W'.\Psi)$, $W'.\Psi(\ell_1, \ell_2) = W.\Psi(\ell_1, \ell_2)$. Moreover, by the operational semantics of `callgc`, $L'_1 \cap \text{dom}(H_{1g}) \subseteq \text{dom}(H'_{1g})$, so $\ell_1 \in \text{dom}(H'_{1g})$ and $H'_{1g}(\ell_1) = H_{1g}(\ell_1)$. By similar reasoning, $\ell_2 \in \text{dom}(H'_{2g})$ and $H'_{2g}(\ell_2) = H_{2g}(\ell_2)$. Thus, we deduce that

$$(\triangleright W, (\emptyset, H'_{1g}(\ell_1)), (\emptyset, H'_{2g}(\ell_2))) \in W'.\Psi(\ell_1, \ell_2) \quad (18)$$

Next, notice that, by the definition of `reachablelocs`, since $\ell_1 \in L'_1$, it follows that $FL(H'_{1g}(\ell_1)) = FL(H_{1g}(\ell_1)) \subseteq L'_1$. By similar reasoning, $FL(H'_{2g}(\ell_2)) \subseteq L'_2$. Ergo,

$$\text{rchgclocs}(W, FL(H'_{1g}(\ell_1)), FL(H'_{2g}(\ell_2))) \subseteq \text{rchgclocs}(W, L'_1, L'_2)$$

By Lemma C.0.1, we then have

$$W \sqsubseteq_{(\emptyset, \emptyset), \text{rchgclocs}(W, FL(H'_{1g}(\ell_1)), FL(H'_{2g}(\ell_2)))} W'$$

so it follows that

$$\triangleright W \sqsubseteq_{(\emptyset, \emptyset), \text{rchgclocs}(W, FL(H'_{1g}(\ell_1)), FL(H'_{2g}(\ell_2)))} \triangleright W'$$

In other words, $\triangleright W \sqsubseteq_{\emptyset, \emptyset, H'_{1g}(\ell_1), H'_{2g}(\ell_2)} \triangleright W'$. Ergo, by (18) and the fact that $W'.\Psi(\ell_1, \ell_2) \in \text{Typ}_{W.k}$ to deduce (17), as was to be proven. \square

Lemma C.0.5 (Compositionality). *If $\Delta \vdash \tau_1$ and $\Delta, \alpha \vdash \tau_2$ and $\rho \in \mathcal{D}[\Delta]$, then*

$$\mathcal{V}[[\alpha \mapsto \tau_1]\tau_2]_\rho = \mathcal{V}[[\tau_2]_{\rho[\mathbf{F}(\alpha) \mapsto \mathcal{V}[\tau_1]_\rho]}]$$

Proof. By induction on τ_2 . We show the interesting cases:

CASE $\tau_2 = \alpha$.

$$\begin{aligned} \mathcal{V}[[\alpha \mapsto \tau_1]\alpha]_\rho &= \mathcal{V}[[\tau_1]_\rho] \quad (\text{by sub}) \\ &= \rho[\mathbf{F}(\alpha) \mapsto \mathcal{V}[\tau_1]_\rho].\mathbf{F}(\alpha) \quad (\text{by lookup}) \\ &= \mathcal{V}[[\alpha]_{\rho[\mathbf{F}(\alpha) \mapsto \mathcal{V}[\tau_1]_\rho]}} \quad (\text{by def } \mathcal{V}[\cdot].) \end{aligned}$$

CASE $\tau_2 = \beta \neq \alpha$.

$$\begin{aligned} \mathcal{V}[[\alpha \mapsto \tau_1]\beta]_\rho &= \mathcal{V}[[\beta]_\rho] \quad (\text{by sub}) \\ &= \rho.\mathbf{F}(\beta) \quad (\text{by def } \mathcal{V}[\cdot].) \\ &= \rho[\mathbf{F}(\alpha) \mapsto \mathcal{V}[\tau_1]_\rho].\mathbf{F}(\beta) \quad (\text{by lookup}) \\ &= \mathcal{V}[[\beta]_{\rho[\mathbf{F}(\alpha) \mapsto \mathcal{V}[\tau_1]_\rho]}} \quad (\text{by def } \mathcal{V}[\cdot].) \end{aligned}$$

The other cases are straightforward by expanding the definitions of $\mathcal{V}[\cdot], \mathcal{E}[\cdot]$, and applying the induction hypotheses. \square

Lemma C.0.6 (L³ Compositionality). *If $\Delta, \zeta \vdash \tau$, $\rho \in \mathcal{D}[\Delta]$, and $\rho(\zeta') = (\ell'_1, \ell'_2)$, then*

$$\mathcal{V}[[\zeta \mapsto \zeta']\tau]_\rho = \mathcal{V}[\tau]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)]}$$

Proof. By induction on τ . We show the interesting cases:

CASE $\tau = \mathbf{ptr} \zeta$.

$$\begin{aligned} & \mathcal{V}[[\zeta \mapsto \zeta']\mathbf{ptr} \zeta]_\rho \\ &= \mathcal{V}[\mathbf{ptr} \zeta']_\rho && \text{(by sub)} \\ &= \{(W, (\emptyset, \ell_1), (\emptyset, \ell_2)) \mid \rho.\mathbf{L3}(\zeta') = (\ell_1, \ell_2)\} && \text{(by def)} \\ &= \{(W, (\emptyset, \ell_1), (\emptyset, \ell_2)) \mid (\ell'_1, \ell'_2) = (\ell_1, \ell_2)\} && \text{(by assumption)} \\ &= \{(W, (\emptyset, \ell_1), (\emptyset, \ell_2)) \mid \rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)].\mathbf{L3}(\zeta) = (\ell_1, \ell_2)\} && \text{(by lookup)} \\ &= \mathcal{V}[\mathbf{ptr} \zeta]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)]} && \text{(by def)} \end{aligned}$$

CASE $\tau = \mathbf{ptr} \zeta_2$ WHERE $\zeta_2 \neq \zeta$.

$$\begin{aligned} & \mathcal{V}[[\zeta \mapsto \zeta']\mathbf{ptr} \zeta_2]_\rho \\ &= \mathcal{V}[\mathbf{ptr} \zeta_2]_\rho && \text{(by sub)} \\ &= \{(W, (\emptyset, \ell_1), (\emptyset, \ell_2)) \mid \rho.\mathbf{L3}(\zeta_2) = (\ell_1, \ell_2)\} && \text{(by def)} \\ &= \{(W, (\emptyset, \ell_1), (\emptyset, \ell_2)) \mid \rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)].\mathbf{L3}(\zeta_2) = (\ell_1, \ell_2)\} && \text{(by lookup)} \\ &= \mathcal{V}[\mathbf{ptr} \zeta_2]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)]} && \text{(by def)} \end{aligned}$$

CASE $\tau = \mathbf{cap} \zeta \tau_2$.

$$\begin{aligned} & \mathcal{V}[[\zeta \mapsto \zeta']\mathbf{cap} \zeta \tau_2]_\rho \\ &= \mathcal{V}[\mathbf{cap} \zeta' [\zeta \mapsto \zeta']\tau_2]_\rho && \text{(by sub)} \\ &= \{(W, (\mathsf{H}_1 \uplus \{\ell_1 \mapsto v_1\}, ()), (\mathsf{H}_2 \uplus \{\ell_2 \mapsto v_2\}, ()) \mid \\ &\quad \rho.\mathbf{L3}(\zeta') = (\ell_1, \ell_2) \wedge (W, (\mathsf{H}_1, v_1), (\mathsf{H}_2, v_2)) \in \mathcal{V}[[\zeta \mapsto \zeta']\tau_2]_\rho\} && \text{(by def)} \\ &= \{(W, (\mathsf{H}_1 \uplus \{\ell_1 \mapsto v_1\}, ()), (\mathsf{H}_2 \uplus \{\ell_2 \mapsto v_2\}, ()) \mid \\ &\quad (\ell'_1, \ell'_2) = (\ell_1, \ell_2) \wedge (W, (\mathsf{H}_1, v_1), (\mathsf{H}_2, v_2)) \in \mathcal{V}[[\zeta \mapsto \zeta']\tau_2]_\rho\} && \text{(by assumption)} \\ &= \{(W, (\mathsf{H}_1 \uplus \{\ell_1 \mapsto v_1\}, ()), (\mathsf{H}_2 \uplus \{\ell_2 \mapsto v_2\}, ()) \mid \\ &\quad \rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)].\mathbf{L3}(\zeta) = (\ell_1, \ell_2) \wedge \\ &\quad (W, (\mathsf{H}_1, v_1), (\mathsf{H}_2, v_2)) \in \mathcal{V}[[\zeta \mapsto \zeta']\tau_2]_\rho\} && \text{(by lookup)} \\ &= \{(W, (\mathsf{H}_1 \uplus \{\ell_1 \mapsto v_1\}, ()), (\mathsf{H}_2 \uplus \{\ell_2 \mapsto v_2\}, ()) \mid \\ &\quad \rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)].\mathbf{L3}(\zeta) = (\ell_1, \ell_2) \wedge \\ &\quad (W, (\mathsf{H}_1, v_1), (\mathsf{H}_2, v_2)) \in \mathcal{V}[\tau_2]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)]}\} && \text{(by induction)} \\ &= \mathcal{V}[\mathbf{cap} \zeta \tau_2]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)]} && \text{(by def)} \end{aligned}$$

CASE $\tau = \text{cap } \zeta_2 \tau_2$ WHERE $\zeta_2 \neq \zeta$.

$$\begin{aligned}
& \mathcal{V}[[\zeta \mapsto \zeta' \text{cap } \zeta_2 \tau_2]]_\rho \\
&= \mathcal{V}[\text{cap } \zeta_2 [\zeta \mapsto \zeta' \tau_2]]_\rho && \text{(by sub)} \\
&= \{(W, (\mathbf{H}_1 \uplus \{\ell_1 \mapsto v_1\}, ()), (\mathbf{H}_2 \uplus \{\ell_2 \mapsto v_2\}, ()) \mid \\
&\quad \rho.\mathbf{L3}(\zeta_2) = (\ell_1, \ell_2) \wedge (W, (\mathbf{H}_1, v_1), (\mathbf{H}_2, v_2)) \in \mathcal{V}[[\zeta \mapsto \zeta' \tau_2]]_\rho\} && \text{(by def)} \\
&= \{(W, (\mathbf{H}_1 \uplus \{\ell_1 \mapsto v_1\}, ()), (\mathbf{H}_2 \uplus \{\ell_2 \mapsto v_2\}, ()) \mid \\
&\quad \rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)].\mathbf{L3}(\zeta_2) = (\ell_1, \ell_2) \wedge \\
&\quad (W, (\mathbf{H}_1, v_1), (\mathbf{H}_2, v_2)) \in \mathcal{V}[[\zeta \mapsto \zeta' \tau_2]]_\rho\} && \text{(by lookup)} \\
&= \{(W, (\mathbf{H}_1 \uplus \{\ell_1 \mapsto v_1\}, ()), (\mathbf{H}_2 \uplus \{\ell_2 \mapsto v_2\}, ()) \mid \\
&\quad \rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)].\mathbf{L3}(\zeta_2) = (\ell_1, \ell_2) \wedge \\
&\quad (W, (\mathbf{H}_1, v_1), (\mathbf{H}_2, v_2)) \in \mathcal{V}[[\tau_2]]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)]}\} && \text{(by induction)} \\
&= \mathcal{V}[\text{cap } \zeta \tau_2]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell'_1, \ell'_2)]} && \text{(by def)}
\end{aligned}$$

The other cases are straightforward by expanding the definitions of $\mathcal{V}[\cdot]_\cdot, \mathcal{E}[\cdot]_\cdot$ and applying the induction hypotheses. \square

Lemma C.0.7 (Irrelevant Location Variables in \mathbf{L}^3). *If $\Delta \vdash \tau$, $\rho \in \mathcal{D}[\Delta]$, and $\zeta \notin \Delta$, then*

$$\mathcal{V}[\tau]_\rho = \mathcal{V}[\tau]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_1, \ell_2)]}$$

Proof. Since $\zeta \notin \Delta$ and $\Delta \vdash \tau$, it must be that ζ is not free in τ . Therefore, the definition of either $\mathcal{V}[\tau]_\rho$ or $\mathcal{V}[\tau]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_1, \ell_2)]}$ will never require looking up $\rho.\mathbf{L3}(\zeta)$, so whether ζ is in the domain of $\rho.\mathbf{L3}$ or not is irrelevant for the definition of the value relation. It then trivially follows that these two value relations are equal. \square

Lemma C.0.8 (Value Lifting). *If $(W, (\mathbf{H}_1, e_1), (\mathbf{H}_2, e_2)) \in \mathcal{V}[\tau]_\rho$ and, if τ is a **Miniml** type, $\mathbf{H}_1 = \mathbf{H}_2 = \emptyset$, then*

$$(W, (\mathbf{H}_1, e_1), (\mathbf{H}_2, e_2)) \in \mathcal{E}[\tau]_\rho$$

Proof. Since **Miniml** and \mathbf{L}^3 have different definitions of $\mathcal{E}[\cdot]_\cdot$, we must show the claim for the two languages separately.

Miniml LANGUAGE. Expanding the definition of $\mathcal{E}[\cdot]_\cdot$, we are to show that

$$\begin{aligned}
& \exists W', \mathbf{H}'_{1g}, \mathbf{H}'_{2g}. \forall \mathbf{H}_{2+} : MHeap. \exists v_2. \\
& \mathbf{H}_{1*} = \mathbf{H}'_{1g} \uplus \mathbf{H}_{1+} \wedge \mathbf{H}'_{1g}, \mathbf{H}'_{2g} : W' \wedge \\
& W \sqsubseteq_{(\text{dom}(\mathbf{H}_{1+}), \text{dom}(\mathbf{H}_{2+}), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(\mathbf{H}_{1+})), L_2 \cup FL(\text{cod}(\mathbf{H}_{2+}))))} W' \wedge \\
& (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[[\tau]]_\rho \wedge \\
& (\mathbf{H}_{2g+} \uplus \mathbf{H}_{2+}, e_2) \xrightarrow{*_{L_2}} (\mathbf{H}'_{2g} \uplus \mathbf{H}_{2+}, v_2) \not\rightarrow_{L_2}
\end{aligned} \tag{19}$$

given arbitrary $L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : M\mathsf{Heap}, \mathsf{H}_{1*}$, such that

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, e_1) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow$$

But if $(W, (\emptyset, e_1), (\emptyset, e_2)) \in \mathcal{V}[\tau]_\rho$, then e_1, e_2 are values. Since configurations with values as programs do not step, $v_1 = e_1$ and we can choose $W' = W$, $\mathsf{H}'_{1g} = \mathsf{H}_{1g}$, $\mathsf{H}'_{2g} = \mathsf{H}_{2g}$, and $v_2 = e_2$. Then, by assumption, we have $(W, (\emptyset, e_1), (\emptyset, e_2)) \in \mathcal{V}[\tau]_\rho$, which suffices to finish the proof.

L3 LANGUAGE Expanding the definition of $\mathcal{E}[\cdot]$, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : M\mathsf{Heap}. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}_{2g'} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\tau]_\rho \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, e_2) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \not\rightarrow \end{aligned} \tag{20}$$

given arbitrary $L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : M\mathsf{Heap}, \mathsf{H}_{1*}$, such that

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, e_1) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow$$

But if $(W, (\mathsf{H}_1, e_1), (\mathsf{H}_2, e_2)) \in \mathcal{V}[\tau]_\rho$, then e_1, e_2 are values. Since configurations with values as programs do not step, $v_1 = e_1$ and we can choose $W' = W$, $\mathsf{H}'_{1g} = \mathsf{H}_{1g}$, $\mathsf{H}'_{2g} = \mathsf{H}_{2g}$, and $v_2 = e_2$. Then, by assumption, we have $(W, (\mathsf{H}_1, e_1), (\mathsf{H}_2, e_2)) \in \mathcal{V}[\tau]_\rho$, which suffices to finish the proof.

□

Lemma C.0.9 (Split Substitutions). *For any world W and substitution γ such that*

$$(W, \mathsf{H}_1, \mathsf{H}_2, \gamma) \in \mathcal{G}[\Gamma_1 \uplus \Gamma_2]_\rho$$

there exist $\gamma_1, \gamma_2, \mathsf{H}_{1l}, \mathsf{H}_{1r}, \mathsf{H}_{2l}, \mathsf{H}_{2r}$ such that $\gamma = \gamma_1 \uplus \gamma_2$, $\mathsf{H}_1 = \mathsf{H}_{1l} \uplus \mathsf{H}_{1r}$, $\mathsf{H}_2 = \mathsf{H}_{2l} \uplus \mathsf{H}_{2r}$,

$$(W, \mathsf{H}_{1l}, \mathsf{H}_{1r}, \gamma_1) \in \mathcal{G}[\Gamma_1]_\rho$$

and

$$(W, \mathsf{H}_{2l}, \mathsf{H}_{2r}, \gamma_2) \in \mathcal{G}[\Gamma_2]_\rho$$

Moreover, for any $i, j \in \{1, 2\}$, for any $\Delta; \Gamma; \Delta; \Gamma_i \vdash \mathbf{e}_i : \tau$ and $\gamma_\Gamma \in \mathcal{G}[\Gamma]_\rho$,

$$\gamma^j(\gamma_\Gamma^j(\mathbf{e}_i^+)) = \gamma_i^j(\gamma_\Gamma^j(\mathbf{e}_i^+))$$

Proof. First, we need to show that there exist substitutions γ_1 and γ_2 . This follows from the inductive structure of $\mathcal{G}[\Gamma_1 \uplus \Gamma_2]_\rho$, where we can separate the parts that came from $\mathcal{G}[\Gamma_1]_\rho$ and $\mathcal{G}[\Gamma_2]_\rho$. The second follows from the fact that the statics means that the rest of the substitution must not occur in the term. and thus $\gamma^j(\mathbf{e}_1^+) = \gamma_1^j(\gamma_2^j(\mathbf{e}_1^+)) = \gamma_1^j(\mathbf{e}_1^+)$ (for example). \square

Lemma C.0.10 (Bang Substitutions Own No Heap). *For any $(W, H_1, H_2, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho$, it must be the case that $H_1 = H_2 = \emptyset$.*

Proof. We will prove the lemma by induction on the size of $!\Gamma$. If $!\Gamma$ is empty, then the theorem is trivial. Otherwise, suppose that $!\Gamma = !\Gamma_2, x : !\tau$. Then,

$$(W, H_1, H_2, \gamma_\Gamma) = (W, H'_1 \uplus H_{1v}, H'_2 \uplus H_{2v}, \gamma_L \cdot \Gamma' [x \mapsto (v_1, v_2)])$$

where $(W, H'_1, H'_2, \gamma_L \cdot \Gamma) \in \mathcal{G}[\Gamma_2]_\rho$ and $(W, (H_{1v}, v_1), (H_{2v}, v_2)) \in \mathcal{V}[\tau]_\rho$. By induction, $H'_1 = H'_2 = \emptyset$ and by expanding the value relation, $H_{1v} = H_{2v} = \emptyset$. Thus, $H_1 = H_2 = \emptyset$, as was to be proven. \square

Lemma C.0.11 (\mathbf{L}^3 Values Compile to LCVM values). *If $\Delta; \Gamma; \Delta; \Gamma \vdash v : \tau$ then given $\rho, \gamma_L, \gamma_\Gamma, W, H_1, H_2$ such that*

$$\rho \cdot \mathbf{L}^3 \in \mathcal{D}[\Delta], \rho \cdot \mathbf{F} \in \mathcal{D}[\Delta], (W, H_1, H_2, \gamma_L) \in \mathcal{G}[\Gamma]_\rho, \gamma_\Gamma \in \mathcal{G}[\Gamma]_\rho$$

it holds that $\gamma_L^1(\gamma_\Gamma^1(v^+))$ and $\gamma_L^2(\gamma_\Gamma^2(v^+))$ are both target values.

Proof. We will prove the theorem by induction over v .

CASE $v = ()$.

If $v = ()$, then $v^+ = ()$, which is a target value.

CASE $v = b$ FOR SOME $b \in \mathbb{B}$.

If $v = b$, then $v^+ = n$ for some $n \in \{0, 1\}$, which is a target value.

CASE $v = \lambda x : \tau. e$.

If $v = \lambda x : \tau. e$, then $v^+ = \lambda x. e^+$, which is a target value.

CASE $v = \Lambda \zeta. e$.

If $v = \Lambda \zeta. e$, then $v^+ = \lambda x_\zeta. e^+$, which is a target value.

CASE $v = \lceil \zeta, v \rceil$.

If $v = \lceil \zeta, v \rceil$, then $v^+ = v'^+$. Ergo, for any $i \in \{1, 2\}$, $\gamma_L^i(\gamma_\Gamma^i(v^+)) = \gamma_L^i(\gamma_\Gamma^i(v'^+))$, which is a target value by induction.

CASE $v = (v_1, v_2)$.

If $v = (v_1, v_2)$, then $v^+ = (v_1^+, v_2^+)$. Ergo, for any $i \in \{1, 2\}$, $\gamma_L^i(\gamma_\Gamma^i(v^+)) = (\gamma_L^i(\gamma_\Gamma^i(v_1^+)), \gamma_L^i(\gamma_\Gamma^i(v_2^+)))$ is a target value because it is a pair of values as, by induction, $\gamma_L^i(\gamma_\Gamma^i(v_1^+))$ and $\gamma_L^i(\gamma_\Gamma^i(v_2^+))$ are target values.

CASE $\mathbf{v} = !\mathbf{v}'$.

If $\mathbf{v} = !\mathbf{v}'$, then $\mathbf{v}^+ = \mathbf{v}'^+$. Ergo, for any $i \in \{1, 2\}$, $\gamma_{\text{L}}^i(\gamma_{\text{T}}^i(\mathbf{v}^+)) = \gamma_{\text{L}}^i(\gamma_{\text{T}}^i(\mathbf{v}'^+))$, which is a target value by induction.

□

Lemma C.0.12 (Lifting Steps Out of Evaluation Context). *If $(\mathbf{H}, \mathbf{K}[\mathbf{e}]) \rightarrow_L (\mathbf{H}', \mathbf{K}[\mathbf{e}'])$ and $\mathbf{K}[\mathbf{e}']$ is not of the form fail c , then $(\mathbf{H}, \mathbf{e}) \rightarrow_{L \cup FL(K[\cdot])} (\mathbf{H}, \mathbf{e}')$.*

Proof. Since it is given that $\mathbf{K}[\mathbf{e}']$ is not of the form fail c , there are two cases:

1. The given \rightarrow_L is the result of a callgc instruction. In this case, \mathbf{e} must be of the form $\mathbf{K}'[\text{callgc}]$ and \mathbf{e}' must be of the form $\mathbf{K}'[()]$ for some evaluation context \mathbf{K}' . Moreover, there exist $\mathbf{H}_{gc} : GCH_{\text{Heap}}$, $\mathbf{H}_m : M_{\text{Heap}}$, \mathbf{H}'_{gc} such that $\mathbf{H} = \mathbf{H}_{gc} \uplus \mathbf{H}_m$, $\mathbf{H}' = \mathbf{H}'_{gc} \uplus \mathbf{H}_m$, $\mathbf{H}'_{gc} \subseteq \mathbf{H}_{gc}$ and

$$\text{reachablelocs}(\mathbf{H}_{gc} \uplus \mathbf{H}_m, FL(K[K'[\cdot]]) \cup L) \cap \text{dom}(\mathbf{H}_{gc}) \subseteq \text{dom}(\mathbf{H}'_{gc})$$

Then, notice that $FL(K[K'[\cdot]]) = FL(K'[\cdot]) \cup FL(K[\cdot])$. Ergo,

$$(\mathbf{H}_{gc} \uplus \mathbf{H}_m, \mathbf{K}'[\text{callgc}]) \rightarrow_{L \cup FL(K[\cdot])} (\mathbf{H}'_{gc} \uplus \mathbf{H}_m, \mathbf{K}'[()])$$

as was to be proven.

2. The given \rightarrow_L is the result of a \Rightarrow . In this case, \mathbf{e} must be of the form $\mathbf{K}'[\mathbf{e}_\bullet]$ and \mathbf{e}' must be of the form $\mathbf{K}'[\mathbf{e}'_\bullet]$ for some evaluation context \mathbf{K} and expressions $\mathbf{e}_\bullet, \mathbf{e}'_\bullet$ such that $(\mathbf{H}, \mathbf{e}_\bullet) \Rightarrow (\mathbf{H}', \mathbf{e}'_\bullet)$. It then follows that $(\mathbf{H}, \mathbf{K}'[\mathbf{e}_\bullet]) \Rightarrow (\mathbf{H}', \mathbf{K}'[\mathbf{e}'_\bullet])$, as was to be proven.

□

Lemma C.0.13 (Stepping Respects Evaluation Context). *If $(\mathbf{H}, \mathbf{e}) \rightarrow_{L \cup FL(K[\cdot])} (\mathbf{H}', \mathbf{e}')$ and \mathbf{e}' is not of the form fail c , then*

1. $(\mathbf{H}, \mathbf{K}[\mathbf{e}]) \rightarrow_L (\mathbf{H}', \mathbf{K}[\mathbf{e}'])$
2. for any $\mathbf{H}_\bullet, \mathbf{e}_\bullet$ such that $(\mathbf{H}, \mathbf{K}[\mathbf{e}]) \rightarrow_L (\mathbf{H}_\bullet, \mathbf{e}_\bullet)$, there exists a $\mathbf{e}_{\bullet\bullet}$ such that $\mathbf{e}_\bullet = \mathbf{K}[\mathbf{e}_{\bullet\bullet}]$.

Proof. Proving (1) is trivially similar to the proof of Lemma C.0.12, so we focus on proving (2). We do case analysis on the reduction $(\mathbf{H}, \mathbf{e}) \rightarrow_{L \cup FL(K[\cdot])} (\mathbf{H}', \mathbf{e}')$:

1. The given $\rightarrow_{L \cup FL(K[\cdot])}$ is the result of a callgc instruction. In this case, \mathbf{e} must be of the form $\mathbf{K}'[\text{callgc}]$. Then, for any $\mathbf{H}_\bullet, \mathbf{e}_\bullet$ such that $(\mathbf{H}, \mathbf{K}[\mathbf{e}]) \rightarrow_L (\mathbf{H}_\bullet, \mathbf{e}_\bullet)$, because $\mathbf{K}[\mathbf{e}] = \mathbf{K}[\mathbf{K}'[\text{callgc}]]$, that step must be a callgc instruction, so $\mathbf{e}_\bullet = \mathbf{K}[\mathbf{K}'[()]]$, and thus choosing $\mathbf{e}_{\bullet\bullet} = \mathbf{K}'[()]$ suffices to finish the proof.

2. The given $\rightarrow_{L \cup FL(K[\cdot])}$ is the result of a \Rightarrow . In this case, e must be of the form $K'[e^*]$ and e' must be of the form $K'[e^{*\prime}]$ for some evaluation context K' and expressions $e^*, e^{*\prime}$ such that $(H, e^*) \Rightarrow (H', e^{*\prime})$. Moreover, $e^* \neq \text{callgc}$, because $(H, \text{callgc}) \not\Rightarrow$, and e^* is not of the form $\text{fail } c$, because $(H, \text{fail } c) \not\Rightarrow$.

Ergo, for any H_\bullet, e_\bullet such that $(H, K[e]) \rightarrow_L (H_\bullet, e_\bullet)$, because $K[e] = K[K'[e^*]]$, this step must be the result of a \Rightarrow . Thus, there exists some $e^{*\prime\prime}$ such that $(H, e^*) \Rightarrow (H_\bullet, e^{*\prime\prime})$ and $e_\bullet = K[K'[e^{*\prime\prime}]]$, so choosing $e'_{\bullet\bullet} = K'[e^{*\prime\prime}]$ suffices to finish the proof.

□

Lemma C.0.14 (Subterm Termination). *If $(H, e) \xrightarrow{*}_L (H', e') \not\rightarrow_L$ where e' is not of the form $\text{fail } c$ and $(H, e) \xrightarrow{*}_L (H_\bullet, K[e_\bullet])$ is a prefix of the aforementioned reduction, then $(H, e) \xrightarrow{*}_L (H'_\bullet, K[e'_\bullet])$ is also a prefix of the original reduction for some H'_\bullet, e'_\bullet such that $(H_\bullet, e_\bullet) \xrightarrow{*}_{L \cup FL(K[\cdot])} (H'_\bullet, e'_\bullet) \not\rightarrow_L$*

Proof. Consider the largest integer n such that there is a reduction

$$(H, e) \xrightarrow{*}_L (H_\bullet, K[e_\bullet]) \rightarrow_L (H_{\bullet,1}, K[e_{\bullet,1}]) \rightarrow_L \cdots \rightarrow_L (H_{\bullet,n}, K[e_{\bullet,n}]) \quad (21)$$

that is a prefix of the original reduction $(H, e) \xrightarrow{*}_L (H', e') \not\rightarrow_L$.

There exists such an integer n because we can choose $n = 0$. Moreover, there is an upper bound on such integers n because the original reduction is terminating and thus has finite length. Also, since $(H_\bullet, K[e_\bullet]) \xrightarrow{*}_L (H_{\bullet,n}, K[e_{\bullet,n}])$, by Lemma C.0.12, $(H_\bullet, e_\bullet) \xrightarrow{*}_L (H_{\bullet,n}, e_{\bullet,n})$. There are two cases:

1. This prefix is the entire reduction $(H, e) \xrightarrow{*}_L (H', e') \not\rightarrow_L$, implying that $H' = H_{\bullet,n}$ and $e' = K[e_{\bullet,n}]$. Thus, $(H_{\bullet,n}, K[e_{\bullet,n}]) \not\rightarrow_L$, so by Lemma C.0.13, $(H_{\bullet,n}, e_{\bullet,n}) \not\rightarrow_{L \cup FL(K[\cdot])}$. Thus, choosing $H'_\bullet = H_{\bullet,n}$ and $e'_\bullet = e_{\bullet,n}$ suffices to finish the proof.
2. This prefix is not the entire reduction, so $(H, e) \xrightarrow{*}_L (H_{\bullet,n}, K[e_{\bullet,n}]) \rightarrow_L (H'', e'')$ is also a prefix of the original reduction. e'' can not be of the form $K[e''']$ because if it were, then we could choose $H_{\bullet,n+1} = H''$ and $e_{\bullet,n+1} = e''$ to create a longer reduction of the form (21), which would contradict the maximality of n . Ergo, if $(H_{\bullet,n}, e_{\bullet,n})$ were not irreducible under $\rightarrow_{L \cup FL(K[\cdot])}$, that would contradict Lemma C.0.13. Thus, $(H_{\bullet,n}, e_{\bullet,n}) \not\rightarrow_{L \cup FL(K[\cdot])}$, so choosing $H'_\bullet = H_{\bullet,n}$ and $e'_\bullet = e_{\bullet,n}$ suffices to finish the proof.

□

Note that when applying Lemma C.0.14, we sometimes leave K implicit and we often write “ e'_\bullet ” as “ v ”, even though it must actually be proven to be a value.

Moreover, in the proofs of the compatibility lemma, we are often given that $(H, e) \xrightarrow{^*} (H_1, v_1) \not\rightarrow$ and then we apply Lemma C.0.14, possibly multiple times, to show a reduction $(H, e) \xrightarrow{^*} (H', v') \not\rightarrow$ for some other configuration (H', v') . We then conclude that $(H_1, v_1) = (H', v')$ because, even though \rightarrow_L is not confluent, we implicitly deduce that since we applied Lemma C.0.14, the reduction $(H, e) \xrightarrow{^*} (H', v') \not\rightarrow$ is a prefix of the original given reduction $(H, e) \xrightarrow{^*} (H_1, v_1) \not\rightarrow$.

Theorem C.0.15 (Convertibility Soundness). *If $\tau_A \sim \tau_B$ then for all ρ ,*

1. $\forall (W, (H_1, e_1), (H_2, e_2)) \in \mathcal{E}[\![\tau_A]\!]_\rho. (W, (H_1, C_{\tau_A \mapsto \tau_B} e_1), (H_2, C_{\tau_A \mapsto \tau_B} e_2)) \in \mathcal{E}[\![\tau_B]\!]_\rho; \text{ and}$
2. $\forall (W, (H_1, e_1), (H_2, e_2)) \in \mathcal{E}[\![\tau_B]\!]_\rho. (W, (H_1, C_{\tau_B \mapsto \tau_A} e_1), (H_2, C_{\tau_B \mapsto \tau_A} e_2)) \in \mathcal{E}[\![\tau_A]\!]_\rho$

Proof. By simultaneous induction on the structure of the convertibility relation.

$$\boxed{\langle \tau \rangle \sim \tau}$$

1. We are to show that

$$\forall (W, (H_1, e_1), (H_2, e_2)) \in \mathcal{E}[\![\langle \tau \rangle]\!]_\rho. (W, (H_1, C_{\langle \tau \rangle \mapsto \tau} e_1), (H_2, C_{\langle \tau \rangle \mapsto \tau} e_2)) \in \mathcal{E}[\![\tau]\!]_\rho$$

Expanding the definition of $C_{\langle \tau \rangle \mapsto \tau}$, $\mathcal{E}[\![\cdot]\!]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} &\exists H'_1, H'_{1g}. \forall H_{2+} : MHeap. \exists H'_2, W', H'_{2g}, v_2. \\ &H_{1*} = H'_{1g} \uplus H'_1 \uplus H_{1+} \wedge H'_{1g}, H'_{2g} : W' \wedge \\ &W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(H_{1+})), L_2 \cup \text{FL}(\text{cod}(H_{2+})))} W' \wedge \\ &(W', (H'_1, v_1), (H'_2, v_2)) \in \mathcal{V}[\![\tau]\!]_\rho \wedge \\ &(H_{2g+} \uplus H_2 \uplus H_{2+}, e_2) \xrightarrow{^*}_{L_2} (H'_{2g} \uplus H'_2 \uplus H_{2+}, v_2) \not\rightarrow_{L_2} \end{aligned}$$

given arbitrary $W, L_1, L_2, H_{1g+}, H_{2g+} : MHeap, H_{1*}$ such that

$$(H_{1g+} \uplus H_1 \uplus H_{1+}, e_1) \xrightarrow{^*}_{L_1} (H_{1*}, v_1) \not\rightarrow_{L_1}$$

Because $(W, (H_1, e_1), (H_2, e_2)) \in \mathcal{E}[\![\langle \tau \rangle]\!]_\rho$, we find that $H_{1*} = H'_{1g} \uplus H_{1+}$ and

$$(H_{2g+} \uplus H_2 \uplus H_{2+}, e_2) \xrightarrow{^*}_{L_2} (H'_{2g} \uplus H_{2+}, v_2) \not\rightarrow_{L_2}$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some world
 $W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'$
such that

$$(W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\langle \tau \rangle]_\rho$$

From here, we can take $W' = W'$, $\mathsf{H}'_1 = \emptyset$, $\mathsf{H}'_2 = \emptyset$, $\mathsf{H}'_{1g} = \mathsf{H}'_{1g}$, and $\mathsf{H}'_{2g} = \mathsf{H}'_{2g}$. Then, from expanding $(W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\langle \tau \rangle]_\rho$, we find $(W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\langle \tau \rangle]_\rho$, which suffices to finish the proof.

2. We are to show that

$$\forall (W, (\mathsf{H}_1, e_1), (\mathsf{H}_2, e_2)) \in \mathcal{E}[\tau]_\rho. (W, (\mathsf{H}_1, C_{\langle \tau \rangle \mapsto \tau} e_1), (\mathsf{H}_2, C_{\langle \tau \rangle \mapsto \tau} e_2)) \in \mathcal{E}[\langle \tau \rangle]_\rho$$

Expanding the definition of $C_{\langle \tau \rangle \mapsto \tau}$, $\mathcal{E}[\cdot]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : M\text{Heap}. \exists W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}_{2g'} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\langle \tau \rangle]_\rho \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, e_2) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, v_2) \not\rightarrow_{L_2} \end{aligned}$$

given arbitrary $W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : M\text{Heap}, \mathsf{H}_{1*}$ such that

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, e_1) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

Because $(W, (\emptyset, e_1), (\emptyset, e_2)) \in \mathcal{E}[\tau]_\rho$, we find that $\mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+}$ and

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, e_2) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_1) \not\rightarrow_{L_2}$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some world
 $W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'$
such that

$$(W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\tau]_\rho$$

Recall that $\tau \in \text{DUPLICABLE} = \{\text{unit}, \text{bool}, \text{ptr } \zeta, !\tau\}$. Then by inspecting definitions of $\mathcal{V}[\tau]_\rho$ for all four of these cases, we have that $\mathsf{H}'_1 = \mathsf{H}'_2 = \emptyset$.

We then take $W' = W'$, $\mathsf{H}'_1 = \emptyset$, $\mathsf{H}'_2 = \emptyset$, $\mathsf{H}'_{1g} = \mathsf{H}'_{1g}$, and $\mathsf{H}'_{2g} = \mathsf{H}'_{2g}$. Finally, given $(W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau]_\rho$, it follows that $(W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\langle \tau \rangle]_\rho$.

$\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha) \sim \text{bool}$

1. We are to show that

$$\begin{aligned} \forall (W, (\mathsf{H}_1, e_1), (\mathsf{H}_2, e_2)) \in \mathcal{E}[\![\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha)]\!]_\rho. \\ ((W, (\mathsf{H}_1, C_{\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha) \mapsto \text{bool}} e_1), (\mathsf{H}_2, C_{\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha) \mapsto \text{bool}} e_2)) \in \mathcal{E}[\![\text{bool}]\!]_\rho \end{aligned}$$

Expanding the definition of $C_{\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha) \mapsto \text{bool}}$, we are to show that

$$(W, (\mathsf{H}_1, \text{let } f_1 = e_1 \text{ in } ((f_1 ()) 0) 1), (\mathsf{H}_2, \text{let } f_2 = e_2 \text{ in } ((f_2 ()) 0) 1)) \in \mathcal{E}[\![\text{bool}]\!]_\rho$$

given arbitrary e_1, e_2 such that $(W, (\mathsf{H}_1, e_1), (\mathsf{H}_2, e_2)) \in \mathcal{E}[\![\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha)]\!]_\rho$.

Expanding the definition of $C_{\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha) \mapsto \text{bool}}$, $\mathcal{E}[\![\cdot]\!]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : MHeap. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}_{2g'} : W' \wedge \\ W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\![\text{bool}]\!]_\rho \wedge \\ (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } f_2 = e_2 \text{ in } ((f_2 ()) 0) 1) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \rightsquigarrow_{L_2} \end{aligned}$$

given arbitrary $W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : MHeap, \mathsf{H}_{1*}$ such that

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } f_1 = e_1 \text{ in } ((f_1 ()) 0) 1) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \rightsquigarrow_{L_1}$$

By Lemma C.0.14, we have that $(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, e_1) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}^1, v_1^1) \rightsquigarrow_{L_1}$ for some H_{1*}^1, v_1^1 . Expanding the definition of $\mathcal{E}[\![\cdot]\!]_\rho$ in the premise and specializing where appropriate, we have that $\mathsf{H}_{1*}^1 = \mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}$ and

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, e_2) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, v_2^1) \rightsquigarrow_{L_2}$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some

$$W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'$$

such that

$$(W', (\emptyset, v_1^1), (\emptyset, v_2^1)) \in \mathcal{V}[\![\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha)]\!]_\rho$$

Expanding the definition of $\mathcal{V}[\![\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha)]\!]_\rho$, we have that

$$\begin{aligned} v_1^1 = \lambda_- e_1^1 \wedge v_2^1 = \lambda_- e_2^1 \wedge \\ \forall R \in \text{RelT}. (W', (\emptyset, e_1^1), (\emptyset, e_2^1)) \in \mathcal{E}[\![\alpha \rightarrow (\alpha \rightarrow \alpha)]\!]_{\rho[\mathbf{F}(\alpha) \mapsto R]} \end{aligned}$$

To proceed, we take $R = \mathcal{V}[\langle \text{bool} \rangle]_\rho$. We do this because we expect the reduction to eventually need a value in $\mathcal{V}[\text{bool}]_\rho$, but by using the type $\langle \text{bool} \rangle$ instead (which has the same interpretation in our model), we can apply Lemma C.0.5 to get that:

$$(W', (\emptyset, e_1^1), (\emptyset, e_2^1)) \in \mathcal{E}[\langle \text{bool} \rangle \rightarrow (\langle \text{bool} \rangle \rightarrow \langle \text{bool} \rangle)]_\rho$$

By the operational semantics of LCVM, we now have that

$$\begin{aligned} & (\mathsf{H}_{1g} \uplus \mathsf{H}_{1+}, \text{let } f_1 = e_1 \text{ in } ((f_1 \ ()) \ 0) \ 1) \\ & \xrightarrow{*_{L_1}} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, \text{let } f_1 = \lambda _. e_1^1 \text{ in } ((f_1 \ ()) \ 0) \ 1) \\ & \xrightarrow{*_{L_1}} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, [f_1 \mapsto \lambda _. e_1^1] ((f_1 \ ()) \ 0) \ 1) \\ & = (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, (((\lambda _. e_1^1) \ ()) \ 0) \ 1) \\ & \xrightarrow{*_{L_1}} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, (e_1^1 \ 0) \ 1) \\ & \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1} \end{aligned}$$

By Lemma C.0.14, we have that

$$(\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, e_1^1) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}^2, v_1^2) \not\rightarrow_{L_1}$$

for some H_{1*}^2, v_1^2 . Expanding the definition of $\mathcal{E}[\]_\rho$ and specializing where appropriate, we have that $\mathsf{H}_{1*}^2 = \mathsf{H}_{1g}'' \uplus \mathsf{H}_{1+}$ and

$$(\mathsf{H}_{2g}' \uplus \mathsf{H}_{2+}, e_2^1) \xrightarrow{*_{L_2}} (\mathsf{H}_{2g}'' \uplus \mathsf{H}_{2+}, v_2^2) \not\rightarrow_{L_2}$$

where $\mathsf{H}_{1g}'', \mathsf{H}_{2g}'' : W''$ for some $W' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+}), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(\mathsf{H}_{1+})), L_2 \cup FL(\text{cod}(\mathsf{H}_{2+}))))} W''$ such that

$$(W'', (\emptyset, v_1^2), (\emptyset, v_2^2)) \in \mathcal{V}[\langle \text{bool} \rangle \rightarrow (\langle \text{bool} \rangle \rightarrow \langle \text{bool} \rangle)]_\rho$$

Expanding the definition of $\mathcal{V}[\langle \text{bool} \rangle \rightarrow (\langle \text{bool} \rangle \rightarrow \langle \text{bool} \rangle)]_\rho$, we have that

$$\begin{aligned} v_1^2 &= \lambda x_1^2. e_1^2 \wedge v_2^2 = \lambda x_2^2. e_2^2 \wedge \\ &\forall (W'', (\emptyset, v_1^a), (\emptyset, v_2^a)) \in \mathcal{V}[\langle \text{bool} \rangle]_\rho. \\ &(W'', (\emptyset, [x_1^2 \mapsto v_1^a] e_1^2), (\emptyset, [x_2^2 \mapsto v_2^a] e_2^2)) \in \mathcal{E}[\langle \text{bool} \rangle \rightarrow \langle \text{bool} \rangle]_\rho \end{aligned}$$

Observe that $\mathcal{V}[\langle \text{bool} \rangle]_\rho = \mathcal{V}[\text{bool}]_\rho$ and $(W'', (\emptyset, 0), (\emptyset, 0)) \in \mathcal{V}[\text{bool}]_\rho$ by definition, so

$(W'', (\emptyset, [x_1^2 \mapsto 0]e_1^2), (\emptyset, [x_2^2 \mapsto 0]e_2^2)) \in \mathcal{E}[\langle \text{bool} \rangle \rightarrow \langle \text{bool} \rangle]_\rho$. By Lemma C.0.14, we now have that

$$(\mathsf{H}_{1g}'' \uplus \mathsf{H}_{1+}, (\lambda x_1^2.e_1^2) \ 0) \xrightarrow{*} 1_{L_1} (\mathsf{H}_{1g}'' \uplus \mathsf{H}_{1+}, [x_1^2 \mapsto 0]e_1^2) \xrightarrow{*}_{L_1} (\mathsf{H}_{1*}^3, v_1^3) \rightsquigarrow$$

for some H_{1*}^3, v_1^3 . Expanding the definition of $\mathcal{E}[\cdot]_\rho$, we have that $\mathsf{H}_{1*}^3 = \mathsf{H}_{1g}''' \uplus \mathsf{H}_{1+}$ and

$$(\mathsf{H}_{2g}'' \uplus \mathsf{H}_{2+}, [x_2^2 \mapsto 0]e_2^2) \rightarrow_{L_2} (\mathsf{H}_{2g}''' \uplus \mathsf{H}_{2+}, v_2^3) \rightsquigarrow_{L_2}$$

where $\mathsf{H}_{1g}''', \mathsf{H}_{2g}''' : W'''$ for some $W'' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'''$ such that

$$(W''', (\emptyset, v_1^3), (\emptyset, v_1^3)) \in \mathcal{V}[\langle \text{bool} \rangle \rightarrow \langle \text{bool} \rangle]_\rho$$

Expanding the definition of $\mathcal{V}[\langle \text{bool} \rangle \rightarrow \langle \text{bool} \rangle]_\rho$, we have that

$$\begin{aligned} v_1^3 &= \lambda x_1^3.e_1^3 \wedge v_2^3 = \lambda x_2^3.e_2^3 \wedge \\ \forall (W''', (\emptyset, v_1^a), (\emptyset, v_2^a)) \in \mathcal{V}[\langle \text{bool} \rangle]_\rho. \\ (W''', (\emptyset, [x_1^3 \mapsto v_1^a]e_1^3), (\emptyset, [x_2^3 \mapsto v_2^a]e_2^3)) &\in \mathcal{E}[\langle \text{bool} \rangle]_\rho \end{aligned}$$

Recall that $\mathcal{V}[\langle \text{bool} \rangle]_\rho = \mathcal{V}[\text{bool}]_\rho$ and $(W''', (\emptyset, 1), (\emptyset, 1)) \in \mathcal{V}[\text{bool}]_\rho$ by definition, so $(W''', (\emptyset, [x_1^3 \mapsto 1]e_1^3), (\emptyset, [x_2^3 \mapsto 1]e_2^3)) \in \mathcal{E}[\langle \text{bool} \rangle]_\rho$. We now have that

$$(\mathsf{H}_{1g}'' \uplus \mathsf{H}_{1+}, (\lambda x_1^3.e_1^3) \ 1) \xrightarrow{*} 1_{L_1} (\mathsf{H}_{1g}'' \uplus \mathsf{H}_{1+}, [x_1^3 \mapsto 1]e_1^3) \xrightarrow{*}_{L_1} (\mathsf{H}_{1*}^4, v_1^4) \rightsquigarrow$$

Expanding the definition of $\mathcal{E}[\cdot]_\rho$, we have that $\mathsf{H}_{1*}^4 = \mathsf{H}_{1g}''' \uplus \mathsf{H}_{1+}$ and

$$(\mathsf{H}_{2g}'' \uplus \mathsf{H}_{2+}, [x_2^3 \mapsto 1]e_2^3) \xrightarrow{*}_{L_2} (\mathsf{H}_{2g}''' \uplus \mathsf{H}_{2+}, v_2^4) \rightsquigarrow_{L_2}$$

where $\mathsf{H}_{1g}''', \mathsf{H}_{2g}''' : W'''$ for some $W'' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'''$ and

$$(W''', (\emptyset, v_1^4), (\emptyset, v_2^4)) \in \mathcal{V}[\langle \text{bool} \rangle]_\rho$$

It follows that $\mathsf{H}_{1*} = \mathsf{H}_{1g}''' \uplus \mathsf{H}_{1+}$ and $v_1 = v_1^4$. Thus, we choose $\mathsf{H}'_1 = \emptyset$, $\mathsf{H}'_2 = \emptyset$, $\mathsf{H}'_{1g} = \mathsf{H}_{1g}'''$, $\mathsf{H}'_{2g} = \mathsf{H}_{2g}'''$, and $W' = W'''$. The fact that $(W''', (\emptyset, v_1^4), (\emptyset, v_2^4)) \in \mathcal{V}[\text{bool}]_\rho$ follows trivially from the above statement and that $\mathcal{V}[\langle \text{bool} \rangle]_\rho = \mathcal{V}[\text{bool}]_\rho$.

Finally, all that remains to show that

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, \text{let } f_2 = e_2 \text{ in } ((f_2 \ ()) \ 0) \ 1) \xrightarrow{*}_{L_2} (\mathsf{H}_{2g}''' \uplus \mathsf{H}_{2+}, v_2^4) \rightsquigarrow_{L_2}$$

given arbitrary H_{2+} .

We have that

$$\begin{aligned}
 & (H_{2g+} \uplus H_{2+}, \text{let } f_2 = e_2 \text{ in } ((f_2 \ (\)) \ 0) \ 1) \\
 & \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, \text{let } f_2 = \lambda _. e_2^1 \text{ in } ((f_2 \ (\)) \ 0) \ 1) \\
 & \xrightarrow{*_{L_2}} 1_{L_2}(H'_{2g} \uplus H_{2+}, [f_2 \mapsto \lambda _. e_2^1] ((f_2 \ (\)) \ 0) \ 1) \\
 & = (H'_{2g} \uplus H_{2+}, ((\lambda _. e_2^1 \ (\)) \ 0) \ 1) \\
 & \xrightarrow{*_{L_2}} 1_{L_2}(H'_{2g} \uplus H_{2+}, (e_2^1 \ 0) \ 1) \\
 & \xrightarrow{*_{L_2}} (H''_{2g} \uplus H_{2+}, ((\lambda x_2^2. e_2^2) \ 0) \ 1) \\
 & \xrightarrow{*_{L_2}} 1_{L_2}(H''_{2g} \uplus H_{2+}, [x_2^2 \mapsto 0] e_2^2 \ 1) \\
 & \xrightarrow{*_{L_2}} (H'''_{2g} \uplus H_{2+}, \lambda x_2^3. e_2^3 \ 1) \\
 & \xrightarrow{*_{L_2}} 1_{L_2}(H''''_{2g} \uplus H_{2+}, [x_2^3 \mapsto 1] e_2^3) \\
 & \xrightarrow{*_{L_2}} (H''''_{2g} \uplus H_{2+}, v_2^4) \\
 & \not\rightarrow_{L_2}
 \end{aligned}$$

as was to be demonstrated.

2. We are to show that

$$\begin{aligned}
 & \forall (W, (H_1, e_1), (H_2, e_2)) \\
 & \in \mathcal{E}[\![\text{bool}]\!]_\rho. \\
 & (W, (H_1, C_{\text{bool} \rightarrow \forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha)} e_1), (H_2, C_{\text{bool} \rightarrow \forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha)} e_2)) \\
 & \in \mathcal{E}[\![\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha)]!]_\rho
 \end{aligned}$$

Expanding the definition of $C_{\text{bool} \rightarrow \forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha)}$, we are to show that

$$\begin{aligned}
 & (W, (H_1, \text{let } x_1 = e_1 \text{ in } (\lambda _. \lambda t_1. \lambda f_1. \text{if } x_1 \ t_1 \ f_1)), \\
 & (H_2, \text{let } x_2 = e_2 \text{ in } (\lambda _. \lambda t_2. \lambda f_2. \text{if } x_2 \ t_2 \ f_2))) \in \mathcal{E}[\![\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha)]!]_\rho
 \end{aligned}$$

given arbitrary e_1, e_2 such that $(W, (H_1, e_1), (H_2, e_2)) \in \mathcal{E}[\![\text{bool}]\!]_\rho$. Expanding the definition of $\mathcal{E}[\![\cdot]\!]_\rho$, we are to show that

$$\begin{aligned}
 & \exists H'_{1g}. \forall H_{2+} : MHeap. \exists W', H'_{2g}, v_2. \\
 & H_{1*} = H'_{1g} \uplus H_{1+} \wedge H'_{1g}, H_{2g'} : W' \wedge \\
 & W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))} W' \wedge \\
 & (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha)]!]_\rho \wedge \\
 & (H_{2g+} \uplus H_2 \uplus H_{2+}, \text{let } x_1 = e_1 \text{ in } (\lambda _. \lambda t_1. \lambda f_1. \text{if } x_1 \ t_1 \ f_1)) \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, v_2) \not\rightarrow_{L_2}
 \end{aligned}$$

given arbitrary $W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : M\mathsf{Heap}, \mathsf{H}_{1*}$ such that

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } x_2 = e_2 \text{ in } (\lambda _. \lambda t_2. \lambda f_2. \text{if } x_2 t_2 f_2)) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

By Lemma C.0.14, we have that $(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, e_1) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}^1, v_1^1) \not\rightarrow_{L_1}$. Expanding the definition of $\mathcal{E}[\cdot]_\rho$ in the premise and specializing where appropriate, we have that $\mathsf{H}_{1*}^1 = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+}$ and

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, e_2) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2^1) \not\rightarrow_{L_2}$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some

$$W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'$$

such that

$$(W', (\mathsf{H}'_1, v_1^1), (\mathsf{H}'_2, v_2^1)) \in \mathcal{V}[\![\text{bool}]\!]_\rho$$

Expanding the definition of $\mathcal{V}[\![\text{bool}]\!]_\rho$, we have that

$$\mathsf{H}'_1 = \emptyset \wedge \mathsf{H}'_2 = \emptyset \wedge v_1^1 = v_2^1 = b \wedge b \in \{0, 1\}$$

By Lemma C.0.14, we now have that

$$\begin{aligned} & (\mathsf{H}'_{1g} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } x_1 = b \text{ in } (\lambda _. \lambda t_1. \lambda f_1. \text{if } x_1 t_1 f_1)) \\ & \xrightarrow{*} 1_{L_1}(\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, [x_1 \mapsto b] (\lambda _. \lambda t_1. \lambda f_1. \text{if } x_1 t_1 f_1)) \\ & = (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, (\lambda _. \lambda t_1. \lambda f_1. \text{if } b t_1 f_1)) \\ & \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \\ & \not\rightarrow \end{aligned}$$

from which we conclude that $\mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}$ and $v_1 = (\lambda _. \lambda t_1. \lambda f_1. \text{if } b t_1 f_1)$ since configurations with values as programs do not step.

Then, to prove the goal, we take $W' = W'$, $\mathsf{H}'_{1g} = \mathsf{H}'_{1g}$, and $\mathsf{H}'_{2g} = \mathsf{H}'_{2g}$.

To show the configuration with the heap $H_{2g+} \uplus H_2 \uplus H_{2+}$ terminates, we have

$$\begin{aligned}
& (H_{2g+} \uplus H_2 \uplus H_{2+}, \text{let } x_2 = e_2 \text{ in } (\lambda _. \lambda t_2. \lambda f_2. \text{if } x_2 t_2 f_2)) \\
& \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, \text{let } x_2 = b \text{ in } (\lambda _. \lambda t_2. \lambda f_2. \text{if } x_2 t_2 f_2)) \\
& \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, [x_2 \mapsto b] (\lambda _. \lambda t_2. \lambda f_2. \text{if } x_2 t_2 f_2)) \\
& = (H'_{2g} \uplus H_{2+}, (\lambda _. \lambda t_2. \lambda f_2. \text{if } b t_2 f_2)) \\
& \not\rightarrow
\end{aligned}$$

Then take $H'_{2+} = H^1_{2+}$.

All that remains to show is

$$(W', (\emptyset, (\lambda _. \lambda t_1. \lambda f_1. \text{if } b t_1 f_1)), (\emptyset, (\lambda _. \lambda t_2. \lambda f_2. \text{if } b t_2 f_2))) \in \mathcal{V}[\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha)]_\rho$$

Expanding the definition of $\mathcal{V}[\cdot]_\rho$ and applying Lemma C.0.8, we are to show that

$$(W'', (\emptyset, (\lambda t_1. \lambda f_1. \text{if } b t_1 f_1)), (\emptyset, (\lambda t_2. \lambda f_2. \text{if } b t_2 f_2))) \in \mathcal{V}[\alpha \rightarrow (\alpha \rightarrow \alpha)]_{\rho[\mathbf{F}(\alpha) \mapsto R]}$$

given arbitrary $R \in RelT$ and worlds W'' such that $W' \sqsubseteq_{\emptyset, \emptyset, \lambda t_1. \lambda f_1. \text{if } b t_1 f_1, \lambda t_2. \lambda f_2. \text{if } b t_2 f_2} W''$. Expanding the definition of $\mathcal{V}[\alpha \rightarrow (\alpha \rightarrow \alpha)]_{\rho[\mathbf{F}(\alpha) \mapsto R]}$, pushing substitutions, and applying Lemma C.0.8, we are to show that

$$(W''', (\emptyset, (\lambda f_1. \text{if } b v_{1t} f_1)), (\emptyset, (\lambda f_2. \text{if } b v_{2t} f_2))) \in \mathcal{V}[\alpha \rightarrow \alpha]_{\rho[\mathbf{F}(\alpha) \mapsto R]}$$

given arbitrary worlds W''' such that $W'' \sqsubseteq_{\emptyset, \emptyset, \lambda f_1. \text{if } b v_{1t} f_1, \lambda f_2. \text{if } b v_{2t} f_2} W'''$ and arbitrary v_{1t}, v_{2t} such that $(W''', (\emptyset, v_{1t}), (\emptyset, v_{2t})) \in \mathcal{V}[\alpha]_{\rho[\mathbf{F}(\alpha) \mapsto R]}$. Expanding the definition of $\mathcal{V}[\alpha \rightarrow \alpha]_{\rho[\mathbf{F}(\alpha) \mapsto R]}$ and pushing substitutions, we are to show that

$$(W''', (\emptyset, (\text{if } b v_{1t} v_{1f})), (\emptyset, (\text{if } b v_{2t} v_{2f}))) \in \mathcal{E}[\alpha]_{\rho[\mathbf{F}(\alpha) \mapsto R]}$$

given arbitrary worlds W'''' such that $W''' \sqsubseteq_{\emptyset, \emptyset, \text{if } b v_{1t} v_{1f}, \text{if } b v_{2t} v_{2f}} W''''$ and arbitrary v_{1f}, v_{2f} such that $(\emptyset, v_{1f}, \emptyset, v_{2f}) \in \mathcal{V}[\alpha]_{\rho[\mathbf{F}(\alpha) \mapsto R]}$. Expanding the definition of $\mathcal{E}[\cdot]_\rho$, we are to show that

$$\begin{aligned}
& \exists H'_{1g}. \forall H_{2+} : MHeap. \exists W', H'_{2g}, v_2. \\
& H_{1*} = H'_{1g} \uplus H_{1+} \wedge H'_{1g}, H_{2g'} : W' \wedge \\
& W'''' \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))} W' \wedge \\
& (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha)]_\rho \wedge \\
& (H_{2g+} \uplus H_{2+}, \text{if } b v_{1t} v_{1f}) \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, v_2) \not\rightarrow_{L_2}
\end{aligned}$$

given arbitrary $L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W''', v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : M\mathsf{Heap}, \mathsf{H}_{1*}$ such that

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, \text{if } b \ v_{2t} \ v_{2f}) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

The operational semantics of LCVM offers two cases depending on the value of b . Suppose, without loss of generality, that $b = 0$. Then we have

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, \text{if } b \ v_{1t} \ v_1) \xrightarrow{*_{L_1}} (\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, v_{1t}) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

from which we conclude that $v_1 = v_{1t}, \mathsf{H}_{1*} = \mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}$ since configurations with values as programs do not step. Then we can take $W' = W''', \mathsf{H}'_{1g} = \mathsf{H}_{1g+}, \mathsf{H}'_{2g} = \mathsf{H}_{2g+}$, and $v_2 = v_{2t}$. All that remains is to show that

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, \text{if } 0 \ v_{2t} \ v_{2f}) \xrightarrow{*_{L_2}} (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, v_{2t}) \not\rightarrow_{L_2}$$

This is actually just one step by the operational semantics of LCVM.

The case in which $b = 1$ is analogous, exchanging v_{1t} with v_{if} where appropriate.

$$\boxed{\tau_1 \rightarrow \tau_2 \sim !(!\tau_1 \multimap \tau_2)}$$

1. We are to show that

$$\begin{aligned} & \forall (W, (\emptyset, e_1), (\emptyset, e_2)) \in \mathcal{E}[\![\tau_1 \rightarrow \tau_2]\!]_\rho. \\ & (W, (\emptyset, C_{\tau_1 \rightarrow \tau_2 \mapsto !(!\tau_1 \multimap \tau_2)} e_1), (\emptyset, C_{\tau_1 \rightarrow \tau_2 \mapsto !(!\tau_1 \multimap \tau_2)} e_2)) \in \mathcal{E}[\![!(!\tau_1 \multimap \tau_2)]!]_\rho \end{aligned}$$

Expanding the definition of $C_{\tau_1 \rightarrow \tau_2 \mapsto !(!\tau_1 \multimap \tau_2)} e_1$, we are to show that

$$(W, (\emptyset, \text{let } f_1 = e_1 \text{ in } \lambda x_1. (C_{\tau_2 \mapsto \tau_2} (\dots))), (\emptyset, \text{let } f_2 = e_2 \text{ in } \lambda x_2. (C_{\tau_2 \mapsto \tau_2} (\dots)))) \in \mathcal{E}[\![!(!\tau_1 \multimap \tau_2)]!]_\rho$$

given arbitrary e_1, e_2 such that $(\emptyset, e_1, \emptyset, e_2) \in \mathcal{E}[\![\tau_1 \rightarrow \tau_2]\!]_\rho$. Expanding the definition of $\mathcal{E}[\![\cdot]\!]_\rho$, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : M\mathsf{Heap}. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}_{2g'} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\![!(!\tau_1 \multimap \tau_2)]!]_\rho \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \\ & \text{let } f_2 = e_2 \text{ in } \lambda x_2. (C_{\tau_2 \mapsto \tau_2} (\dots))) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \not\rightarrow_{L_2} \end{aligned}$$

given arbitrary $W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : M\mathsf{Heap}, \mathsf{H}_{1*}$ such that

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } f_1 = e_1 \text{ in } \lambda x_1. (C_{\tau_2 \mapsto \tau_2} (\dots))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

By Lemma C.0.14, we have that $(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, e_1) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}^1, v_1^1) \not\rightarrow_{L_1}$ for some H_{1*}^1, v_1^1 . Expanding the definition of $\mathcal{E}[\tau_1 \rightarrow \tau_2]_\rho$ in the premise and specializing where appropriate, we have that $\mathsf{H}_{1*}^1 = \mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}$ and

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, e_2) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, v_2^1)$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some $W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'$ such that

$$(W', (\emptyset, v_1^1), (\emptyset, v_2^1)) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]_\rho$$

Expanding the definition of $\mathcal{V}[\tau_1 \rightarrow \tau_2]_\rho$, we have that

$$\begin{aligned} v_1^1 &= \lambda x_1^1. e_1^1 \wedge v_2^1 = \lambda x_2^1. e_2^1 \wedge \\ &\forall W''. W' \sqsubseteq_{\emptyset, \emptyset, e_1^1, e_2^1} W'' \wedge \forall (W'', (\emptyset, v_1^a), (\emptyset, v_2^a)) \in \mathcal{V}[\tau_1]_\rho. \\ &(W'', (\emptyset, [x_1^1 \mapsto v_1^a] e_1^1), (\emptyset, [x_2^1 \mapsto v_2^a] e_2^1)) \in \mathcal{E}[\tau_2]_\rho \end{aligned} \quad (22)$$

By the operational semantics of LCVM, we now have that

$$\begin{aligned} &(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } f_1 = e_1 \text{ in } \lambda x_1. (\dots)) \\ &\xrightarrow{*_{L_1}} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, \text{let } f_1 = \lambda x_1^1. e_1^1 \text{ in } \lambda x_1. (\dots)) \\ &\xrightarrow{*_{L_1}} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, [f_1 \mapsto \lambda x_1^1. e_1^1] \lambda x_1. (\dots)) \\ &= (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, \lambda x_1. (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1. e_1^1) (C_{\tau_1 \mapsto \tau_1} x_1)))) \\ &\not\rightarrow \end{aligned}$$

so $\mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}$ and

$$v_1 = \lambda x_1. (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1. e_1^1) (C_{\tau_1 \mapsto \tau_1} x_1)))$$

Then we show the goal by taking $W' = W', \mathsf{H}'_{1g} = \mathsf{H}'_{1g}, \mathsf{H}'_{2g} = \mathsf{H}'_{2g}, \mathsf{H}'_1 = \emptyset, \mathsf{H}'_2 = \emptyset$, and

$$v_2 = \lambda x_2. (C_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1. e_2^1) (C_{\tau_1 \mapsto \tau_1} x_2)))$$

To show the configuration with $\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}$ terminates, we have

$$\begin{aligned}
& (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } f_2 = e_2 \text{ in } \lambda x_2.(\dots)) \\
& \xrightarrow{*_{L_2}} (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } f_2 = \lambda x_2^1.e_2^1 \text{ in } \lambda x_1.(\dots)) \\
& \xrightarrow{*_{L_2}} (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, [f_2 \mapsto \lambda x_2^1.e_2^1] \lambda x_2.(\dots)) \\
& = (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, \lambda x_2. (C_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1.e_2^1) (C_{\tau_1 \mapsto \tau_1} x_2)))) \\
& \not\rightarrow_{L_2}
\end{aligned}$$

Then take $\mathsf{H}'_2 = \emptyset$, $\mathsf{H}'_{2+} = \mathsf{H}_{2+}^1$.

All that remains to show is that:

$$\begin{aligned}
& (W', (\emptyset, (\lambda x_1. (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1.e_1^1) (C_{\tau_1 \mapsto \tau_1} v_1))))), \\
& (\emptyset, (\lambda x_2. (C_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1.e_2^1) (C_{\tau_1 \mapsto \tau_1} v_2)))))) \\
& \in \mathcal{V}[\![(!\tau_1 \multimap \tau_2)]\!]_\rho
\end{aligned}$$

Expanding the definitions of $\mathcal{V}[\![(!\tau_1 \multimap \tau_2)]\!]_\rho$, $\mathcal{V}[\![\cdot]\!]_\rho$ (twice), and pushing substitutions, we are to show that

$$\begin{aligned}
& (W'', (\emptyset, (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1.e_1^1) (C_{\tau_1 \mapsto \tau_1} v_1^a)))), \\
& (\emptyset, (C_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1.e_2^1) (C_{\tau_1 \mapsto \tau_1} v_2^a)))) \\
& \in \mathcal{E}[\![\tau_2]\!]_\rho
\end{aligned}$$

given arbitrary worlds W'' such that $W' \sqsubseteq_{\emptyset, \emptyset, e_1^1, e_2^1} W''$ and arbitrary v_1^a, v_2^a such that

$(W'', (\emptyset, v_1^a), (\emptyset, v_2^a)) \in \mathcal{V}[\![\tau_1]\!]_\rho$. Expanding the definition of $\mathcal{E}[\![\cdot]\!]$, we are to show that

$$\begin{aligned}
& \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : MHeap. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\
& \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}_{2g'} : W' \wedge \\
& W'' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W'', L_1 \cup FL(\text{cod}(\mathsf{H}_{1+})), L_2 \cup FL(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\
& (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\![\tau]\!]_\rho \wedge \\
& (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1.e_1^1) (C_{\tau_1 \mapsto \tau_1} v_1^a)))) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \not\rightarrow_{L_2}
\end{aligned}$$

given arbitrary $L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : MHeap, \mathsf{H}_{1*}$ such that

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1.e_1^1) (C_{\tau_1 \mapsto \tau_1} v_1^a)))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

By Lemma C.0.14, we have that $(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, C_{\tau_1 \mapsto \tau_1} v_1^a) \xrightarrow{*_{L_1 \cup FL(e_1^1)}} (\mathsf{H}_{1*}^1, v_1^1) \not\rightarrow_{L_1 \cup FL(e_1^1)}$ for some H_{1*}^1, v_1^1 . Recall that $(W'', (\emptyset, v_1^a), (\emptyset, v_2^a)) \in \mathcal{V}[\![\tau_1]\!]_\rho$ by assumption, so $(W'', (\emptyset, v_1^a), (\emptyset, v_2^a)) \in \mathcal{E}[\![\tau_1]\!]_\rho$ by Lemma C.0.8. Then, appealing to the induction hypothesis that $\tau_1 \sim \tau_1$ is sound,

expanding the definition of $\mathcal{E}[\cdot]_\rho$, and specializing as appropriate, we have that $H'_{1*} = H'_{1g} \uplus H_{1+}$ and

$$(H_{2g+} \uplus H_{2+}, C_{\tau_1 \mapsto \tau_1} v^a_2) \xrightarrow{*_{L_2 \cup FL(e^1_2)}} (H'_{2g} \uplus H_{2+}, v^1_2) \not\rightarrow_{L_2 \cup FL(e^1_2)}$$

where H'_{1g}, H'_{2g} : W''' for some
 $W'' \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W'', L_1 \cup FL(e^1_1) \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(e^1_2) \cup FL(\text{cod}(H_{2+})))}$
 W''' such that

$$(W''', (\emptyset, v^1_1), (\emptyset, v^1_2)) \in \mathcal{V}[\tau_1]_\rho$$

Now, by the operational semantics of LCVM, we have that

$$\begin{aligned} & (H_{1g+} \uplus H_{1+}, (C_{\tau_2 \mapsto \tau_2} ((\lambda x^1_1.e^1_1) (C_{\tau_1 \mapsto \tau_1} v^a_1)))) \\ & \xrightarrow{*_{L_1}} (H'_{1g} \uplus H_{1+}, (C_{\tau_2 \mapsto \tau_2} ((\lambda x^1_1.e^1_1) v^1_1))) \\ & \xrightarrow{*_{L_1}} (H'_{1g} \uplus H_{1+}, (C_{\tau_2 \mapsto \tau_2} [x^1_1 \mapsto v^1_1]e^1_1)) \\ & \xrightarrow{*_{L_1}} (H_{1*}, v_1) \\ & \not\rightarrow_{L_1} \end{aligned}$$

Then applying Lemma C.0.14 again, we have that $(H'_{1g} \uplus H_{1+}, [x^1_1 \mapsto v^1_1]e^1_1) \xrightarrow{*_{L_1}} (H^2_{1*}, v^2_1) \not\rightarrow_{L_1}$ for some H^2_{1*}, v^2_1 . Since $(W''', (\emptyset, v^1_1), (\emptyset, v^1_2)) \in \mathcal{V}[\tau_1]_\rho$ and $W' \sqsubseteq_{\emptyset, \emptyset, e^1_1, e^1_2} W'''$ (by Lemma C.0.2), we have $(W''', (\emptyset, [x^1_1 \mapsto v^1_1]e^1_1), (\emptyset, [x^2_2 \mapsto v^1_2]e^1_2)) \in \mathcal{E}[\tau_2]_\rho$ by (22). Expanding the definition of $\mathcal{E}[\cdot]_\rho$, we have that $H^2_{1*} = H''_{1g} \uplus H_{1+}$ and

$$(H'_{2g} \uplus H_{2+}, [x^2_2 \mapsto v^2_2]e^1_2) \xrightarrow{*_{L_2}} (H''_{2g} \uplus H_{2+}, v^2_2) \not\rightarrow_{L_2}$$

where H''_{1g}, H''_{2g} : W'''' for some
 $W'''' \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W''', L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))}$
 W'''' such that

$$(W''', (\emptyset, v^2_1), (\emptyset, v^2_2)) \in \mathcal{V}[\tau_2]_\rho$$

Now, by the operational semantics of LCVM, we have that

$$\begin{aligned} & (H'_{1g} \uplus H_{1+}, (C_{\tau_2 \mapsto \tau_2} [x^1_1 \mapsto v^1_1]e^1_1)) \xrightarrow{*_{L_1}} (H''_{1g} \uplus H_{1+}, (C_{\tau_2 \mapsto \tau_2} v^2_1)) \\ & \xrightarrow{*_{L_1}} (H_{1*}, v_1) \\ & \not\rightarrow_{L_1} \end{aligned}$$

Recall that $(W''', (\emptyset, v^2_1), (\emptyset, v^2_2)) \in \mathcal{V}[\tau_2]_\rho$, so $(W''', (\emptyset, v^2_1), (\emptyset, v^2_2)) \in \mathcal{E}[\tau_2]_\rho$ by Lemma C.0.8. Then, appealing to the induction hypothesis that $\tau_2 \sim \tau_2$ is sound, expanding

the definition of $\mathcal{E}[\cdot]_\rho$, and specializing as appropriate, we have that $H_{1*} = H''_{1g} \uplus H''_1 \uplus H_{1+}$ and

$$(H''_{2g} \uplus H_{2+}, C_{\tau_2 \mapsto \tau_2} v_2^2) \xrightarrow{*_{L_2}} (H'''_{2g} \uplus H''_2 \uplus H_{2+}, v_2) \not\rightarrow_{L_2}$$

where H''_{1g}, H''_{2g} : W'''' for some $W'''' \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W''', L_1 \cup \text{FL}(\text{cod}(H_{1+})), L_2 \cup \text{FL}(\text{cod}(H_{2+})))} W'''''$ such that

$$(W''''', (H''_1, v_1), (H''_2, v_2)) \in \mathcal{V}[\tau_2]_\rho$$

Then we show the goal by taking $W' = W''''', H'_1 = H''_1, H'_2 = H''_2, H'_{1g} = H''_{1g}$, and $H'_{2g} = H''_{2g}$. Finally, to show the configuration with $H_{2g+} \uplus H_{2+}$ terminates, we have:

$$\begin{aligned} & (H_{2g+} \uplus H_{2+}, (C_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1. e_2^1) (C_{\tau_1 \mapsto \tau_1} v_2^a)))) \\ & \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, (C_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1. e_2^1) v_2^1))) \\ & \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, (C_{\tau_2 \mapsto \tau_2} [x_2^1 \mapsto v_2^1] e_2^1)) \\ & \xrightarrow{*_{L_2}} (H''_{2g} \uplus H_{2+}, (C_{\tau_2 \mapsto \tau_2} v_2^2)) \\ & \xrightarrow{*_{L_2}} (H'''_{2g} \uplus H''_2 \uplus H_{2+}, v_2) \\ & \not\rightarrow_{L_2} \end{aligned}$$

2. We are to show that

$$\begin{aligned} & \forall (W, (\emptyset, e_1), (\emptyset, e_2)) \in \mathcal{E}[\mathbf{!}(!\tau_1 \multimap \tau_2)]_\rho. \\ & (W, (\emptyset, C_{\mathbf{!}(!\tau_1 \multimap \tau_2) \mapsto \tau_1 \rightarrow \tau_2} e_1), (\emptyset, C_{\mathbf{!}(!\tau_1 \multimap \tau_2) \mapsto \tau_1 \rightarrow \tau_2} e_2)) \in \mathcal{E}[\tau_1 \rightarrow \tau_2]_\rho \end{aligned}$$

Expanding the definition of $C_{\mathbf{!}(!\tau_1 \multimap \tau_2) \mapsto \tau_1 \rightarrow \tau_2} e_1$, we are to show that

$$(W, (\emptyset, \text{let } f_1 = e_1 \text{ in } \lambda x_1. (C_{\tau_2 \mapsto \tau_2} (\dots))), (\emptyset, \text{let } f_2 = e_2 \text{ in } \lambda x_2. (C_{\tau_2 \mapsto \tau_2} (\dots)))) \in \mathcal{E}[\tau_1 \rightarrow \tau_2]_\rho$$

given arbitrary e_1, e_2 such that $(\emptyset, e_1, \emptyset, e_2) \in \mathcal{E}[\mathbf{!}(!\tau_1 \multimap \tau_2)]_\rho$. Expanding the definition of $\mathcal{E}[\cdot]_\rho$, we are to show that

$$\begin{aligned} & \exists H'_{1g}. \forall H_{2+} : MHeap. \exists W', H'_{2g}, v_2. \\ & H_{1*} = H'_{1g} \uplus H'_1 \uplus H_{1+} \wedge H'_{1g}, H_{2g'} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(H_{1+})), L_2 \cup \text{FL}(\text{cod}(H_{2+})))} W' \wedge \\ & (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]_\rho \wedge \\ & (H_{2g+} \uplus H_2 \uplus H_{2+}, \text{let } f_2 = e_2 \text{ in } \lambda x_2. (C_{\tau_2 \mapsto \tau_2} (\dots))) \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, v_2) \not\rightarrow_{L_2} \end{aligned}$$

given arbitrary $W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : M\mathsf{Heap}, \mathsf{H}_{1*}$ such that

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } f_1 = e_1 \text{ in } \lambda x_1. (C_{\tau_2 \mapsto \tau_2} (\dots))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\sim_{L_1}$$

By Lemma C.0.14, we have that $\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, e_1 \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}^1, v_1^1) \not\sim_{L_1}$ for some H_{1*}^1, v_1^1 . Expanding the definition of $\mathcal{E}[\![!(\mathsf{!}\tau_1 \multimap \tau_2)]\!]_\rho$ in the premise and specializing where appropriate, we have that $\mathsf{H}_{1*}^1 = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}'_{1+}$ and

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, e_2) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2^1)$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some $W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+}), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+}))))} W'$ such that

$$(W', (\mathsf{H}'_1, v_1^1), (\mathsf{H}'_2, v_2^1)) \in \mathcal{V}[\![!(\mathsf{!}\tau_1 \multimap \tau_2)]\!]_\rho$$

Expanding the definition of $\mathcal{V}[\![!\cdot]\!]$ (twice) and $\mathcal{V}[\![\cdot \multimap \cdot]\!]$, we have that

$$\begin{aligned} v_1^1 &= \lambda x_1^1. e_1^1 \wedge v_2^1 = \lambda x_2^1. e_2^1 \wedge \mathsf{H}'_1 = \emptyset \wedge \mathsf{H}'_2 = \emptyset \\ \forall W''. W' \sqsubseteq_{\emptyset, \emptyset, e_1^1, e_2^1} W'' &\implies \\ \forall (W'', (\emptyset, v_1^a), (\emptyset, v_2^a)) \in \mathcal{V}[\![\tau_1]\!]_\rho. (\emptyset, [x_1^1 \mapsto v_1^a]e_1^1, \emptyset, [x_2^1 \mapsto v_2^a]e_2^1) &\in \mathcal{E}[\![\tau_2]\!]_\rho \end{aligned} \tag{23}$$

where we associate empty heaps with the v_i^a because the tuple comes from $\mathcal{V}[\![\mathsf{!}\tau_1]\!]_\rho$. By the operational semantics of LCV, we now have that

$$\begin{aligned} &(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } f_1 = e_1 \text{ in } \lambda x_1. (\dots)) \\ &\xrightarrow{*_{L_1}} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, \text{let } f_1 = \lambda x_1^1. e_1^1 \text{ in } \lambda x_1. (\dots)) \\ &\xrightarrow{*_{L_1}} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, [f_1 \mapsto \lambda x_1^1. e_1^1] \lambda x_1. (\dots)) \\ &= (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, \lambda x_1. (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1. e_1^1) (C_{\tau_1 \mapsto \tau_1} x_1)))) \\ &\not\sim_{L_1} \end{aligned}$$

so $\mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}$ and

$$v_1 = \lambda x_1. (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1. e_1^1) (C_{\tau_1 \mapsto \tau_1} x_1)))$$

Then we show the goal by taking $W' = W'$, $\mathsf{H}'_{1g} = \mathsf{H}'_{1g}$, $\mathsf{H}'_{2g} = \mathsf{H}'_{2g}$ and

$$v_2 = \lambda x_2. (C_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1. e_2^1) (C_{\tau_1 \mapsto \tau_1} x_2)))$$

To show the configuration $\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}$ terminates, we have

$$\begin{aligned}
& (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } f_2 = e_2 \text{ in } \lambda x_2.(\dots)) \\
& \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, \text{let } f_2 = \lambda x_2^1.e_2^1 \text{ in } \lambda x_1.(\dots)) \\
& \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, [f_2 \mapsto \lambda x_2^1.e_2^1] \lambda x_2.(\dots)) \\
& = (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, \lambda x_2. (C_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1.e_2^1) (C_{\tau_1 \mapsto \tau_1} x_2)))) \\
& \not\rightarrow_{L_2}
\end{aligned}$$

All that remains to show is that

$$\begin{aligned}
& (W', (\emptyset, (\lambda x_1. (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1.e_1^1) (C_{\tau_1 \mapsto \tau_1} v_1))))), \\
& (\emptyset, (\lambda x_2. (C_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1.e_2^1) (C_{\tau_1 \mapsto \tau_1} v_2)))))) \\
& \in \mathcal{V}[\![\tau_1 \rightarrow \tau_2]\!]_\rho
\end{aligned}$$

Expanding the definition of $\mathcal{V}[\![\tau_1 \rightarrow \tau_2]\!]_\rho$ and pushing substitutions, we are to show that

$$\begin{aligned}
& (W'', (\emptyset, (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1.e_1^1) (C_{\tau_1 \mapsto \tau_1} v_1^a)))), \\
& (\emptyset, (C_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1.e_2^1) (C_{\tau_1 \mapsto \tau_1} v_2^a)))))) \\
& \in \mathcal{E}[\![\tau_2]\!]_\rho
\end{aligned}$$

given arbitrary worlds W'' such that $W' \sqsubseteq_{\emptyset, \emptyset, e_1^1, e_2^1} W''$ and v_1^a, v_2^a such that $(W'', (\emptyset, v_1^a), (\emptyset, v_2^a)) \in \mathcal{V}[\![\tau_1]\!]_\rho$. Expanding the definition of $\mathcal{E}[\![\cdot]\!]$, we are to show that

$$\begin{aligned}
& \exists \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : MHeap. \exists W', \mathsf{H}'_{2g}, v_2. \\
& \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}_{2g'} : W' \wedge \\
& W'' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\
& (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\tau_2]\!]_\rho \wedge \\
& (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1.e_1^1) (C_{\tau_1 \mapsto \tau_1} v_1^a)))) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, v_2) \not\rightarrow_{L_2}
\end{aligned}$$

given arbitrary $L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : MHeap, \mathsf{H}_{1*}$ such that

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, (C_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1.e_2^1) (C_{\tau_1 \mapsto \tau_1} v_2^a)))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

By Lemma C.0.14, we have that $(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, C_{\tau_1 \mapsto \tau_1} v_1^a) \xrightarrow{*_{L_1 \cup \text{FL}(e_1^1)}} (\mathsf{H}_{1*}^1, v_1^1) \not\rightarrow_{L_1 \cup \text{FL}(e_1^1)}$ for some H_{1*}^1, v_1^1 . Recall that $(W'', (\emptyset, v_1^a), (\emptyset, v_2^a)) \in \mathcal{V}[\![\tau_1]\!]_\rho$ by assumption, so $(W'', (\emptyset, v_1^a), (\emptyset, v_2^a)) \in \mathcal{E}[\![\tau_1]\!]_\rho$ by Lemma C.0.8. Then, appealing to the inductive hypothesis that $\tau_1 \sim \tau_1$ is sound, expanding the

definition of $\mathcal{E}[\cdot]_\rho$, and specializing as appropriate, we have that $H_{1g}^1 = H'_{1g} \uplus H'_1 \uplus H_{1+}$ and

$$(H_{2g+} \uplus H_{2+}, C_{\tau_2 \mapsto \tau_2} v_2^a) \xrightarrow{*}_{L_2 \cup FL(e_2^1)} (H'_{2g} \uplus H'_2 \uplus H_{2+}, v_2^1) \not\rightarrow_{L_2 \cup FL(e_2^1)}$$

where $H'_{1g}, H'_{2g} : W'''$ for some $W'' \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W'', L_1 \cup FL(e_1^1) \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(e_2^1) \cup FL(\text{cod}(H_{2+})))} W'''$ such that

$$(W''', (H'_1, v_1^1), (H'_2, v_2^1)) \in \mathcal{V}[\tau_1]_\rho$$

Since $\tau_1 \in \text{DUPLICABLE}$, expanding the definition of DUPLICABLE and $\mathcal{V}[\cdot]$. reveals that we have $H'_1 = H'_2 = \emptyset$.

Now, by the operational semantics of LCVM, we have that

$$\begin{aligned} & (H_{1g+} \uplus H_{1+}, (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1.e_1^1) (C_{\tau_1 \mapsto \tau_1} v_1^a)))) \\ & \xrightarrow{*}_{L_1} (H'_{1g} \uplus H_{1+}, (C_{\tau_2 \mapsto \tau_2} ((\lambda x_1^1.e_1^1) v_1^1))) \\ & \xrightarrow{*}_{L_1} (H'_{1g} \uplus H_{1+}, (C_{\tau_2 \mapsto \tau_2} [x_1^1 \mapsto v_1^1]e_1^1)) \\ & \xrightarrow{*}_{L_1} (H_{1*}, v_1) \\ & \not\rightarrow_{L_1} \end{aligned}$$

Then applying Lemma C.0.14 again, we have that $(H'_{1g} \uplus H_{1+}, [x_1^1 \mapsto v_1^1]e_1^1) \xrightarrow{*}_{L_1} (H_{1*}^2, v_1^2) \not\rightarrow_{L_1}$ for some H_{1*}^2, v_1^2 . Since $(W''', (\emptyset, v_1^1), (\emptyset, v_2^1)) \in \mathcal{V}[\tau_1]_\rho$ and $W' \sqsubseteq_{\emptyset, \emptyset, e_1^1, e_2^1} W'''$ (by Lemma C.0.2), we have $(W''', (\emptyset, [x_1^1 \mapsto v_1^1]e_1^1), (\emptyset, [x_2^1 \mapsto v_2^1]e_2^1)) \in \mathcal{E}[\tau_2]_\rho$ by (23). Expanding the definition of $\mathcal{E}[\cdot]_\rho$, we have that $H_{1*}^2 = H'_{1g} \uplus H''_1 \uplus H_{1+}$ and

$$(H_{2g+} \uplus H_{2+}, [x_2^1 \mapsto v_2^1]e_2^1) \xrightarrow{*}_{L_2} (H''_{2g} \uplus H''_2 \uplus H_{2+}, v_2^2) \not\rightarrow_{L_2}$$

where $H''_{1g}, H''_{2g} : W''''$ for some $W'''' \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W''', L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))} W'''$ such that

$$(W''', (H''_1, v_1^2), (H''_2, v_2^2)) \in \mathcal{V}[\tau_2]_\rho$$

Now, by the operational semantics of LCVM, we have that

$$\begin{aligned} & (H'_{1g+} \uplus H_{1+}, (C_{\tau_2 \mapsto \tau_2} [x_1^1 \mapsto v_1^1]e_1^1)) \xrightarrow{*} (H'_{1g} \uplus H''_1 \uplus H_{1+}, (C_{\tau_2 \mapsto \tau_2} v_1^2)) \\ & \xrightarrow{*} (H_{1*}, v_1) \\ & \not\rightarrow \end{aligned}$$

Recall that $(W''', (\mathsf{H}_1'', \mathsf{v}_1^2), (\mathsf{H}_2'', \mathsf{v}_2^2)) \in \mathcal{V}[\![\tau_2]\!]_\rho$, so $(W''', (\mathsf{H}_1'', \mathsf{v}_1^2), (\mathsf{H}_2'', \mathsf{v}_2^2)) \in \mathcal{E}[\![\tau_2]\!]_\rho$ by Lemma C.0.8. Then, appealing to the inductive hypothesis that $\tau_2 \sim \tau_2$ is sound, expanding the definition of $\mathcal{E}[\![\cdot]\!]_\rho$, and specializing as appropriate, we have that $\mathsf{H}_{1g}''' = \mathsf{H}_{1g}'' \uplus \mathsf{H}_{1+}$ and

$$(\mathsf{H}_{2g}'' \uplus \mathsf{H}_2'' \uplus \mathsf{H}_{2+}, \mathsf{C}_{\tau_2 \mapsto \tau_2} \mathsf{v}_2^2) \xrightarrow{*_{L_2}} (\mathsf{H}_{2g}''' \uplus \mathsf{H}_{2+}, \mathsf{v}_2) \not\rightarrow_{L_2}$$

where $\mathsf{H}_{1g}''', \mathsf{H}_{2g}'''$: W''''' for some $W''''' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W''', L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W''''$ such that

$$(W''''', (\emptyset, \mathsf{v}_1), (\emptyset, \mathsf{v}_2)) \in \mathcal{V}[\![\tau_2]\!]_\rho$$

Then we show the goal by taking $W' = W''''', \mathsf{H}'_{1g} = \mathsf{H}_{1g}''', \mathsf{H}'_{2g} = \mathsf{H}_{2g}'''$, and $\mathsf{v}_2 = \mathsf{v}_2$. For showing the configuration with $\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}$ terminates, we have

$$\begin{aligned} & (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, (\mathsf{C}_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1. e_2^1) \ (\mathsf{C}_{\tau_1 \mapsto \tau_1} \mathsf{v}_2^a)))) \\ & \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, (\mathsf{C}_{\tau_2 \mapsto \tau_2} ((\lambda x_2^1. e_2^1) \ \mathsf{v}_2^1))) \\ & \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, (\mathsf{C}_{\tau_2 \mapsto \tau_2} [x_2^1 \mapsto \mathsf{v}_2^1] e_2^1)) \\ & \xrightarrow{*_{L_2}} (\mathsf{H}_{2g}'' \uplus \mathsf{H}_2'' \uplus \mathsf{H}_{2+}, (\mathsf{C}_{\tau_2 \mapsto \tau_2} \mathsf{v}_2^2)) \\ & \xrightarrow{*_{L_2}} (\mathsf{H}_{2g}''' \uplus \mathsf{H}_{2+}, \mathsf{v}_2) \\ & \not\rightarrow_{L_2} \end{aligned}$$

`ref` $\tau \sim \exists \zeta. \mathbf{cap} \zeta \tau \otimes !\mathbf{ptr} \zeta$

1. For the first direction, we show that

$$\begin{aligned} & \forall (W, (\mathsf{H}_1, \mathsf{e}_1), (\mathsf{H}_2, \mathsf{e}_2)) \in \mathcal{E}[\![\mathbf{ref} \ \tau]\!]_\rho. \\ & (W, (\mathsf{H}_1, \mathsf{C}_{\mathbf{ref} \ \tau \mapsto \exists \zeta. \mathbf{cap} \zeta \tau \otimes !\mathbf{ptr} \zeta} (\mathsf{e}_1)), \\ & (\mathsf{H}_2, \mathsf{C}_{\mathbf{ref} \ \tau \mapsto \exists \zeta. \mathbf{cap} \zeta \tau \otimes !\mathbf{ptr} \zeta} (\mathsf{e}_2))) \in \mathcal{E}[\![\exists \zeta. \mathbf{cap} \zeta \tau \otimes !\mathbf{ptr} \zeta]\!]_\rho \end{aligned}$$

where we have, by our induction hypothesis, that we can convert τ to τ .

We first expand the conversions, noting that the terms in question are:

$$\mathbf{let} \ x_\ell = \mathbf{alloc} \ \mathsf{C}_{\tau \mapsto \tau} (!\mathsf{e}_i) \ \mathbf{in} \ (((), x_\ell)$$

Expanding the definition of $\mathcal{E}[\![\exists \zeta. \mathbf{cap} \zeta \tau \otimes !\mathbf{ptr} \zeta]\!]_\rho$, we see that what we need to show is that:

$$\begin{aligned}
& \exists H'_1, H'_{1g}. \forall H_{2+} : MHeap. \exists H'_2, W', H'_{2g}, v_2. \\
& H_{1*} = H'_{1g} \uplus H'_1 \uplus H_{1+} \wedge H'_{1g}, H_{2g'} : W' \wedge \\
& W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))} W' \wedge \\
& (W', (H'_1, v_1), (H'_2, v_2)) \in \mathcal{V}[\exists \zeta. \text{cap } \zeta \tau \otimes !\text{ptr } \zeta]_\rho \wedge \\
& (H_{2g+} \uplus H_2 \uplus H_{2+}, \text{let } x_\ell = \text{alloc } C_{\tau \mapsto \tau}(!e_2) \text{ in } (((), x_\ell)) \xrightarrow{*}_{L_2} (H'_{2g} \uplus H'_2 \uplus H_{2+}, v_2)) \not\rightarrow_{L_2}
\end{aligned}$$

given arbitrary $L_1, L_2, H_{1g+}, H_{2g+} : W, v_1, H_{1+} : MHeap, H_{1*}$, such that

$$(H_{1g+} \uplus H_1 \uplus H_{1+}, \text{let } x_\ell = \text{alloc } C_{\tau \mapsto \tau}(!e_1) \text{ in } (((), x_\ell)) \xrightarrow{*}_{L_1} (H_{1*}, v_1) \not\rightarrow_{L_1}$$

By Lemma C.0.14, we have that $(H_{1g+} \uplus H_1 \uplus H_{1+}, e_1) \xrightarrow{*}_{L_1} (H_{1*}^1, v_1^1) \not\rightarrow_{L_1}$ for some H_{1*}^1, v_1^1 .

Our induction hypothesis, appropriately instantiated and simplified, then tells us that

$$\begin{aligned}
& \exists W^1 H_{1g+}^1. H_{1*}^1 = H_{1g+}^1 \uplus H_{1+} \wedge H_{1g+}^1, H_{2g+}^1 : W^1 \wedge \\
& W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))} W^1 \wedge \\
& \forall H_{2+}. \exists v_2^1, H_{2g+}^1. (H_{2g+}^1 \uplus H_{2+}, e_2) \xrightarrow{*}_{L_2} (H_{2g+}^1 \uplus H_{2+}, v_2^1) \not\rightarrow_{L_2} \\
& \wedge (W^1, (\emptyset, v_1^1), (\emptyset, v_2^1)) \in \mathcal{V}[\text{ref } \tau]_\rho
\end{aligned} \tag{24}$$

This means, in particular, that v_1^1 and v_2^1 are locations, call them ℓ_1 and ℓ_2 , and heap satisfaction means that $H_{ig+}^1(\ell_i)$ are values (call them v_1 and v_2) related by $\mathcal{V}[\tau]_\rho$. Also, since the value relation for **MiniML** doesn't allow heap fragments, this means that the locations in H_i have been consumed.

Thus, we can instantiate our induction hypothesis for $C_{\tau \mapsto \tau}$ with v_i and get reductions that we can use to again appeal to Lemma C.0.14, with. In particular, we know that we proceed with the following reductions thus far (with related ones on the other side):

$$\begin{aligned}
& (H_{1g+} \uplus H_1 \uplus H_{1+}, \text{let } x_\ell = \text{alloc } C_{\tau \mapsto \tau}(!e_1) \text{ in } (((), x_\ell)) \\
& \xrightarrow{*}_{L_1} (H_{1g+}^1 \uplus H_{1+}, \text{let } x_\ell = \text{alloc } C_{\tau \mapsto \tau}(!\ell_1) \text{ in } (((), x_\ell)) \\
& \xrightarrow{*}_{L_1} (H_{1g+}^1 \uplus H_{1+}, \text{let } x_\ell = \text{alloc } C_{\tau \mapsto \tau}(v_1) \text{ in } (((), x_\ell)) \\
& \xrightarrow{*}_{L_1} (H_{1g+}^2 \uplus H_1^2 \uplus H_{1+}, \text{let } x_\ell = \text{alloc } v_1 \text{ in } (((), x_\ell)))
\end{aligned}$$

Where we know we have

$$W^1 \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W^1, L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))} W^2$$

$$\begin{aligned} \mathsf{H}_{1g+}^2, \mathsf{H}_{2g+}^2 : W^2 \\ (W^2, (\mathsf{H}_1^2, v_{1'}), (\mathsf{H}_2^2, v_{2'})) \in \mathcal{V}[\![\tau]\!]_\rho \end{aligned}$$

Now, we can proceed with the remaining reductions, after which we have to complete all our original obligations at the resulting future world. The reductions are:

$$\begin{aligned} & (\mathsf{H}_{1g+}^2 \uplus \mathsf{H}_1^2 \uplus \mathsf{H}_{1+}, \text{let } x_\ell = \text{alloc } v_{1'} \text{ in } (((), x_\ell))) \\ \xrightarrow{*}{L_1} & (\mathsf{H}_{1g+}^2 \uplus \{\ell_{1'} \xrightarrow{m} v'_1\} \uplus \mathsf{H}_1^2 \uplus \mathsf{H}_{1+}, \text{let } x_\ell = \ell_{1'} \text{ in } (((), x_\ell))) \\ \xrightarrow{*}{L_1} & (\mathsf{H}_{1g+}^2 \uplus \mathsf{H}_1^2 \uplus \{\ell_{1'} \xrightarrow{m} v'_1\} \uplus \mathsf{H}_{1+}, (((), \ell_{1'}))) \end{aligned}$$

Where the latter has clearly terminated to a value. We know, analogously, that the other side will run in the same way, terminating with the configuration:

$$(\mathsf{H}_{2g+}^2 \uplus \mathsf{H}_2^2 \uplus \{\ell_{2'} \xrightarrow{m} v'_2\} \uplus \mathsf{H}_{2+}, (((), \ell_{2'})))$$

The world we choose is simply W^2 — our manual allocation doesn't change the garbage collected fragments of the heap (indicated by name with a subscript g), and thus the same world and heap satisfaction still holds. Since we already have the values to which both sides terminated, our remaining obligation is to show:

$$\begin{aligned} & (W^2, (\mathsf{H}_1^2 \uplus \{\ell_{1'} \xrightarrow{m} v'_1\}, (((), \ell_{1'}))), \\ & (\mathsf{H}_2^2 \uplus \{\ell_{2'} \xrightarrow{m} v'_2\}, (((), \ell_{2'})))) \in \mathcal{V}[\![\exists \zeta. \mathbf{cap} \zeta \tau \otimes !\mathbf{ptr} \zeta]\!]_\rho \end{aligned}$$

Expanding the definition of $\mathcal{V}[\![\exists \zeta. \tau]\!]_\rho$, it suffices to show that:

$$\begin{aligned} & (W^2, (\mathsf{H}_1^2 \uplus \{\ell_{1'} \xrightarrow{m} v_{1'}\}, (((), \ell_{1'}))), (\mathsf{H}_2^2 \uplus \{\ell_{2'} \xrightarrow{m} v_{2'}\}, (((), \ell_{2'})))) \\ & \in \mathcal{V}[\![\mathbf{cap} \zeta \tau \otimes !\mathbf{ptr} \zeta]\!]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_{1'}, \ell_{2'})]} \end{aligned}$$

Now, we turn to the definition of $\mathcal{V}[\![\tau_1 \otimes \tau_2]\!]_\rho$, which says we need to split the heaps and then prove, using the split (we use empty heaps on one side of our split), the following two obligations:

$$\begin{aligned} & (W^2, (\mathsf{H}_1^2 \uplus \{\ell_{1'} \xrightarrow{m} v'_1\}, ()), (\mathsf{H}_2^2 \uplus \{\ell_{2'} \xrightarrow{m} v'_2\}, ())) \in \mathcal{V}[\![\mathbf{cap} \zeta \tau]\!]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_{1'}, \ell_{2'})]} \\ & (W^2, (\emptyset, \ell_{1'}), (\emptyset, \ell_{2'})) \in \mathcal{V}[\![!\mathbf{ptr} \zeta]\!]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_{1'}, \ell_{2'})]} \end{aligned}$$

The second one holds trivially, since $!$ requires empty heaps and the **ptr** type requires that the locations are mapped to by the type environment, which they are. The first is only slightly less trivial: it requires, first, that $\rho[\mathbf{L3}(\zeta) \mapsto (\ell_1, \ell_2)](\zeta) = (\ell_1, \ell_2)$, which it clearly does. Then, that those locations map to values in the heap, and that, for the rest of the heap, the following holds:

$$(W^2, (\mathsf{H}_1^2, v_{1'}), (\mathsf{H}_2^2, v_{21})) \in \mathcal{V}[\![\tau]\!]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_1, \ell_2)]}$$

This holds by earlier assumption on $v_{1'}$ and $v_{2'}$ and weakening in the type substitution.

2. The other direction, requires that we show

$$\begin{aligned} \forall (W, (\mathsf{H}_1, e_1), (\mathsf{H}_2, e_2)) \in \mathcal{E}[\![\exists \zeta. \mathsf{cap} \zeta \tau \otimes !\mathsf{ptr} \zeta]\!]_\rho. \\ (W, (\mathsf{H}_1, C_{\exists \zeta. \mathsf{cap} \zeta \tau \otimes !\mathsf{ptr} \zeta \mapsto \mathsf{ref} \tau}(e_1)), (\mathsf{H}_2, C_{\exists \zeta. \mathsf{cap} \zeta \tau \otimes !\mathsf{ptr} \zeta \mapsto \mathsf{ref} \tau}(e_2))) \\ \in \mathcal{E}[\![\mathsf{ref} \tau]\!]_\rho \end{aligned}$$

where we have, by our induction hypothesis, that we can convert τ to τ .

We first expand the conversions, noting that the terms in question are:

$$\text{let } x_\ell = \text{snd } e_i \text{ in let } _- = (x_\ell := C_{\tau \mapsto \tau}(!x_\ell)) \text{ in gcmov } x_\ell$$

As before, we expand the definition our obligation, in this case $\mathcal{E}[\![\mathsf{ref} \tau]\!]_\rho$, to show that what we need is that:

$$\begin{aligned} & \exists W', \mathsf{H}'_{1g}, \mathsf{H}'_{2g}, \forall \mathsf{H}_{2+}. \exists v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\mathsf{ref} \tau]\!]_\rho \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } x_\ell = \text{snd } e_2 \text{ in let } _- = (x_\ell := C_{\tau \mapsto \tau}(!x_\ell)) \text{ in gcmov } x_\ell) \\ & \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, v_2) \rightsquigarrow_{L_1} \end{aligned}$$

given arbitrary $L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_{1+}, \mathsf{H}_{1*}$, such that

$$\begin{aligned} & (\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } x_\ell = \text{snd } e_1 \text{ in let } _- = (x_\ell := C_{\tau \mapsto \tau}(!x_\ell)) \text{ in gcmov } x_\ell) \\ & \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \rightsquigarrow_{L_1} \end{aligned}$$

We appeal to Lemma C.0.14, which tells us that $(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, e_1) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}^1, v_1^1) \rightsquigarrow_{L_1}$ for some H_{1*}^1, v_1^1 .

Our induction hypothesis, appropriately instantiated and simplified, then tells us that

$$\begin{aligned} & \exists H_1^1, H_{1g}^1, \forall H_{2+} : MHeap. \exists H_2^1, W', H_{2g}^1, v_2^1. \\ & H_{1*} = H_{1g}^1 \uplus H_1^1 \uplus H_{1+} \wedge H_{1g}^1, H_{2g'}^1 : W^1 \wedge \\ & W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))} W^1 \wedge \\ & (W^1, (H_1^1, v_1^1), (H_2^1, v_2^1)) \in \mathcal{V}[\exists \zeta. \text{cap } \zeta \tau \otimes !\text{ptr } \zeta]_\rho \wedge \\ & (H_{2g+} \uplus H_{2+}, e_1) \xrightarrow{*_{L_2}} (H_{2g}^1 \uplus H_2^1 \uplus H_{2+}, v_2^1) \rightsquigarrow \end{aligned} \quad (25)$$

In particular, that means that v_1^1 and v_2^1 have the form $(((), \ell_i)$, where the value relation means that the heap fragments map ℓ_i to a v_i . Note that H_i^1 is composed of $\{\ell_1 \xrightarrow{m} v_i\} \uplus H_i^{1'}$. This follows from the value relation.

If we continue evaluating our original terms, we step as follows:

$$\begin{aligned} & (H_{1g+} \uplus H_1 \uplus H_{1+}, \\ & \text{let } x_\ell = \text{snd } e_1 \text{ in let } _- = (x_\ell := C_{\tau \mapsto \tau}(!x_\ell) \text{ in gcmov } x_\ell) \xrightarrow{*_{L_1}} \\ & (H_{1g+}^1 \uplus \{\ell_1 \xrightarrow{m} v_i\} \uplus H_i^{1'} \uplus H_{1+}, \\ & \text{let } x_\ell = \text{snd } (((), \ell_1) \text{ in let } _- = (x_\ell := C_{\tau \mapsto \tau}(!x_\ell) \text{ in gcmov } x_\ell) \rightarrow_{L_1} \\ & (H_{1g+}^1 \uplus \{\ell_1 \xrightarrow{m} v_i\} \uplus H_i^{1'} \uplus H_{1+}, \\ & \text{let } x_\ell = \ell_1 \text{ in let } _- = (x_\ell := C_{\tau \mapsto \tau}(!x_\ell) \text{ in gcmov } x_\ell) \rightarrow_{L_1} \\ & (H_{1g+}^1 \uplus \{\ell_1 \xrightarrow{m} v_1\} \uplus H_i^{1'} \uplus H_{1+}, \text{let } _- = (\ell_1 := C_{\tau \mapsto \tau}(!\ell_1) \text{ in gcmov } \ell_1) \rightarrow_{L_1} \\ & (H_{1g+}^1 \uplus \{\ell_1 \xrightarrow{m} v_1\} \uplus H_i^{1'} \uplus H_{1+}, \text{let } _- = (\ell_1 := C_{\tau \mapsto \tau}(v_1) \text{ in gcmov } \ell_1) \end{aligned}$$

Since we know that v_1 was in the value relation at type τ , we can appeal to our induction hypothesis with the heap fragment $H_i^{1'}$ to get that $C_{\tau \mapsto \tau}(v_1)$ (and, correspondingly $C_{\tau \mapsto \tau}(v_2)$) are in the expression relation at $\mathcal{E}[\tau]_\rho$. That expression relation will tell us that once the term runs to a value, that heap fragment will be consumed.

This means, in particular, that we can combine Lemma C.0.14 with the definition of the expression relation to get that

$(H_{1g+}^1 \uplus \{\ell_1 \xrightarrow{m} v_1\} \uplus H_i^{1'} \uplus H_{1+}, C_{\tau \mapsto \tau}(v_1)) \xrightarrow{*_{L_1}} (H_{1g+}^2 \uplus \{\ell_1 \xrightarrow{m} v_1\} \uplus H_{1+}, v_1^2) \rightsquigarrow_{L_1}$ for some H_{1g+}^2, v_1^2 , where v_1^2 is related to a corresponding v_2^2 in $\mathcal{V}[\tau]_\rho$ at a world W^2 that is an extension of W^1 (note that all the other steps did not change the garbage collected portion of the heap, so the only changes happened during the conversion, and are thus guided by the expression relation that our induction hypothesis produces).

This means our final sequence of steps are:

$$\begin{aligned}
& (\mathsf{H}_{1g+}^1 \uplus \{\ell_1 \xrightarrow{m} v_1\} \uplus \mathsf{H}_i^{1'} \uplus \mathsf{H}_{1+}, \text{let } _ = (\ell_1 := \mathsf{C}_{\tau \mapsto \tau}(v_1)) \text{ in gcmov } \ell_1) \xrightarrow{*_{L_1}} \\
& (\mathsf{H}_{1g+}^2 \uplus \{\ell_1 \xrightarrow{m} v_1\} \uplus \mathsf{H}_{1+}, \text{let } _ = (\ell_1 := v_1^2) \text{ in gcmov } \ell_1) \rightarrow_{L_1} \\
& (\mathsf{H}_{1g+}^2 \uplus \{\ell_1 \xrightarrow{m} v_1^2\} \uplus \mathsf{H}_{1+}, \text{let } _ = () \text{ in gcmov } \ell_1) \rightarrow_{L_1} \\
& (\mathsf{H}_{1g+}^2 \uplus \{\ell_1 \xrightarrow{m} v_1^2\} \uplus \mathsf{H}_{1+}, \text{gcmov } \ell_1) \rightarrow_{L_1} \\
& (\mathsf{H}_{1g+}^2 \uplus \{\ell_1 \xrightarrow{gc} v_1^2\} \uplus \mathsf{H}_{1+}, \ell_1)
\end{aligned}$$

And in particular, we can relate our final values, ℓ_1 and ℓ_2 , at $\mathcal{V}[\![\text{ref } \tau]\!]_\rho$ at a world W^3 , which is W^2 extended with the mapping from (ℓ_1, ℓ_2) to $\mathcal{V}[\![\tau]\!]_\rho$. We note, critically, that the owned portion of the heap is now empty, a requirement of $\mathcal{V}[\![\tau]\!]_\rho$, having been moved into the garbage collected portion of the heap.

□

Lemma C.0.16 (Compat \mathbf{x}).

$$\Delta; !\Gamma; \Delta; \Gamma, \mathbf{x} : \tau \vdash \mathbf{x} \preceq \mathbf{x} : \tau$$

Proof. Expanding the definition of \preceq , \cdot^+ , and $\mathcal{E}[\![\cdot]\!]$. (noting via Lemma C.0.10 that $\mathsf{H}_1 = \mathsf{H}_2 = \emptyset$), we are to show that

$$\begin{aligned}
& \exists W' \mathsf{H}'_{1g} \mathsf{H}'_{2g} v_2. \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\
& W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \\
& \wedge (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\tau]\!]_\rho \wedge \\
& (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, \gamma_{\mathbf{L}}^2(\gamma_{\Gamma, \mathbf{x}: \tau}^2(x))) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g+} \uplus \mathsf{H}_{2+}, v_2) \not\rightarrow_{L_2} \\
& \tag{26}
\end{aligned}$$

given arbitrary $\rho, \gamma_{\mathbf{L}}, \gamma_{\Gamma, \mathbf{x}: \tau}, W, \mathsf{H}_{1g+}, \mathsf{H}_{2g+}, \mathsf{H}_{1+}, \mathsf{H}_{1*}, \mathsf{H}_{2+}, v_1, L_1, L_2$ such that $\rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!]$, $\rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!]$, $(W, \emptyset, \emptyset, \gamma_{\mathbf{L}}) \in \mathcal{G}[\![!\Gamma]\!]_\rho$,

$\gamma_{\Gamma, \mathbf{x}: \tau} \in \mathcal{G}[\![\Gamma, \mathbf{x} : \tau]\!]_\rho$,

$\mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W$ and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma, \mathbf{x}: \tau}^1(x))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

Expanding the definition of $\mathcal{G}[\![\cdot]\!]$, we have that

$$\gamma_{\Gamma, \mathbf{x}: \tau} = \gamma[x \mapsto (v_1, v_2)] \wedge \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho \wedge (W, (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\tau]\!]_\rho$$

so $\gamma_{\mathbf{L}}^i(\gamma_{\Gamma, \mathbf{x}: \tau}^i(x)) = v_i$. Then we have (26) by taking $W' = W$, $\mathsf{H}'_{1g} = \mathsf{H}_{1g+}$ and $\mathsf{H}'_{2g} = \mathsf{H}_{2g+}$ noting that configurations with values as programs do not step. □

Lemma C.0.17 (Compat $\text{()}\preceq\text{()}$).

$$\Delta; !\Gamma; \Delta; \Gamma \vdash () \preceq () : \text{unit}$$

Proof. Expanding the definition of \preceq , \cdot^+ , and $\mathcal{E}[\cdot]$. (noting via Lemma C.0.10 that $H_1 = H_2 = \emptyset$), we are to show that

$$\begin{aligned} & \exists W' H'_{1g} H'_{2g} v_2.H_{1*} = H'_{1g} \uplus H_{1+} \wedge H'_{1g}, H'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))} W' \\ & \wedge (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau]_\rho \wedge \\ & (H_{2g+} \uplus H_{2+}, \gamma_L^2(\gamma_\Gamma^2(()))) \xrightarrow{*_{L_2}} (H'_{2g+} \uplus H_{2+}, v_2) \not\rightarrow_{L_2} \end{aligned} \tag{27}$$

given arbitrary $\rho, \gamma_L, \gamma_\Gamma, W, H_{1g+}, H_{2g+}, H_{1+}, H_{1*}, H_{2+}, v_1, L_1, L_2$ such that $\rho.\text{L3} \in \mathcal{D}[\Delta]$, $\rho.\text{F} \in \mathcal{D}[\Delta]$, $(W, \emptyset, \emptyset, \gamma_L) \in \mathcal{G}[!F]_\rho$, $\gamma_\Gamma \in \mathcal{G}[\Gamma]_\rho$, $H_{1g+}, H_{2g+} : W$ and

$$(H_{1g+} \uplus H_{1+}, \gamma_L^1(\gamma_\Gamma^1(()))) \xrightarrow{*_{L_1}} (H_{1*}, v_1) \not\rightarrow_{L_1}$$

We can simplify the substitutions away, and note that the configuration $(H_{1g+} \uplus H_{1+}, ())$ does not step because $()$ is a value. Thus, we have (27) by taking $W' = W$, $H'_{1g} = H_{1g+}$ and $H'_{2g} = H_{2g+}$.

□

Lemma C.0.18 (Compat $\lambda x : \tau.e$). *If $\Delta; !\Gamma; \Delta; \Gamma, x : \tau_1 \vdash e \preceq e : \tau_2$, then*

$$\Delta; !\Gamma; \Delta; \Gamma, x : \tau_1 \vdash \lambda x : \tau_1.e \preceq \lambda x : \tau_1.e : \tau_1 \rightarrow \tau_2$$

Proof. Expanding the definition of \preceq , \cdot^+ , and $\mathcal{E}[\cdot]$. (noting via Lemma C.0.10 that $H_1 = H_2 = \emptyset$), we are to show that

$$\begin{aligned} & \exists W' H'_{1g} H'_{2g} v_2.H_{1*} = H'_{1g} \uplus H_{1+} \wedge H'_{1g}, H'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))} W' \\ & \wedge (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau]_\rho \wedge (H_{2g+} \uplus H_{2+}, \lambda x. \gamma_L^2(\gamma_\Gamma^2(e^+))) \xrightarrow{*} (H'_{2g+} \uplus H_{2+}, v_2) \not\rightarrow \end{aligned} \tag{28}$$

given arbitrary $\rho, \gamma_L, \gamma_\Gamma, W, H_{1g+}, H_{2g+}, H_{1+}, H_{1*}, H_{2+}, v_1, L_1, L_2$ such that $\rho.\text{L3} \in \mathcal{D}[\Delta]$, $\rho.\text{F} \in \mathcal{D}[\Delta]$, $(W, \emptyset, \emptyset, \gamma_L) \in \mathcal{G}[!F]_\rho$, $\gamma_\Gamma \in \mathcal{G}[\Gamma]_\rho$, $H_{1g+}, H_{2g+} : W$ and

$$(H_{1g+} \uplus H_{1+}, \lambda x. \gamma_L^1(\gamma_\Gamma^1(e^+))) \xrightarrow{*_{L_1}} (H_{1*}, v_1) \not\rightarrow$$

We show (28) by taking $W' = W$, $\mathsf{H}'_{1g} = \mathsf{H}_{1g+}$ and $\mathsf{H}'_{2g} = \mathsf{H}_{2g+}$, noting that configurations with values as programs do not step. It thus suffices to show:

$$(W, (\emptyset, \lambda x. \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(e^+))), (\emptyset, \lambda x. \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(e^+)))) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]_{\rho}$$

Expanding the definition of $\mathcal{V}[\tau_1 \rightarrow \tau_2]_{\rho}$ and pushing substitutions inside $\gamma_{\mathbf{L}}$, we are to show that

$$(W^*, (\emptyset, \gamma_{\mathbf{L}, x:\tau}^1(\gamma_{\Gamma}^1(v_{1a}, v_{2a}))(e^+)), (\emptyset, \gamma_{\mathbf{L}, x:\tau}^2(\gamma_{\Gamma}^2(v_{1a}, v_{2a}))(e^+))) \in \mathcal{E}[\tau_2]_{\rho}$$

given arbitrary v_{1a}, v_{2a} such that $W \sqsubseteq_{\emptyset, \emptyset, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(e^+)), \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(e^+))} W^*$ and $(W^*, (\emptyset, v_{1a}), (\emptyset, v_{2a})) \in \mathcal{V}[\tau_1]_{\rho}$. We have this by expanding the definition of \preceq in the premise and specializing where appropriate. \square

Lemma C.0.19 (Compat $e_1 e_2$). *If $\Delta; !\Gamma; \Delta; \Gamma \vdash e_1 \preceq e_1 : \tau_1 \rightarrow \tau_2$ and $\Delta; !\Gamma; \Delta; \Gamma \vdash e_2 \preceq e_2 : \tau_1$, then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash e_1 e_2 \preceq e_1 e_2 : \tau_2$$

Proof. Expanding the definition of \preceq , \cdot^+ , and $\mathcal{E}[\cdot]$. (noting via Lemma C.0.10 that $\mathsf{H}_1 = \mathsf{H}_2 = \emptyset$), we are to show that

$$\begin{aligned} & \exists W' \mathsf{H}'_{1g} \mathsf{H}'_{2g} v_2. \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \\ & \wedge (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau_2]_{\rho} \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, (\gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(e_1^+)) \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(e_2^+)))) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g+} \uplus \mathsf{H}_{2+}, v_2) \rightsquigarrow \end{aligned} \tag{29}$$

given arbitrary $\rho, \gamma_{\mathbf{L}}, \gamma_{\Gamma}, W, \mathsf{H}_{1g+}, \mathsf{H}_{2g+}, \mathsf{H}_{1+}, \mathsf{H}_{1*}, \mathsf{H}_{2+}, v_1, L_1, L_2$ such that $\rho.\mathbf{L}3 \in \mathcal{D}[\Delta]$, $\rho.\mathbf{F} \in \mathcal{D}[\Delta]$, $(W, \emptyset, \emptyset, \gamma_{\mathbf{L}}) \in \mathcal{G}[!]\Gamma]_{\rho}$, $\gamma_{\Gamma} \in \mathcal{G}[\Gamma]_{\rho}$,

$\mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W$ and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, (\gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(e_1^+)) \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(e_2^+)))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \rightsquigarrow$$

By Lemma C.0.14, we have that

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(e_1^+))) \xrightarrow{*_{L_1 \cup \text{FL}(\gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(e_2^+)))}} (\mathsf{H}_{1*}^1, v_1^1) \rightsquigarrow$$

for some H_{1*}^1, v_1^1 . Then expanding the definition of \preceq and $\mathcal{E}[\cdot]$. in the first premise and specializing where appropriate, we have that

$$\begin{aligned} \exists W^1 H_{1g}^1 H_{2g}^1 v_2^1. & H_{1*}^1 = H_{1g}^1 \uplus H_{1+} \wedge H_{1g}^1, H_{2g}^1 : W^1 \wedge \\ & W \sqsubseteq (\text{dom}(H_{1+}), \text{dom}(H_{2+})), \\ & \text{rchgclocs}(W, FL(\text{cod}(H_{1+})) \cup FL(\gamma_L^1(\gamma_\Gamma^1(e_2^+))) \cup L_1, \\ & \quad FL(\text{cod}(H_{2+})) \cup FL(\gamma_L^2(\gamma_\Gamma^2(e_2^+))) \cup L_2) \\ & \wedge (W^1, (\emptyset, v_1^1), (\emptyset, v_2^1)) \in \mathcal{V}[\tau_1 \rightarrow \tau_2]_\rho \wedge \\ & \forall H_{2+}. (H_{2g+} \uplus H_{2+}, \gamma_L^2(\gamma_\Gamma^2(e_1^+))) \xrightarrow{*_{L_2 \cup FL(\gamma_L^2(\gamma_\Gamma^2(e_2^+)))}} (H_{2g}^1 \uplus H_{2+}, v_2^1) \rightsquigarrow \end{aligned} \tag{30}$$

Expanding the definition of $\mathcal{V}[\tau_1 \rightarrow \tau_2]_\rho$, we have that

$$\begin{aligned} v_1^1 &= \lambda x_1.e_{1b} \wedge v_2^1 = \lambda x_2.e_{2b} \wedge \\ &\forall (W^{1*}, (\emptyset, v_{1a}), (\emptyset, v_{2a})) \in \mathcal{V}[\tau_1]_\rho. W^1 \sqsubseteq_{\emptyset, \emptyset, e_{1b}, e_{2b}} W^{1*} \\ &\wedge (W^{1*}, (\emptyset, [x_1 \mapsto v_{1a}]e_{1b}, \emptyset, [x_2 \mapsto v_{2a}]e_{2b}) \in \mathcal{E}[\tau_2]_\rho \end{aligned} \tag{31}$$

Proceeding to work on our second premise, by Lemma C.0.14, we have:

$$(H_{1g}^1 \uplus H_{1+}, \gamma_L^1(\gamma_\Gamma^1(e_2^+))) \xrightarrow{*_{L_1 \cup FL(e_{1b})}} (H_{1*}^2, v_1^2) \rightsquigarrow$$

for some H_{1*}^2, v_1^2 .

Then expanding the definition of \preceq and $\mathcal{E}[\cdot]$. in the second premise, noting due to Lemma C.0.3 that we can use W^1 , and specializing where appropriate, we have that

$$\begin{aligned} \exists W^2 H_{1g}^2 H_{2g}^2 v_2^2. & H_{1*}^2 = H_{1g}^2 \uplus H_{1+} \wedge H_{1g}^2, H_{2g}^2 : W^2 \wedge \\ & W^1 \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W^1, L_1 \cup FL(e_{1b}) \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(e_{2b}) \cup FL(\text{cod}(H_{2+})))} W^2 \\ & \wedge (W^2, (\emptyset, v_1^2), (\emptyset, v_2^2)) \in \mathcal{V}[\tau_1]_\rho \wedge \\ & \forall H_{2+}. (H_{2g}^2 \uplus H_{2+}, \gamma_L^2(\gamma_\Gamma^2(e_2^+))) \xrightarrow{*_{L_2 \cup FL(e_{2b})}} (H_{2g}^2 \uplus H_{2+}, v_2^2) \rightsquigarrow \end{aligned} \tag{32}$$

Now, we want to start putting things together. We appeal to (31), instantiating it with the values found in (30), taking W^{1*} to be W^2 . Thus we have $(W^2, (\emptyset, [x_1 \mapsto v_1^2]e_{1b}), (\emptyset, [x_2 \mapsto v_2^2]e_{2b})) \in \mathcal{E}[\tau_2]_\rho$.

Then, expanding the definition of $\mathcal{E}[\cdot]$ and specializing where appropriate, we have that

$$\begin{aligned} \exists W^3 H_{1g}^3 H_{2g}^3 v_2^3. H_{1*}^3 = H_{1g}^3 \uplus H_{1+} \wedge H_{1g}^3, H_{2g}^3 : W^3 \wedge \\ W^2 \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W^2, L_1 \cup \text{FL}(\text{cod}(H_{1+})), L_2 \cup \text{FL}(\text{cod}(H_{2+})))} W^3 \\ \wedge (W^3, (\emptyset, v_1^3), (\emptyset, v_2^3)) \in \mathcal{V}[\tau_2]_\rho \wedge \\ \forall H_{2+}. (H_{2g}^2 \uplus H_{2+}, [x_2 \mapsto v_2^2]e_{2b}) \xrightarrow{*_{L_2}} (H_{2g}^2 \uplus H_{2+}, v_2) \not\rightarrow \end{aligned} \tag{33}$$

Then we show (29) by taking $H_{1*} = H_{1g}^3 \uplus H_{1+}$ and $v_2 = v_2$. All that remains is to show that

$$\exists H'_{2g}. (H'_{2g} \uplus H_{2+}, (\gamma_L^2(\gamma_\Gamma^2(e_1^+)) \ \gamma_L^2(\gamma_\Gamma^2(e_2^+)))) \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, v_2) \not\rightarrow$$

Specializing where appropriate, we have that

$$\begin{aligned} & (H_{2g+} \uplus H_{2+}, (\gamma_L^2(\gamma_\Gamma^2(e_1^+)) \ \gamma_L^2(\gamma_\Gamma^2(e_2^+)))) \\ & \xrightarrow{*_{L_2 \cup \text{FL}(\gamma_L^2(\gamma_\Gamma^2(e_2^+)))}} (H_{2g}^2 \uplus H_{2+}, (\lambda x_2.e_{2b}) \ \gamma_L^2(\gamma_\Gamma^2(e_2^+))) \quad (\text{by 30}) \\ & \xrightarrow{*_{L_2 \cup \text{FL}(e_{2b})}} (H_{2g}^3 \uplus H_{2+}, (\lambda x_2.e_{2b}) \ v_2^2) \quad (\text{by 32}) \\ & \xrightarrow{*_{1_{L_2}}} (H_{2g}^2 \uplus H_{2+}, [x_2 \mapsto v_2^2]e_{2b}) \quad (\text{by LCVM}) \\ & \xrightarrow{*_{L_2}} (H_{2g}^3 \uplus H_{2+}, v_2) \quad (\text{by 33}) \\ & \not\rightarrow \quad (\text{values don't step}) \end{aligned}$$

□

Lemma C.0.20 (Compat $\Lambda\alpha.e$). *If $\Delta; !\Gamma; \Delta, \alpha; \Gamma \vdash e \preceq e : \tau$, then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash \Lambda\alpha.e \preceq \Lambda\alpha.e : \forall\alpha.\tau$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\cdot]$, and pushing substitutions in the goal (noting via Lemma C.0.10 that $H_1 = H_2 = \emptyset$), we are to show that

$$\begin{aligned} \exists W', H'_{1g}, H'_{2g}, \forall H_{2+}. \exists v_2. \\ H_{1*} = H'_{1g} \uplus H_{1+} \wedge H'_{1g}, H'_{2g} : W' \wedge \\ W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(H_{1+})), L_2 \cup \text{FL}(\text{cod}(H_{2+})))} W' \wedge \\ (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\forall\alpha. \tau]_\rho \wedge \\ (H_{2g+} \uplus H_{2+}, \lambda _. \ \gamma_L^2(\gamma_\Gamma^2(e^+))) \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, v_2) \not\rightarrow \end{aligned}$$

given arbitrary $\rho, \gamma_L, \gamma_\Gamma, W, L_1, L_2, H_{1g+}, H_{2g+} : W, v_1, H_{1+}, H_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\Delta], \rho.\mathbf{F} \in \mathcal{D}[\Delta], (W, \emptyset, \emptyset, \gamma_L) \in \mathcal{G}[!\Gamma]_\rho, (W, \gamma_\Gamma) \in \mathcal{G}[\Gamma]_\rho$$

and

$$(H_{1g+} \uplus H_{1+}, \lambda _. \ \gamma_L^1(\gamma_\Gamma^1(e^+))) \xrightarrow{*_{L_1}} (H_{1*}, v_1) \not\rightarrow$$

We show the goal by taking $W' = W$, $\mathsf{H}'_{1g} = \mathsf{H}_{1g+}$, and $\mathsf{H}'_{2g} = \mathsf{H}_{2g+}$, noting that configurations with values as programs do not step. Thus, it suffices to show that

$$(W, (\emptyset, \lambda_{-}.\gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^{+}))), (\emptyset, \lambda_{-}.\gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^{+})))) \in \mathcal{V}[\![\forall \alpha. \tau]\!]_{\rho}$$

Expanding the definition of $\mathcal{V}[\![\forall \alpha. \tau]\!]_{\rho}$, we are to show that

$$(W', (\emptyset, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^{+}))), (\emptyset, \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^{+})))) \in \mathcal{E}[\![\tau_2]\!]_{\rho[\mathbf{F}(\alpha) \mapsto R]}$$

given arbitrary $R \in \text{RelT}$ and W' such that $W \sqsubseteq_{\emptyset, \emptyset, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^{+})), \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^{+}))} W'$.

We have this by expanding the definition of \preceq and then $\mathcal{D}[\cdot]$ in the premise and specializing where appropriate. \square

Lemma C.0.21 (Compat $\mathbf{e}[\tau]$). *If $\Delta; !\Gamma; \Delta; \Gamma \vdash \mathbf{e} \preceq \mathbf{e} : \forall \alpha. \tau_2$, then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash \mathbf{e}[\tau_1] \preceq \mathbf{e}[\tau_1] : [\alpha \mapsto \tau_1]\tau_2$$

Proof. Expanding the definition of \preceq , \cdot^{+} , $\mathcal{E}[\cdot]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists W', \mathsf{H}'_{1g}, \mathsf{H}'_{2g}. \forall \mathsf{H}_{2+}. \exists v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\alpha \mapsto \tau_1]\!]\tau_2]_{\rho} \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^{+}))) \xrightarrow{*} (W', (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, v_2)) \not\rightarrow \end{aligned}$$

given arbitrary $\rho, \gamma_{\mathbf{L}}, \gamma_{\Gamma}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_{1+}, \mathsf{H}_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!], \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!], (W, \emptyset, \emptyset, \gamma_{\mathbf{L}}) \in \mathcal{G}[\![\Gamma_1 \uplus \Gamma_2]\!]_{\rho}, (W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]_{\rho}$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^{+}))) \xrightarrow{*} (\mathsf{H}_{1*}, v_1) \not\rightarrow$$

By Lemma C.0.14, we have that $(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^{+}))) \xrightarrow{*} (\mathsf{H}_{1*}^1, v_1^1) \not\rightarrow$ for some H_{1*}^1, v_1^1 . Then expanding the definition of \preceq and $\mathcal{E}[\cdot]$. in the premise and specializing where appropriate, we have that

$$\begin{aligned} & \exists W', \mathsf{H}'_{1g}, \mathsf{H}'_{2g}. \forall \mathsf{H}_{2+}. \exists v_2. \\ & \mathsf{H}_{1*}^1 = \mathsf{H}'_{1g} \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\emptyset, v_1^1), (\emptyset, v_2^1)) \in \mathcal{V}[\![\forall \alpha. \tau_2]\!]_{\rho} \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^{+}))) \xrightarrow{*} (W', (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, v_2^1)) \not\rightarrow \end{aligned}$$

Expanding the definition of $\mathcal{V}[\![\forall \alpha. \tau_2]\!]_\rho$, we have that

$$\begin{aligned} v_1^1 &= \lambda _. e_{1b} \wedge v_2^1 = \lambda _. e_{2b} \wedge \\ \forall R \in RelT. (W', (\emptyset, e_{1b}), (\emptyset, e_{2b})) &\in \mathcal{E}[\![\tau_2]\!]_{\rho[\mathbf{F}(\alpha) \mapsto R]} \end{aligned}$$

By Lemma C.0.14, we now have that

$$\begin{aligned} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, (\lambda _. e_{1b}) ()) &\xrightarrow{*} 1_{L_1} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, e_{1b}) \\ &\xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \\ &\not\rightarrow \end{aligned}$$

Recall that $(W', (\emptyset, e_{1b}), (\emptyset, e_{2b})) \in \mathcal{E}[\![\tau_2]\!]_{\rho[\mathbf{F}(\alpha) \mapsto R]}$ given arbitrary $R \in RelT$. Then take $R = \mathcal{V}[\![\tau_1]\!]_\rho$. Expanding the definition of $\mathcal{E}[\!.\!]$, specializing where appropriate, and applying Lemma C.0.5, we have that

$$\begin{aligned} \exists W'', \mathsf{H}''_{1g}, \mathsf{H}''_{2g}. \forall \mathsf{H}_{2+}. \exists v_2. \\ \mathsf{H}_{1*}^1 = \mathsf{H}''_{1g} \uplus \mathsf{H}_{1+} \wedge \mathsf{H}''_{1g}, \mathsf{H}''_{2g} : W'' \wedge \\ W' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(\mathsf{H}_{1+})), L_2 \cup FL(\text{cod}(\mathsf{H}_{2+})))} W'' \wedge \\ (W'', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\alpha \mapsto \tau]\!]_{\tau_2} \wedge \\ (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, e_{2b}) \xrightarrow{*_{L_2}} (\mathsf{H}''_{2g} \uplus \mathsf{H}_{2+}, v_2) \not\rightarrow \end{aligned}$$

Then all that remains is to show that

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, (\gamma_{\mathbf{L}}^2 (\gamma_{\Gamma}^2 (e^+)) ()) \xrightarrow{*_{L_2}} (\mathsf{H}_{2g''} \uplus \mathsf{H}_{2+}, v_2) \not\rightarrow$$

Specializing where appropriate, the above gives us that

$$\begin{aligned} (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, (\gamma_{\mathbf{L}}^2 (\gamma_{\Gamma}^2 (e^+)) ()) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, (\lambda _. e_{2b}) ()) \\ \xrightarrow{*_{L_2}} 1_{L_2} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, e_{2b}) \quad (\text{by MiniML}) \\ \xrightarrow{*_{L_2}} (\mathsf{H}''_{2g} \uplus \mathsf{H}_{2+}, v_2) \\ \not\rightarrow \quad (\text{values don't step}) \end{aligned}$$

□

Lemma C.0.22 (Compat ref e). *If $\Delta; !\Gamma; \Delta; \Gamma \vdash e \leq e : \tau$, then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash \text{ref } e \leq \text{ref } e : \text{ref } \tau$$

Proof. Expanding the definition of \preceq and \cdot^+ and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists W', H'_{1g}, H'_{2g}. \forall H_{2+}. \exists v_2. \\ & H_{1*} = H'_{1g} \uplus H_{1+} \wedge H'_{1g}, H'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(H_{1+})), L_2 \cup \text{FL}(\text{cod}(H_{2+})))} W' \wedge \\ & (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\text{ref } \tau]\!]_\rho \wedge \\ & (H_{2g+} \uplus H_{2+}, \text{let } _- = \text{callgc in ref } \gamma_L^2(\gamma_\Gamma^2(e^+))) \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, v_2) \rightsquigarrow \end{aligned}$$

given arbitrary $\rho, \gamma_L, \gamma_\Gamma, W, L_1, L_2, H_{1g+}, H_{2g+} : W, v_1, H_{1+}, H_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!], \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!], (W, \emptyset, \emptyset, \gamma_L) \in \mathcal{G}[\![!\Gamma]\!]_\rho, (W, \gamma_\Gamma) \in \mathcal{G}[\![\Gamma]\!]_\rho$$

and

$$(H_{1g+} \uplus H_{1+}, \text{let } _- = \text{callgc in ref } \gamma_L^1(\gamma_\Gamma^1(e^+))) \xrightarrow{*_{L_1}} (H_{1*}, v_1) \rightsquigarrow$$

First, notice that

$$\begin{aligned} & (H_{1g+} \uplus H_{1+}, \text{let } _- = \text{callgc in ref } \gamma_L^1(\gamma_\Gamma^1(e^+))) \xrightarrow{*_{L_1}} \\ & (H_{1ga} \uplus H_{1+}, \text{let } _- = () \text{ in ref } \gamma_L^1(\gamma_\Gamma^1(e^+))) \xrightarrow{*_{L_1}} \\ & (H_{1ga} \uplus H_{1+}, \text{ref } \gamma_L^1(\gamma_\Gamma^1(e^+))) \end{aligned}$$

and

$$\begin{aligned} & (H_{2g+} \uplus H_{2+}, \text{let } _- = \text{callgc in ref } \gamma_L^2(\gamma_\Gamma^2(e^+))) \xrightarrow{*_{L_2}} \\ & (H_{2ga} \uplus H_{2+}, \text{ref } \gamma_L^2(\gamma_\Gamma^2(e^+))) \end{aligned}$$

for some heaps $H_{1ga} : GCH\!eap, H_{2ga} : GCH\!eap$. By Lemma C.0.4, there exists a world

$$\begin{aligned} W \sqsubseteq & (\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, \text{FL}(\text{cod}(H_{1+})) \cup \text{FL}(\gamma_L^1(\gamma_\Gamma^1(e^+))) \cup L_1, \\ & \text{FL}(\text{cod}(H_{2+})) \cup \text{FL}(\gamma_L^2(\gamma_\Gamma^2(e^+))) \cup L_2) \end{aligned} \quad W_a$$

such that $H_{1ga}, H_{2ga} : W_a$.

Then, since $\mathcal{G}[\![\Gamma]\!]_\rho, \mathcal{G}[\![!\Gamma]\!]_\rho$ are closed under world extension by Lemma C.0.3, we can instantiate the induction hypothesis with $\rho, \gamma_\Gamma, \gamma_L, W_a$ and then expanding the expression relation, so we find that:

$$(W_a, (\emptyset, \gamma_L^1(\gamma_\Gamma^1(e^+))), (\emptyset, \gamma_L^2(\gamma_\Gamma^2(e^+)))) \in \mathcal{E}[\![\tau]\!]_\rho$$

Then, by applying Lemma 2.1 and expanding the expression relation, we find that

$$(H_{1ga} \uplus H_{1+}, \gamma_L^1(\gamma_\Gamma^1(e^+))) \xrightarrow{*_{L_1}} (H'_{1g} \uplus H_{1+}, v_1^*) \rightsquigarrow$$

and

$$(H_{2ga} \uplus H_{2+}, \gamma_L^2(\gamma_\Gamma^2(e^+))) \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, v_2^*) \rightsquigarrow$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some

$$W_a \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1+})) \cup L_1, FL(\text{cod}(\mathsf{H}_{2+})) \cup L_2)} W'$$

where

$$(W', (\emptyset, v_1^*), (\emptyset, v_2^*)) \in \mathcal{V}[\![\tau]\!]_\rho$$

Thus, we find that

$$\begin{aligned} & (\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, \text{let } _- = \text{callgc in ref } \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+))) \xrightarrow{*} L_1 \\ & (\mathsf{H}_{1ga} \uplus \mathsf{H}_{1+}, \text{ref } \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+))) \xrightarrow{*} L_1 \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, \text{ref } v_1^*) \xrightarrow{*} L_1 \\ & (\mathsf{H}'_{1g}[\ell_1 \xrightarrow{gc} v_1^*] \uplus \mathsf{H}_{1+}, \ell_1) \end{aligned}$$

and

$$\begin{aligned} & (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, \text{let } _- = \text{callgc in ref } \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+))) \xrightarrow{*} L_2 \\ & (\mathsf{H}_{2ga} \uplus \mathsf{H}_{2+}, \text{ref } \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}^+))) \xrightarrow{*} L_2 \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, \text{ref } v_2^*) \xrightarrow{*} L_2 \\ & (\mathsf{H}'_{2g}[\ell_2 \xrightarrow{gc} v_2^*] \uplus \mathsf{H}_{2+}, \ell_2) \end{aligned}$$

for some $\ell_1 \notin \text{dom}(\mathsf{H}'_{1g+} \uplus \mathsf{H}_{1+})$ and $\ell_2 \notin \text{dom}(\mathsf{H}'_{2g+} \uplus \mathsf{H}_{2+})$.

Since $\mathsf{H}'_{1g+}, \mathsf{H}'_{2g+} : W'$, $\ell_1 \notin \text{dom}(\mathsf{H}'_{1g+})$, and $\ell_2 \notin \text{dom}(\mathsf{H}'_{2g+})$, it follows that $(\ell_1, \ell_2) \notin \text{dom}(W'.\Psi)$. Then, let

$$W'' = (W'.k, \lfloor W'.\Psi \rfloor_{W'.k}[(\ell_1, \ell_2) \mapsto |\mathcal{V}[\![\tau]\!]_\rho|_{W'.k}])$$

Notice that $W''.k \leq W'.k$. Moreover, since $0W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+}))} W'$, we have $\text{dom}(\mathsf{H}_{1+}) \# W'.\Psi$ and $\text{dom}(\mathsf{H}_{2+}) \# W'.\Psi$. Since $\ell_1 \notin \text{dom}(\mathsf{H}_{1+})$ and $\ell_2 \notin \text{dom}(\mathsf{H}_{2+})$, it follows that $\text{dom}(\mathsf{H}_{1+}) \# W''.\Psi$ and $\text{dom}(\mathsf{H}_{2+}) \# W''.\Psi$. Finally, for all $(\ell'_1, \ell'_2) \in \text{dom}(W'.\Psi)$, $W''.\Psi(\ell'_1, \ell'_2) = \lfloor W'.\Psi \rfloor_{W'.k}(\ell'_1, \ell'_2) = \lfloor W'.\Psi(\ell_1, \ell_2) \rfloor_{W''.k}$. This suffices to show that $W'' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+}), \text{dom}(W'.\Psi))} W''$. Then, by Lemma C.0.2, $W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1+})) \cup L_1, FL(\text{cod}(\mathsf{H}_{2+})) \cup L_2)} W''$.

Then, choose $\mathsf{H}'_{1g} = \mathsf{H}'_{1g+}[\ell_1 \xrightarrow{gc} v_1^*]$, $\mathsf{H}'_{2g} = \mathsf{H}'_{2g+}[\ell_2 \xrightarrow{gc} v_2^*]$, and $W' = W''$. One can see that

$$(W'', (\emptyset, \ell_1), (\emptyset, \ell_2)) \in \mathcal{V}[\![\text{ref } \tau]\!]_\rho$$

because by definition of W'' , $W''(\ell_1, \ell_2) = |\mathcal{V}[\![\tau]\!]_\rho|_{W''.k}$. To finish the proof, we must show

$$\mathsf{H}'_{1g+}[\ell_1 \xrightarrow{gc} v_1^*], \mathsf{H}'_{2g+}[\ell_2 \xrightarrow{gc} v_2^*] : W''$$

For any $(\ell'_1, \ell'_2) \mapsto R \in W''.\Psi$, there are two cases: (1) $(\ell_1, \ell_2) = (\ell'_1, \ell'_2)$, in which case $W''.\Psi(\ell_1, \ell_2) = |\mathcal{V}[\![\tau]\!]_\rho|_{W'.k}$. Then, since $(W', (\emptyset, v_1^*), (\emptyset, v_2^*)) \in \mathcal{V}[\![\tau]\!]_\rho$, by Lemma C.0.2, we have $(W'', (\emptyset, v_1^*), (\emptyset, v_2^*)) \in \mathcal{V}[\![\tau]\!]_\rho$ and thus $(\triangleright W'', (\emptyset, v_1^*), (\emptyset, v_2^*)) \in$

$\lfloor \mathcal{V}[\tau] \rfloor_{\rho} \rfloor_{W'.k}$ (2) $(\ell'_1, \ell'_2) \in \text{dom}(W'.\Psi)$, in which case we must show $(\triangleright W'', (\emptyset, H'_1(\ell'_1)), (\emptyset, H'_2(\ell'_2))) \in W''.\Psi(\ell'_1, \ell'_2) = \lfloor W'.\Psi(\ell'_1, \ell'_2) \rfloor_{W'.k}$. First, since $H'_1, H'_2 : W'$, we have $(\triangleright W', (\emptyset, H'_1(\ell'_1)), (\emptyset, H'_2(\ell'_2))) \in W'.\Psi(\ell'_1, \ell'_2)$. Then, since $\triangleright W'.k < W'.k$, it follows that $(\triangleright W', (\emptyset, H'_1(\ell'_1)), (\emptyset, H'_2(\ell'_2))) \in \lfloor W'.\Psi(\ell'_1, \ell'_2) \rfloor_{W'.k}$. Finally, since $W' \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+}), \text{dom}(W'.\Psi))} W''$, it follows that $\triangleright W' \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+}), \text{dom}(W'.\Psi))} \triangleright W''$, so by Lemma C.0.3, we have

$$(\triangleright W'', (\emptyset, H'_1(\ell'_1)), (\emptyset, H'_2(\ell'_2))) \in \lfloor W'.\Psi(\ell'_1, \ell'_2) \rfloor_{W'.k}$$

as was to be proven. \square

Lemma C.0.23 (Compat !e). *If $\Delta; !\Gamma; \Delta; \Gamma \vdash e \preceq e : \text{ref } \tau$ then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash !e \preceq !e : \tau$$

Proof. Expanding the definition of \preceq and \cdot^+ and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists W', H'_{1g}, H'_{2g}. \forall H_{2+}. \exists v_2. \\ & H_{1*} = H'_{1g} \uplus H_{1+} \wedge H'_{1g}, H'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+}), \text{rhcgclocs}(W, \text{FL}(\text{cod}(H_{1+})) \cup L_1, \text{FL}(\text{cod}(H_{2+})) \cup L_2))} W' \wedge \\ & (W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\tau]_{\rho} \wedge \\ & (H_{2g+} \uplus H_{2+}, !\gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(e^+))) \xrightarrow{*_{L_2}} (H'_{2g} \uplus H_{2+}, v_2) \rightsquigarrow \end{aligned}$$

given arbitrary $\rho, \gamma_{\textcolor{red}{L}}, \gamma_{\textcolor{blue}{T}}, W, L_1, L_2, H_{1g+}, H_{2g+} : W, v_1, H_{1+}, H_{1*}$, such that

$$\rho.\textcolor{red}{L}3 \in \mathcal{D}[\Delta], \rho.\textcolor{blue}{F} \in \mathcal{D}[\Delta], (W, \emptyset, \emptyset, \gamma_{\textcolor{red}{L}}) \in \mathcal{G}[!T]_{\rho}, (W, \gamma_{\textcolor{blue}{T}}) \in \mathcal{G}[T]_{\rho}$$

and

$$(H_{1g+} \uplus H_{1+}, !\gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(e^+))) \xrightarrow{*_{L_1}} (H_{1*}, v_1) \rightsquigarrow$$

By Lemma C.0.14, we have that $(H_{1g+} \uplus H_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(e^+))) \xrightarrow{*_{L_1}} (H_{1*}^1, v_1^1) \rightsquigarrow$ for some H_{1*}^1, v_1^1 . Then expanding the definition of \preceq and $\mathcal{E}[\cdot]$, in the first premise and specializing where appropriate, we have that

$$\begin{aligned} & \exists W^1 H_{1g}^1 H_{2g}^1 v_2^1. H_{1*}^1 = H_{1g}^1 \uplus H_{1+} \wedge H_{1g}^1, H_{2g}^1 : W^1 \wedge \\ & W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+}), \text{rhcgclocs}(W, \text{FL}(\text{cod}(H_{1+})) \cup L_1, \text{FL}(\text{cod}(H_{2+})) \cup L_2))} W^1 \\ & \wedge (W^1, (\emptyset, v_1^1), (\emptyset, v_2^1)) \in \mathcal{V}[\text{ref } \tau]_{\rho} \wedge \\ & \forall H_{2+}. (H_{2g+} \uplus H_{2+}, \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(e^+))) \xrightarrow{*_{L_2}} (H_{2g}^1 \uplus H_{2+}, v_2^1) \rightsquigarrow \end{aligned} \tag{34}$$

From the definition of $\mathcal{V}[\text{ref } \tau]_{\rho}$, we know that v_1^1 and v_2^1 are both locations (call them ℓ_1 and ℓ_2) and that $W^1.\Psi(\ell_1, \ell_2) = \lfloor \mathcal{V}[\tau]_{\rho} \rfloor_{W^1.k}$. Since $H_{1g}^1, H_{2g}^1 : W^1$, this means that

$$(\mathsf{H}_{1g}^1 \uplus \mathsf{H}_{1+}, !\ell_1) \xrightarrow{*_{L_1}} (\mathsf{H}_{1g}^1 \uplus \mathsf{H}_{1+}, v_1)$$

and

$$(\mathsf{H}_{2g}^1 \uplus \mathsf{H}_{2+}, !\ell_2) \xrightarrow{*_{L_2}} (\mathsf{H}_{2g}^1 \uplus \mathsf{H}_{2+}, v_2)$$

Further, we know that $(\triangleright W^1, (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\tau]\!]_\rho$.

By Lemma C.0.3, we know that

$$\begin{aligned} W &\sqsubseteq_{\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+}), \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1+})) \cup L_1, FL(\text{cod}(\mathsf{H}_{2+})) \cup L_2)} W^1 \\ &\sqsubseteq_{\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+}), \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1+})) \cup L_1, FL(\text{cod}(\mathsf{H}_{2+})) \cup L_2)} \triangleright W^1 \end{aligned}$$

which, with $\mathsf{H}_{1*} = \mathsf{H}_{1g}^1 \uplus \mathsf{H}_{1+}$ and $\mathsf{H}_{2g+} = \mathsf{H}_{2g}^1$, is enough to prove our goal.

□

Lemma C.0.24 (Compat $e := e$). *If $\Delta; !\Gamma; \Delta; \Gamma \vdash e_1 \preceq e_1 : \text{ref } \tau$ and $\Delta; !\Gamma; \Delta; \Gamma \vdash e_2 \preceq e_2 : \tau$ then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash e_1 := e_2 \preceq e_1 := e_2 : \text{unit}$$

Proof. Expanding the definition of \preceq and \cdot^+ and pushing substitutions in the goal, we are to show that

$$\begin{aligned} &\exists W', \mathsf{H}'_{1g}, \mathsf{H}'_{2g}, \forall \mathsf{H}_{2+}. \exists v_2. \\ &\mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ &W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1+})) \cup L_1, FL(\text{cod}(\mathsf{H}_{2+})) \cup L_2)} W' \wedge \\ &(W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\text{unit}]\!]_\rho \wedge \\ &(\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}_1^+)) := \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}_2^+))) \xrightarrow{*_{L_2}} (\mathsf{H}_{2g}^1 \uplus \mathsf{H}_{2+}, v_2) \rightsquigarrow \end{aligned}$$

given arbitrary $\rho, \gamma_{\textcolor{red}{L}}, \gamma_{\textcolor{blue}{T}}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_{1+}, \mathsf{H}_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!], \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!], (W, \emptyset, \emptyset, \gamma_{\textcolor{red}{L}}) \in \mathcal{G}[\![!\Gamma]\!]_\rho, (W, \gamma_{\textcolor{blue}{T}}) \in \mathcal{G}[\![\Gamma]\!]_\rho$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_1^+)) := \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \rightsquigarrow$$

By Lemma C.0.14, we have that

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_1^+))) \xrightarrow{*_{L_1 \cup FL(\gamma_{\textcolor{red}{L}}^2(\mathbf{e}_2^+))}} (\mathsf{H}_{1*}^1, v_1^1) \rightsquigarrow$$

for some H_{1*}^1, v_1^1 .

Then expanding the definition of \preceq and $\mathcal{E}[\cdot]$. in the first premise and specializing where appropriate, we have that

$$\begin{aligned} \exists W^1 \ H_{1g}^1 \ H_{2g}^1 \ v_2^1. \ H_{1*}^1 = H_{1g}^1 \uplus H_{1+} \wedge H_{1g}^1, H_{2g}^1 : W^1 \wedge \\ W \sqsubseteq (\text{dom}(H_{1+}), \text{dom}(H_{2+})), \quad W^1 \wedge \\ \text{rchgclocs}(W, FL(\text{cod}(H_{1+})) \cup FL(\gamma_L^1(\gamma_\Gamma^1(e_2^+))) \cup L_1, \\ FL(\text{cod}(H_{2+})) \cup FL(\gamma_L^2(\gamma_\Gamma^2(e_2^+))) \cup L_2) \\ (W^1, (\emptyset, v_1^1), (\emptyset, v_2^1)) \in \mathcal{V}[\text{ref } \tau]_\rho \wedge \\ \forall H_{2+}. (H_{2g+} \uplus H_{2+}, \gamma_L^2(\gamma_\Gamma^2(e_1^+))) \xrightarrow{*}_{L_2 \cup FL(\gamma_L^1(\gamma_\Gamma^1(e_2^+)))} (H_{2g}^1 \uplus H_{2+}, v_2^1) \rightsquigarrow \end{aligned} \tag{35}$$

From the definition of $\mathcal{V}[\text{ref } \tau]_\rho$, we know that v_1^1 and v_2^1 are both locations (call them ℓ_1 and ℓ_2) and that $W^1.\Psi(\ell_1, \ell_2) = [\mathcal{V}[\tau]_\rho]_{W^1.k}$.

Now, we again appeal to Lemma C.0.14, this time with the context $\ell_i := [\cdot]$. This means, in particular, that we have that:

$$(H_{1g}^1 \uplus H_{1+}, \gamma_L^1(\gamma_\Gamma^1(e_2^+))) \xrightarrow{*}_{L_1 \cup FL(\ell_1)} (H_{1*}^1, v_1^2) \rightsquigarrow \text{for some } H_{1*}^2, v_1^2.$$

Now we expand the definition of \preceq and $\mathcal{E}[\cdot]$. in the second premise and specialize where appropriate to get that

$$\begin{aligned} \exists W^2 \ H_{1g}^2 \ H_{2g}^2 \ v_2^2. \ H_{1*}^2 = H_{1g}^2 \uplus H_{1+} \wedge H_{1g}^2, H_{2g}^2 : W^2 \wedge \\ W^1 \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+}), \text{rchgclocs}(W^1, FL(\ell_1) \cup FL(\text{cod}(H_{1+})) \cup L_1, FL(\ell_2) \cup FL(\text{cod}(H_{2+})) \cup L_2))} W^2 \\ \wedge (W^2, (\emptyset, v_1^2), (\emptyset, v_2^2)) \in \mathcal{V}[\text{unit}]_\rho \wedge \\ \forall H_{2+}. (H_{2g+}^2 \uplus H_{2+}, \gamma_L^2(\gamma_\Gamma^2(e_2^+))) \xrightarrow{*}_{L_2 \cup FL(\ell_2)} (H_{2g}^2 \uplus H_{2+}, v_2^2) \rightsquigarrow \end{aligned} \tag{36}$$

Now we can assemble the pieces that we need to complete the proof. First, we stitch together our reductions (we reduced analogously on the left side):

$$\begin{aligned} & (H_{2g+} \uplus H_{2+}, \gamma_L^2(\gamma_\Gamma^2(e_1^+)) := \gamma_L^2(\gamma_\Gamma^2(e_2^+))) \\ & \xrightarrow{*}_{L_2 \cup FL(\gamma_L^1(\gamma_\Gamma^1(e_2^+)))} (H_{2g}^1 \uplus H_{2+}, \ell_2 := \gamma_L^2(\gamma_\Gamma^2(e_2^+))) \\ & \xrightarrow{*}_{L_2 \cup FL(\ell_1)} (H_{2g}^2 \uplus H_{2+}, \ell_2 := v_2^2) \\ & \xrightarrow{*}_{L_2 \cup FL(\ell_1)} (H_{2g}^2[\ell_2 := v_2^2] \uplus H_{2+}, ()) \end{aligned}$$

Next, we need to show a W' such that $W \sqsubseteq_{\text{dom}(H_{1+}), \text{dom}(H_{2+}), \text{rchgclocs}(W, FL(\text{cod}(H_{1+})) \cup L_1, FL(\text{cod}(H_{2+})) \cup L_2)} W'$ and $H_{1g}^2[\ell_1 := v_1^2], H_{2g}^2[\ell_2 := v_2^2] : W'$. We can choose W^2 , as we know that at W^1 , ℓ_1, ℓ_2 mapped to $\mathcal{V}[\tau]_\rho$, and W^2 is an extension of W^1 that protected those locations, and thus the above worlds satisfy this world. Since otherwise, membership in $\mathcal{V}[\text{unit}]_\rho$ is trivial, this suffices to finish the proof. \square

Lemma C.0.25 (Compat $\langle e \rangle_\tau$). *If $\Delta; \Gamma; \Delta; !\Gamma \vdash e \preceq e : \tau$ and $\tau \sim \tau$, then*

$$\Delta; !\Gamma; \Delta; \Gamma \vdash \langle e \rangle_\tau \preceq \langle e \rangle_\tau : \tau$$

Proof. Expanding the definition of \preceq and \cdot^+ and pushing substitutions in the goal, we are to show that

$$(W, (\mathsf{H}_1, \mathsf{C}_{\tau \mapsto \tau}(\gamma_{\Gamma}^1(\gamma_{\text{L}}^1(e^+))))), (\mathsf{H}_2, \mathsf{C}_{\tau \mapsto \tau}(\gamma_{\Gamma}^2(\gamma_{\text{L}}^2(e^+)))) \in \mathcal{E}[\tau]_\rho \quad (37)$$

given arbitrary $\rho, \gamma_{\Gamma}, \gamma_{\text{L}}$ such that $\rho.\mathbf{L3} \in \mathcal{D}[\Delta]$, $\rho.\mathbf{F} \in \mathcal{D}[\Delta]$, $(W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\text{L}}) \in \mathcal{G}[\Gamma]_\rho$, $\gamma_{\Gamma} \in \mathcal{G}[\Gamma]_\rho$. Expanding the definition of \approx in the premise, specializing where appropriate, and commuting substitutions, we have that

$$(W, (\mathsf{H}_1, \gamma_{\Gamma}^1(\gamma_{\text{L}}^1(e^+))), (\mathsf{H}_2, \gamma_{\Gamma}^2(\gamma_{\text{L}}^2(e^+)))) \in \mathcal{E}[\tau]_\rho$$

Then since $\tau \sim \tau$, we have (37) by Lemma C.0.15. \square

Lemma C.0.26 (Compat \mathbf{x}).

$$\Delta; \Gamma; \Delta; \mathbf{x} : \tau \vdash \mathbf{x} \preceq \mathbf{x} : \tau$$

Proof. Expanding the conclusion, we must show that given

$$\begin{aligned} &\forall \rho, \gamma_{\Gamma}, \gamma_{\text{L}}, W, \mathsf{H}_1, \mathsf{H}_2, \\ &\rho.\mathbf{F} \in \mathcal{D}[\Delta] \wedge \rho.\mathbf{L3} \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_\rho \wedge (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\text{L}}) \in \mathcal{G}[\mathbf{x} : \tau]_\rho \end{aligned}$$

it holds that:

$$(W, (\mathsf{H}_1, \gamma_{\text{L}}^1(\gamma_{\Gamma}^1(\mathbf{x}^+))), (\mathsf{H}_2, \gamma_{\text{L}}^2(\gamma_{\Gamma}^2(\mathbf{x}^+)))) \in \mathcal{E}[\tau]_\rho$$

By Lemma C.0.8, it suffices to show that:

$$(W, (\mathsf{H}_1, \gamma_{\text{L}}^1(\gamma_{\Gamma}^1(\mathbf{x}^+))), (\mathsf{H}_2, \gamma_{\text{L}}^2(\gamma_{\Gamma}^2(\mathbf{x}^+)))) \in \mathcal{V}[\tau]_\rho$$

Because $(W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\text{L}}) \in \mathcal{G}[\mathbf{x} : \tau]_\rho$, we must have $\gamma_{\text{L}}(\mathbf{x}) = (v_1, v_2)$ and $(W, (\mathsf{H}_1, v_1), (\mathsf{H}_2, v_2)) \in \mathcal{V}[\tau]_\rho$. Thus,

$$\gamma_{\text{L}}^1(\gamma_{\Gamma}^1(\mathbf{x}^+)) = \gamma_{\text{L}}^1(\gamma_{\Gamma}^1(\mathbf{x})) = \gamma_{\text{L}}^1(\mathbf{x}) = v_1$$

$$\gamma_{\text{L}}^2(\gamma_{\Gamma}^2(\mathbf{x}^+)) = \gamma_{\text{L}}^2(\gamma_{\Gamma}^2(\mathbf{x})) = \gamma_{\text{L}}^2(\mathbf{x}) = v_2$$

Finally, noting that $(W, (\mathsf{H}_1, v_1), (\mathsf{H}_2, v_2)) \in \mathcal{V}[\tau]_\rho$ by assumption suffices to finish the proof. \square

Lemma C.0.27 (Compat $\lambda x : \tau.e$). *If $\Delta; \Gamma; \Delta; \Gamma, x : \tau_1 \vdash e \preceq e : \tau_2$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \lambda x : \tau.e \preceq \lambda x : \tau.e : \tau_1 \multimap \tau_2$$

Proof. Expanding the conclusion, we must show that given

$$\begin{aligned} & \forall \rho, \gamma_{\Gamma}, \gamma_{\mathbf{L}}, W, \mathsf{H}_1, \mathsf{H}_2. \\ & \rho.\mathbf{F} \in \mathcal{D}[\Delta] \wedge \rho.\mathbf{L3} \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}) \in \mathcal{G}[\Gamma]_{\rho} \end{aligned}$$

it holds that:

$$(W, (\mathsf{H}_1, \lambda x. \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^+))), (\mathsf{H}_2, \lambda x. \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^+)))) \in \mathcal{E}[\tau_1 \multimap \tau_2]_{\rho}$$

By Lemma C.0.8, it suffices to show that:

$$(W, (\mathsf{H}_1, \lambda x. \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^+))), (\mathsf{H}_2, \lambda x. \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^+)))) \in \mathcal{V}[\tau_1 \multimap \tau_2]_{\rho}$$

Thus, consider some arbitrary $W', \mathsf{H}_{1v}, v_1, \mathsf{H}_{2v}, v_2$ such that $W \sqsubseteq_{\mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^+)), \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^+))} W'$ and $(W', (\mathsf{H}_{1v}, v_1), (\mathsf{H}_{2v}, v_2)) \in \mathcal{V}[\tau_1]_{\rho}$. We must show

$$(W', (\mathsf{H}_1 \uplus \mathsf{H}_{1v}, [x \mapsto v_1] \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^+))), (\mathsf{H}_2 \uplus \mathsf{H}_{2v}, [x \mapsto v_2] \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^+)))) \in \mathcal{E}[\tau_2]_{\rho}$$

Let $\gamma'_{\mathbf{L}} = \gamma_{\mathbf{L}}[x \mapsto (v_1, v_2)]$. Next, notice that $(W', \mathsf{H}_1 \uplus \mathsf{H}_{1v}, \mathsf{H}_2 \uplus \mathsf{H}_{2v}, \gamma'_{\mathbf{L}}) \in \mathcal{G}[\Gamma, \mathbf{x} : \tau_1]_{\rho}$ because $(W', \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}) \in \mathcal{G}[\Gamma]_{\rho}$ (by Lemma C.0.3) and $(W', (\mathsf{H}_{1v}, v_1), (\mathsf{H}_{2v}, v_2)) \in \mathcal{V}[\tau_1]_{\rho}$. Thus, we can instantiate the first induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma'_{\mathbf{L}}, W', \mathsf{H}_1 \uplus \mathsf{H}_{1v}, \mathsf{H}_2 \uplus \mathsf{H}_{2v}$, which suffices to prove the above statement. \square

Lemma C.0.28 (Compat $\mathbf{e}_1 \mathbf{e}_2$). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash \mathbf{e}_1 \preceq \mathbf{e}_1 : \tau_1 \multimap \tau_2$ and $\Delta; \Gamma; \Delta; \Gamma_2 \vdash \mathbf{e}_2 \preceq \mathbf{e}_2 : \tau_1$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \mathbf{e}_1 \mathbf{e}_2 \preceq \mathbf{e}_1 \mathbf{e}_2 : \tau_2$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\cdot]$, and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : MHeap. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\tau_2]_{\rho} \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+)), \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \not\sim \end{aligned}$$

given arbitrary $\rho, \gamma_{\mathbf{L}}, \gamma_{\Gamma}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : MHeap, \mathsf{H}_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\Delta], \rho.\mathbf{F} \in \mathcal{D}[\Delta], (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho}, (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}) \in \mathcal{G}[\Gamma]_{\rho}$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_1^+)) \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow$$

Then, by Lemma C.0.9, there exist $\gamma_{\textcolor{red}{L}1}, \gamma_{\textcolor{red}{L}2}, \mathsf{H}_{1l}, \mathsf{H}_{1r}, \mathsf{H}_{2l}, \mathsf{H}_{2r}$ such that $\gamma_{\textcolor{red}{L}} = \gamma_{\textcolor{red}{L}1} \uplus \gamma_{\textcolor{red}{L}2}$, $\mathsf{H}_1 = \mathsf{H}_{1l} \uplus \mathsf{H}_{1r}$, $\mathsf{H}_2 = \mathsf{H}_{2l} \uplus \mathsf{H}_{2r}$,

$$(W, \mathsf{H}_{1l}, \mathsf{H}_{2l}, \gamma_{\textcolor{red}{L}1}) \in \mathcal{G}[\![\Gamma_1]\!]_\rho$$

$$(W, \mathsf{H}_{1r}, \mathsf{H}_{2r}, \gamma_{\textcolor{red}{L}2}) \in \mathcal{G}[\![\Gamma_2]\!]_\rho$$

and for all $j \in \{1, 2\}$,

$$\gamma_{\textcolor{red}{L}}^j(\gamma_{\textcolor{blue}{T}}^j(\mathbf{e}_1^+)) = \gamma_{\textcolor{red}{L}1}^j(\gamma_{\textcolor{blue}{T}}^j(\mathbf{e}_1^+))$$

$$\gamma_{\textcolor{red}{L}}^j(\gamma_{\textcolor{blue}{T}}^j(\mathbf{e}_2^+)) = \gamma_{\textcolor{red}{L}2}^j(\gamma_{\textcolor{blue}{T}}^j(\mathbf{e}_2^+))$$

Then, by instantiating the first induction hypothesis with $\rho, \gamma_{\textcolor{blue}{T}}, \gamma_{\textcolor{red}{L}1}, W, \mathsf{H}_{1l}, \mathsf{H}_{2l}$, we find

$$(W, (\mathsf{H}_{1l}, \gamma_{\textcolor{red}{L}1}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_1^+))), (\mathsf{H}_{2l}, \gamma_{\textcolor{red}{L}2}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}_1^+)))) \in \mathcal{E}[\![\tau_1 \multimap \tau_2]\!]_\rho$$

Thus, by Lemma C.0.14, we have

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1l} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}1}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_1^+))) \xrightarrow{*_{L_1 \cup FL(\gamma_{\textcolor{red}{L}2}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_2^+)))}} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1l}^*, v_{1l}) \not\rightarrow$$

and, for any H_{2+} ,

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_{2l} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+}, \gamma_{\textcolor{red}{L}1}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}_1^+))) \xrightarrow{*_{L_2 \cup FL(\gamma_{\textcolor{red}{L}2}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}_2^+)))}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2l}^*, v_{2l}) \not\rightarrow$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some

$$\begin{aligned} W \sqsubseteq & (\text{dom}(\mathsf{H}_{1r} \uplus \mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2r} \uplus \mathsf{H}_{2+})), \\ & \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1r})) \cup FL(\text{cod}(\mathsf{H}_{1+})) \cup FL(\gamma_{\textcolor{red}{L}2}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_2^+))) \cup L_1, \\ & FL(\text{cod}(\mathsf{H}_{2r})) \cup FL(\text{cod}(\mathsf{H}_{2+})) \cup FL(\gamma_{\textcolor{red}{L}2}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}_2^+))) \cup L_2) \end{aligned} \quad W'$$

and

$$(W', (\mathsf{H}_{1l}^*, v_{1l}), (\mathsf{H}_{2l}^*, v_{2l})) \in \mathcal{V}[\![\tau_1 \multimap \tau_2]\!]_\rho$$

By expanding the value relation, we find that there exist expressions e_{1l}, e_{2l} such that $v_{1l} = \lambda x. e_{1l}$ and $v_{2l} = \lambda x. e_{2l}$.

Then, since $\mathcal{G}[\![\Gamma]\!]_\rho, \mathcal{G}[\![\Gamma_1 \uplus \Gamma_2]\!]_\rho$ are closed under world extension by Lemma C.0.3, we can instantiate the second induction hypothesis with $\rho, \gamma_{\textcolor{blue}{T}}, \gamma_{\textcolor{red}{L}2}, W', \mathsf{H}_{1r}, \mathsf{H}_{2r}$ to find

$$(W', (\mathsf{H}_{1r}, \gamma_{\textcolor{red}{L}2}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_2^+))), (\mathsf{H}_{2r}, \gamma_{\textcolor{red}{L}2}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}_2^+)))) \in \mathcal{E}[\![\tau_1]\!]_\rho$$

Thus, by Lemma C.0.14, we have

$$(\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1l}^*, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_1^+))) \xrightarrow{*_{L_1 \cup FL(e_{1l})}} (\mathsf{H}''_{1g} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1l}^* \uplus \mathsf{H}_{1r}^*, v_{1r})$$

and

$$(\mathsf{H}'_{2g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2l}^*, \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}_2^+))) \xrightarrow{*_{L_2 \cup FL(e_{2l})}} (\mathsf{H}''_{2g} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2l}^* \uplus \mathsf{H}_{2r}^*, v_{2r})$$

where $\mathsf{H}''_{1g}, \mathsf{H}''_{2g} : W''$ for some

$$\begin{aligned} W' \sqsubseteq & (\text{dom}(\mathsf{H}_{1l}^* \uplus \mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2l}^* \uplus \mathsf{H}_{2+})), \text{rchglocs}(W', W'') \\ & FL(\text{cod}(\mathsf{H}_{1l}^*)) \cup FL(\text{cod}(\mathsf{H}_{1+})) \cup FL(e_{1l}) \cup L_1, \\ & FL(\text{cod}(\mathsf{H}_{2l}^*)) \cup FL(\text{cod}(\mathsf{H}_{2+})) \cup FL(e_{2l}) \cup L_2 \end{aligned}$$

and

$$(W'', (\mathsf{H}_{1r}^*, v_{1r}), (\mathsf{H}_{2r}^*, v_{2r})) \in \mathcal{V}[\![\tau_2]\!]_\rho$$

Thus, the original configurations step as follows:

$$\begin{aligned} & (\mathsf{H}_{1g} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_1^+)) \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1}} \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1l}^*, \lambda x.e_{1l} \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1}} \\ & (\mathsf{H}''_{1g} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1l}^* \uplus \mathsf{H}_{1r}^*, \lambda x.e_{1l} v_{1r}) \xrightarrow{*_{L_1}} \\ & (\mathsf{H}''_{1g} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1l}^* \uplus \mathsf{H}_{1r}^*, [x \mapsto v_{1r}]e_{1l}) \end{aligned}$$

and similarly on the other side, the configuration steps to

$$(\mathsf{H}''_{2g} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2l}^* \uplus \mathsf{H}_{2r}^*, [x \mapsto v_{2r}]e_{2l})$$

Since $(W', (\mathsf{H}_{1l}^*, \lambda x.e_{1l}), (\mathsf{H}_{2l}^*, \lambda x.e_{2l})) \in \mathcal{V}[\![\tau_1 \multimap \tau_2]\!]_\rho$, $W' \sqsubseteq_{\mathsf{H}_{1l}^*, \mathsf{H}_{2l}^*, e_{1l}, e_{2l}}$ W'' (by Lemma C.0.1), and $(W'', (\mathsf{H}_{1r}^*, v_{1r}), (\mathsf{H}_{2r}^*, v_{2r})) \in \mathcal{V}[\![\tau_2]\!]_\rho$, we have

$$(W'', (\mathsf{H}_{1l}^* \uplus \mathsf{H}_{1r}^*, [x \mapsto v_{1r}]e_{1l}), (\mathsf{H}_{2l}^* \uplus \mathsf{H}_{2r}^*, [x \mapsto v_{2r}]e_{2l})) \in \mathcal{E}[\![\tau_2]\!]_\rho \quad (38)$$

Next, by the assumption that the configuration on the left-hand side terminates, we have

$$(\mathsf{H}''_{1g} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1l}^* \uplus \mathsf{H}_{1r}^*, [x \mapsto v_{1r}]e_{1l}) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

Then, by applying (38), we find

$$(\mathsf{H}_{1*}, v_1) = (\mathsf{H}'''_{1g} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1f}, v_{1f})$$

and

$$(\mathsf{H}''_{2g} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2l}^* \uplus \mathsf{H}_{2r}^*, [x \mapsto v_{2r}]e_{2l}) \xrightarrow{*_{L_2}} (\mathsf{H}'''_{2g} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2f}, v_{2f})$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'''$ for some $W'' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W'', L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'''$ and

$$(W''', (\mathsf{H}_{1f}, v_{1f}), (\mathsf{H}_{2f}, v_{2f})) \in \mathcal{V}[\![\tau_2]\!]_\rho$$

Then, choose $\mathsf{H}'_1 = \mathsf{H}_{1f}$, $\mathsf{H}'_2 = \mathsf{H}_{2f}$, $W' = W'''$, $\mathsf{H}'_{1g} = \mathsf{H}'_{1g}$, and $\mathsf{H}'_{2g} = \mathsf{H}'_{2g}$. Notice that $W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W'', L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'''$ by Lemma C.0.2. This suffices to finish the proof. \square

Lemma C.0.29 (Compat ()).

$$\Delta; \Gamma; \Delta; \emptyset \vdash () \preceq () : \mathbf{Unit}$$

Proof. Expanding the conclusion, we must show that given

$$\begin{aligned} & \forall \rho, \gamma_{\Gamma}, \gamma_{\mathbf{L}}, W, \mathsf{H}_1, \mathsf{H}_2. \\ & \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!] \wedge \rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]_\rho \wedge (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}.\mathbf{F}) \in \mathcal{G}[\!\emptyset\!]\!]_\rho \\ & \wedge \gamma_{\mathbf{L}}.\Delta = \gamma_{\text{locs}}(\rho.\mathbf{L3}) \end{aligned}$$

it holds that:

$$(W, (\mathsf{H}_1, ()), (\mathsf{H}_2, ())) \in \mathcal{E}[\![\mathbf{Unit}]\!]_\rho$$

By Lemma C.0.8, it suffices to show that:

$$(W, (\mathsf{H}_1, ()), (\mathsf{H}_2, ())) \in \mathcal{V}[\![\mathbf{Unit}]\!]_\rho$$

Notice that, since $(W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}.\mathbf{F}) \in \mathcal{G}[\!\emptyset\!]\!]_\rho$, it must be the case that $\mathsf{H}_1 = \mathsf{H}_2 = \emptyset$. Thus, one can easily see by definition that $(W, (\emptyset, ()), (\emptyset, ())) \in \mathcal{V}[\![\mathbf{Unit}]\!]_\rho$, which suffices to finish the proof. \square

Lemma C.0.30 (Compat \mathbb{B}). *If $\mathbf{b} \in \mathbb{B}$, then*

$$\Delta; \Gamma; \Delta; \emptyset \vdash \mathbf{b} \preceq \mathbf{b} : \mathbf{Bool}$$

Proof. By a simple case analysis, one can see that, for all $\mathbf{b} \in \mathbb{B}$, there exists a $b \in \{0, 1\}$ such that $\mathbf{b}^+ = b$. Expanding the conclusion, we must show that given

$$\begin{aligned} & \forall \rho, \gamma_{\Gamma}, \gamma_{\mathbf{L}}, W, \mathsf{H}_1, \mathsf{H}_2. \\ & \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!] \wedge \rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]_\rho \wedge (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}.\mathbf{F}) \in \mathcal{G}[\!\emptyset\!]\!]_\rho \\ & \wedge \gamma_{\mathbf{L}}.\Delta = \gamma_{\text{locs}}(\rho.\mathbf{L3}) \end{aligned}$$

it holds that:

$$(W, (\mathsf{H}_1, b), (\mathsf{H}_2, b)) \in \mathcal{E}[\![\mathbf{Bool}]\!]_\rho$$

By Lemma C.0.8, it suffices to show that:

$$(W, (\mathsf{H}_1, b), (\mathsf{H}_2, b)) \in \mathcal{V}[\![\mathbf{Bool}]\!]_\rho$$

Notice that, since $(W, H_1, H_2, \gamma_{\mathbf{L}}[\Gamma]) \in \mathcal{G}[\emptyset]_\rho$, it must be the case that $H_1 = H_2 = \emptyset$. Thus, since $b \in \{0, 1\}$, one can easily see by definition that $(W, (\emptyset, b), (\emptyset, b)) \in \mathcal{V}[\text{Bool}]_\rho$, which suffices to finish the proof. \square

Lemma C.0.31 (Compat let ()). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : \text{Unit}$ and $\Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 \preceq e_2 : \tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{let } () = e_1 \text{ in } e_2 \preceq \text{let } () = e_1 \text{ in } e_2 : \tau$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\cdot]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists H'_1, H'_{1g}. \forall H_{2+} : MHeap. \exists H'_2, W', H'_{2g}, v_2. \\ & H_{1*} = H'_{1g} \uplus H'_1 \uplus H_{1+} \wedge H'_{1g}, H'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W'', L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))} W' \wedge \\ & (W', (H'_1, v_1), (H'_2, v_2)) \in \mathcal{V}[\tau]_\rho \wedge \\ & (H_{2g+} \uplus H_2 \uplus H_{2+}, \text{let } _- = \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(e_1^+)) \text{ in } \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(e_2^+))) \xrightarrow{*_{L_2}} (H'_{2g} \uplus H'_2 \uplus H_{2+}, v_2) \rightsquigarrow \end{aligned}$$

given arbitrary $\rho, \gamma_{\mathbf{L}}, \gamma_{\Gamma}, W, L_1, L_2, H_{1g+}, H_{2g+} : MHeap, H_{1*}$, such that

$$\rho.\mathbf{L}3 \in \mathcal{D}[\Delta], \rho.\mathbf{F} \in \mathcal{D}[\Delta], (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_\rho, (W, H_1, H_2, \gamma_{\mathbf{L}}) \in \mathcal{G}[\Gamma_1 \uplus \Gamma_2]_\rho$$

and

$$(H_{1g+} \uplus H_1 \uplus H_{1+}, \text{let } _- = \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(e_1^+)) \text{ in } \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(e_2^+))) \xrightarrow{*_{L_1}} (H_{1*}, v_1) \rightsquigarrow$$

Then, by Lemma C.0.9, there exist $\gamma_{\mathbf{L}1}, \gamma_{\mathbf{L}2}, H_{1l}, H_{1r}, H_{2l}, H_{2r}$ such that $\gamma_{\mathbf{L}} = \gamma_{\mathbf{L}1} \uplus \gamma_{\mathbf{L}2}$, $H_1 = H_{1l} \uplus H_{1r}$, $H_2 = H_{2l} \uplus H_{2r}$,

$$(W, H_{1l}, H_{2l}, \gamma_{\mathbf{L}1}) \in \mathcal{G}[\Gamma_1]_\rho$$

$$(W, H_{1r}, H_{2r}, \gamma_{\mathbf{L}2}) \in \mathcal{G}[\Gamma_2]_\rho$$

and for all $j \in \{1, 2\}$,

$$\gamma_{\mathbf{L}}^j(\gamma_{\Gamma}^j(e_1^+)) = \gamma_{\mathbf{L}1}^j(\gamma_{\Gamma}^j(e_1^+))$$

$$\gamma_{\mathbf{L}}^j(\gamma_{\Gamma}^j(e_2^+)) = \gamma_{\mathbf{L}2}^j(\gamma_{\Gamma}^j(e_2^+))$$

Then, by instantiating the first induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma_{\mathbf{L}1}, W, H_{1l}, H_{2l}$, we find

$$(W, (H_{1l}, \gamma_{\mathbf{L}1}^1(\gamma_{\Gamma}^1(e_1^+))), (H_{2l}, \gamma_{\mathbf{L}1}^2(\gamma_{\Gamma}^2(e_1^+)))) \in \mathcal{E}[\text{Unit}]_\rho$$

Thus, by Lemma C.0.14, we have

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1l} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+))) \xrightarrow{*_{L_1 \cup FL(\gamma_{\textcolor{red}{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+)))}} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1l}^*, v_{1l}) \not\rightarrow$$

and, for any H_{2+} ,

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_{2l} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+}, \gamma_{\textcolor{red}{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+))) \xrightarrow{*_{L_2 \cup FL(\gamma_{\textcolor{red}{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)))}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2l}^*, v_{2l}) \not\rightarrow$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some

$$\begin{aligned} W \sqsubseteq & (\text{dom}(\mathsf{H}_{1r} \uplus \mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2r} \uplus \mathsf{H}_{2+})), \\ & \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1r})) \cup FL(\text{cod}(\mathsf{H}_{1+})) \cup FL(\gamma_{\textcolor{red}{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \cup L_1, \\ & FL(\text{cod}(\mathsf{H}_{2r})) \cup FL(\text{cod}(\mathsf{H}_{2+})) \cup FL(\gamma_{\textcolor{red}{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \cup L_2) \end{aligned}$$

and

$$(W', (\mathsf{H}_{1l}^*, v_{1l}), (\mathsf{H}_{2l}^*, v_{2l})) \in \mathcal{V}[\![\mathbf{Unit}]\!]_{\rho}$$

By expanding the value relation, we find $\mathsf{H}_{1l}^* = \mathsf{H}_{2l}^* = \emptyset$ and $v_1 = v_2 = ()$.

Thus, the original configuration steps as follows:

$$\begin{aligned} & (\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } _- = \gamma_{\textcolor{red}{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+)) \text{ in } \gamma_{\textcolor{red}{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{*} * \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+}, \text{let } _- = () \text{ in } \gamma_{\textcolor{red}{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \rightarrow \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \end{aligned}$$

and

$$\begin{aligned} & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } _- = \gamma_{\textcolor{red}{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+)) \text{ in } \gamma_{\textcolor{red}{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \xrightarrow{*} * \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+}, \text{let } _- = () \text{ in } \gamma_{\textcolor{red}{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \rightarrow \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+}, \gamma_{\textcolor{red}{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \end{aligned}$$

Then, since $\mathcal{G}[\![\Gamma]\!]_{\rho}, \mathcal{G}[\![\mathbf{T}_1 \uplus \mathbf{T}_2]\!]_{\rho}$ are closed under world extension by Lemma C.0.3, we can instantiate the second induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma_{\textcolor{red}{L}}, W', \mathsf{H}_{1r}, \mathsf{H}_{2r}$:

$$(W', (\mathsf{H}_{1r}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))), (\mathsf{H}_{2r}, \gamma_{\textcolor{red}{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)))) \in \mathcal{E}[\![\mathbf{T}]\!]_{\rho} \quad (39)$$

Next, by the assumption that the configuration on the left-hand side terminates, we have

$$(\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

Then, by applying (39), we find

$$(\mathsf{H}_{1*}, v_1) = (\mathsf{H}''_{1g} \uplus \mathsf{H}_{1r}^* \uplus \mathsf{H}_{1+}, v'_1)$$

and

$$(\mathsf{H}'_{2g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+}, \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \xrightarrow{*_{L_2}} (\mathsf{H}''_{2g} \uplus \mathsf{H}^*_{2r} \uplus \mathsf{H}_{2+}, v'_2)$$

where $\mathsf{H}''_{1g}, \mathsf{H}''_{2g} : W'', W' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W', L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'',$ and

$(W'', (\mathsf{H}^*_{1r}, v'_1), (\mathsf{H}^*_{2r}, v'_2)) \in \mathcal{V}[\tau]_{\rho}.$ By Lemma C.0.3, we find that

$W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W''.$ Finally, we can take $\mathsf{H}'_{1g} = \mathsf{H}''_{1g}$, $\mathsf{H}'_1 = \mathsf{H}^*_{1r}$, $\mathsf{H}'_{2g} = \mathsf{H}''_{2g}$, and $\mathsf{H}'_2 = \mathsf{H}^*_{2r}$, which suffices to finish the proof. \square

Lemma C.0.32 (Compat if). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : \mathbf{Bool}$ and $\Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 \preceq e_2 : \tau$ and $\Delta; \Gamma; \Delta; \Gamma_2 \vdash e_3 \preceq e_3 : \tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \mathbf{if} \ e_1 \ e_2 \ e_3 \preceq \mathbf{if} \ e_1 \ e_2 \ e_3 : \tau$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\cdot]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : MHeap. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\tau]_{\rho} \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \mathbf{if} \ \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+)) \ \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)) \ \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_3^+))) \\ & \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \rightsquigarrow \end{aligned}$$

given arbitrary $\rho, \gamma_{\mathbf{L}}, \gamma_{\Gamma}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : MHeap, \mathsf{H}_{1*}$, such that

$$\rho.\mathbf{L}3 \in \mathcal{D}[\Delta], \rho.\mathbf{F} \in \mathcal{D}[\Delta], (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho}, (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}) \in \mathcal{G}[\Gamma_1 \uplus \Gamma_2]_{\rho}$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \mathbf{if} \ \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+)) \ \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+)) \ \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_3^+))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \rightsquigarrow$$

Then, by Lemma C.0.9, there exist $\gamma_{\mathbf{L}1}, \gamma_{\mathbf{L}2}, \mathsf{H}_{1l}, \mathsf{H}_{1r}, \mathsf{H}_{2l}, \mathsf{H}_{2r}$ such that $\gamma_{\mathbf{L}} = \gamma_{\mathbf{L}1} \uplus \gamma_{\mathbf{L}2}$, $\mathsf{H}_1 = \mathsf{H}_{1l} \uplus \mathsf{H}_{1r}$, $\mathsf{H}_2 = \mathsf{H}_{2l} \uplus \mathsf{H}_{2r}$,

$$(W, \mathsf{H}_{1l}, \mathsf{H}_{2l}, \gamma_{\mathbf{L}1}) \in \mathcal{G}[\Gamma_1]_{\rho}$$

$$(W, \mathsf{H}_{1r}, \mathsf{H}_{2r}, \gamma_{\mathbf{L}2}) \in \mathcal{G}[\Gamma_2]_{\rho}$$

and for all $j \in \{1, 2\}$,

$$\gamma_{\mathbf{L}}^j(\gamma_{\Gamma}^j(\mathbf{e}_1^+)) = \gamma_{\mathbf{L}1}^j(\gamma_{\Gamma}^j(\mathbf{e}_1^+))$$

$$\gamma_{\mathbf{L}}^j(\gamma_{\Gamma}^j(\mathbf{e}_2^+)) = \gamma_{\mathbf{L}2}^j(\gamma_{\Gamma}^j(\mathbf{e}_2^+))$$

Then, by instantiating the first induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma_{\mathbf{L}1}, W, \mathbf{H}_{1l}, \mathbf{H}_{2l}$, we find

$$(W, (\mathbf{H}_{1l}, \gamma_{\mathbf{L}1}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+))), (\mathbf{H}_{2l}, \gamma_{\mathbf{L}1}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+)))) \in \mathcal{E}[\mathbb{Bool}]_{\rho}$$

Thus, by Lemma C.0.14, we have

$$(\mathbf{H}_{1g} \uplus \mathbf{H}_{1l} \uplus \mathbf{H}_{1r} \uplus \mathbf{H}_{1+}, \gamma_{\mathbf{L}1}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+))) \xrightarrow{*_{L_1 \cup FL(\gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+)) \cup FL(\gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_3^+))))}} (\mathbf{H}'_{1g} \uplus \mathbf{H}_{1r} \uplus \mathbf{H}_{1+} \uplus \mathbf{H}_{1l}^*, v_{1l}) \not\rightarrow$$

and, for any \mathbf{H}_{2+} ,

$$(\mathbf{H}_{2g} \uplus \mathbf{H}_{2l} \uplus \mathbf{H}_{2r} \uplus \mathbf{H}_{2+}, \gamma_{\mathbf{L}1}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+))) \xrightarrow{*_{L_2 \cup FL(\gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)) \cup FL(\gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_3^+))))}} (\mathbf{H}'_{2g} \uplus \mathbf{H}_{2r} \uplus \mathbf{H}_{2+} \uplus \mathbf{H}_{2l}^*, v_{2l}) \not\rightarrow$$

where $\mathbf{H}'_{1g}, \mathbf{H}'_{2g} : W'$ for some

$$\begin{aligned} W \sqsubseteq & (\text{dom}(\mathbf{H}_{1r} \uplus \mathbf{H}_{1+}), \text{dom}(\mathbf{H}_{2r} \uplus \mathbf{H}_{2+})), \\ & \text{rchgclocs}(W, FL(\text{cod}(\mathbf{H}_{1r})) \cup FL(\text{cod}(\mathbf{H}_{1+})) \cup FL(\gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \cup FL(\gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_3^+))) \cup L_1, \\ & FL(\text{cod}(\mathbf{H}_{2r})) \cup FL(\text{cod}(\mathbf{H}_{2+})) \cup FL(\gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \cup FL(\gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_3^+))) \cup L_2) \end{aligned}$$

and

$$(W', (\mathbf{H}_{1l}^*, v_{1l}), (\mathbf{H}_{2l}^*, v_{2l})) \in \mathcal{V}[\mathbb{Bool}]_{\rho}$$

By expanding the value relation, we find $\mathbf{H}_{1l}^* = \mathbf{H}_{2l}^* = \emptyset$ and either $v_{1l} = v_{2l} = 0$ or $v_1 = v_2 = 1$. Both cases are trivially similar to each other, so we only prove the case where $v_{1l} = v_{2l} = 0$.

Then, the original configuration steps as follows:

$$\begin{aligned} & (\mathbf{H}_{1g} \uplus \mathbf{H}_1 \uplus \mathbf{H}_{1+}, \text{if } \gamma_{\mathbf{L}1}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+)) \gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+)) \gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_3^+))) \xrightarrow{*_{L_1}} *_{L_1} \\ & (\mathbf{H}'_{1g} \uplus \mathbf{H}_{1r} \uplus \mathbf{H}_{1+}, \text{if } 0 \gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+)) \gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_3^+))) \xrightarrow{*_{L_1}} *_{L_1} \\ & (\mathbf{H}'_{1g} \uplus \mathbf{H}_{1r} \uplus \mathbf{H}_{1+}, \gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \end{aligned}$$

and

$$\begin{aligned} & (\mathbf{H}_{2g} \uplus \mathbf{H}_2 \uplus \mathbf{H}_{2+}, \text{if } \gamma_{\mathbf{L}1}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+)) \gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)) \gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_3^+))) \xrightarrow{*_{L_2}} *_{L_2} \\ & (\mathbf{H}'_{2g} \uplus \mathbf{H}_{2r} \uplus \mathbf{H}_{2+}, \text{if } 0 \gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)) \gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_3^+))) \xrightarrow{*_{L_2}} *_{L_2} \\ & (\mathbf{H}'_{2g} \uplus \mathbf{H}_{2r} \uplus \mathbf{H}_{2+}, \gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \end{aligned}$$

Then, since $\mathcal{G}[\Gamma]_{\rho}, \mathcal{G}[\Gamma_1 \uplus \Gamma_2]_{\rho}$ are closed under world extension by Lemma C.0.3, we can instantiate the second induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma_{\mathbf{L}2}, W', \mathbf{H}_{1r}, \mathbf{H}_{2r}$:

$$(W', (\mathbf{H}_{1r}, \gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))), (\mathbf{H}_{2r}, \gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)))) \in \mathcal{E}[\mathcal{T}]_{\rho} \quad (40)$$

Next, by the assumption that the configuration on the left-hand side terminates, we have

$$(\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+}, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1)$$

Then, by applying (40), we find

$$(\mathsf{H}_{1*}, v_1) = (\mathsf{H}''_{1g} \uplus \mathsf{H}_{1r}^* \uplus \mathsf{H}_{1+}, v'_1)$$

and

$$(\mathsf{H}'_{2g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+}, \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \xrightarrow{*_{L_2}} (\mathsf{H}''_{2g} \uplus \mathsf{H}_{2r}^* \uplus \mathsf{H}_{2+}, v'_2)$$

where $\mathsf{H}'_{1g}, \mathsf{H}''_{1g} : W'', W' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W''$, and

$(W'', (\mathsf{H}_{1r}^*, v'_1), (\mathsf{H}_{2r}^*, v'_2)) \in \mathcal{V}[\![\tau]\!]_{\rho}$. By Lemma C.0.3, we find that

$W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W''$. Finally, we can take $\mathsf{H}'_{1g} = \mathsf{H}''_{1g}$, $\mathsf{H}'_1 = \mathsf{H}_{1r}^*$, $\mathsf{H}'_{2g} = \mathsf{H}''_{2g}$, and $\mathsf{H}'_2 = \mathsf{H}_{2r}^*$, which suffices to finish the proof. \square

Lemma C.0.33 (Compat $(\mathbf{e}_1, \mathbf{e}_2)$). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash \mathbf{e}_1 \preceq \mathbf{e}_1 : \tau_1$ and $\Delta; \Gamma; \Delta; \Gamma_2 \vdash \mathbf{e}_2 \preceq \mathbf{e}_2 : \tau_2$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash (\mathbf{e}_1, \mathbf{e}_2) \preceq (\mathbf{e}_1, \mathbf{e}_2) : \tau_1 \otimes \tau_2$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\![\cdot]\!]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : MHeap. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\![\tau_1 \otimes \tau_2]\!]_{\rho} \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, (\gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+)), \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)))) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \not\rightarrow \end{aligned}$$

given arbitrary $\rho, \gamma_{\mathbf{L}}, \gamma_{\Gamma}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : MHeap, \mathsf{H}_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!], \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!], (W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]_{\rho}, (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}) \in \mathcal{G}[\![\Gamma_1 \uplus \Gamma_2]\!]_{\rho}$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, (\gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+)), \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+)))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow$$

Then, by Lemma C.0.9, there exist $\gamma_{\mathbf{L}1}, \gamma_{\mathbf{L}2}, \mathbf{H}_{1l}, \mathbf{H}_{1r}, \mathbf{H}_{2l}, \mathbf{H}_{2r}$ such that $\gamma_{\mathbf{L}} = \gamma_{\mathbf{L}1} \uplus \gamma_{\mathbf{L}2}$, $\mathbf{H}_1 = \mathbf{H}_{1l} \uplus \mathbf{H}_{1r}$, $\mathbf{H}_2 = \mathbf{H}_{2l} \uplus \mathbf{H}_{2r}$,

$$(W, \mathbf{H}_{1l}, \mathbf{H}_{2l}, \gamma_{\mathbf{L}1}) \in \mathcal{G}[\Gamma_1]_\rho$$

$$(W, \mathbf{H}_{1r}, \mathbf{H}_{2r}, \gamma_{\mathbf{L}2}) \in \mathcal{G}[\Gamma_2]_\rho$$

and for all $j \in \{1, 2\}$,

$$\gamma_{\mathbf{L}}^j(\gamma_{\Gamma}^j(\mathbf{e}_1^+)) = \gamma_{\mathbf{L}1}^j(\gamma_{\Gamma}^j(\mathbf{e}_1^+))$$

$$\gamma_{\mathbf{L}}^j(\gamma_{\Gamma}^j(\mathbf{e}_2^+)) = \gamma_{\mathbf{L}2}^j(\gamma_{\Gamma}^j(\mathbf{e}_2^+))$$

Then, by instantiating the first induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma_{\mathbf{L}1}, W, \mathbf{H}_{1l}, \mathbf{H}_{2l}$, we find

$$(W, (\mathbf{H}_{1l}, \gamma_{\mathbf{L}1}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+))), (\mathbf{H}_{2l}, \gamma_{\mathbf{L}1}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+)))) \in \mathcal{E}[\tau_1]_\rho$$

Thus, by Lemma C.0.14, we have

$$(\mathbf{H}_{1g} \uplus \mathbf{H}_{1l} \uplus \mathbf{H}_{1r} \uplus \mathbf{H}_{1+}, \gamma_{\mathbf{L}1}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+))) \xrightarrow{*_{L_1 \cup FL(\gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+)))}} (\mathbf{H}'_{1g} \uplus \mathbf{H}_{1r} \uplus \mathbf{H}_{1+} \uplus \mathbf{H}_{1l}^*, v_{1l}) \not\rightarrow$$

and, for any \mathbf{H}_{2+} ,

$$(\mathbf{H}_{2g} \uplus \mathbf{H}_{2l} \uplus \mathbf{H}_{2r} \uplus \mathbf{H}_{2+}, \gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \xrightarrow{*_{L_2 \cup FL(\gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)))}} (\mathbf{H}'_{2g} \uplus \mathbf{H}_{2r} \uplus \mathbf{H}_{2+} \uplus \mathbf{H}_{2l}^*, v_{2l}) \not\rightarrow$$

where $\mathbf{H}'_{1g}, \mathbf{H}'_{2g} : W'$ for some

$$\begin{aligned} W \sqsubseteq & (\text{dom}(\mathbf{H}_{1r} \uplus \mathbf{H}_{1+}), \text{dom}(\mathbf{H}_{2r} \uplus \mathbf{H}_{2+})), \\ & \text{rchgclocs}(W, FL(\text{cod}(\mathbf{H}_{1r})) \cup FL(\text{cod}(\mathbf{H}_{1+})) \cup FL(\gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \cup L_1, \\ & FL(\text{cod}(\mathbf{H}_{2r})) \cup FL(\text{cod}(\mathbf{H}_{2+})) \cup FL(\gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \cup L_2) \\ W' & \end{aligned}$$

and

$$(W', (\mathbf{H}_{1l}^*, v_{1l}), (\mathbf{H}_{2l}^*, v_{2l})) \in \mathcal{V}[\tau_1]_\rho$$

Thus, since v_{1l}, v_{2l} are values as they are in the value relation, the original configuration will continue reducing on the second component of the pair. Then, since $\mathcal{G}[\Gamma]_\rho, \mathcal{G}[\Gamma_1 \uplus \Gamma_2]_\rho$ are closed under world extension by Lemma C.0.3, we can instantiate the second induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma_{\mathbf{L}2}, W', \mathbf{H}_{1r}, \mathbf{H}_{2r}$ to find

$$(W', (\mathbf{H}_{1r}, \gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))), (\mathbf{H}_{2r}, \gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)))) \in \mathcal{E}[\tau_2]_\rho$$

Thus, by Lemma C.0.14, we have

$$(\mathbf{H}'_{1g} \uplus \mathbf{H}_{1r} \uplus \mathbf{H}_{1+} \uplus \mathbf{H}_{1l}^*, \gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1 \cup FL(v_{1l})}} (\mathbf{H}''_{1g} \uplus \mathbf{H}_{1+} \uplus \mathbf{H}_{1l}^* \uplus \mathbf{H}_{1r}^*, v_{1r}) \not\rightarrow$$

and

$$(\mathsf{H}'_{2g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2l}^*, \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}_2^+))) \xrightarrow{*_{L_2 \cup FL(v_{2l})}} (\mathsf{H}''_{2g} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2l}^* \uplus \mathsf{H}_{2r}^*, v_{2r}) \not\rightarrow$$

where H_{1g}'' , $\mathsf{H}_{2g}'' : W''$ for some

$$\begin{aligned} W' \sqsubseteq & \quad (\text{dom}(\mathsf{H}_{1l}^* \uplus \mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2l}^* \uplus \mathsf{H}_{2+})), \\ & \text{rchgclocs}(W', FL(\text{cod}(\mathsf{H}_{1l}^*)) \cup FL(\text{cod}(\mathsf{H}_{1+})) \cup FL(v_{1l}) \cup L_1, \\ & \quad FL(\text{cod}(\mathsf{H}_{2l}^*)) \cup FL(\text{cod}(\mathsf{H}_{2+})) \cup FL(v_{2l}) \cup L_2) \end{aligned} \quad W''$$

and

$$(W'', (\mathsf{H}_{1r}^*, v_{1r}), (\mathsf{H}_{2r}^*, v_{2r})) \in \mathcal{V}[\![\tau_2]\!]_\rho$$

Thus, the original configurations step as follows:

$$\begin{aligned} & (\mathsf{H}_{1g} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, (\gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_1^+)), \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_2^+)))) \xrightarrow{*_{L_1}} \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1l}^*, (v_{1l}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_2^+)))) \xrightarrow{*_{L_1}} \\ & (\mathsf{H}''_{1g} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1l}^* \uplus \mathsf{H}_{1r}^*, (v_{1l}, v_{2l})) \not\rightarrow \end{aligned}$$

and similarly on the other side, the configuration steps to

$$(\mathsf{H}''_{2g} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2l}^* \uplus \mathsf{H}_{2r}^*, (v_{1r}, v_{2r}))$$

Then, choose $\mathsf{H}'_1 = \mathsf{H}_{1l}^* \uplus \mathsf{H}_{1r}^*$, $\mathsf{H}'_2 = \mathsf{H}_{2l}^* \uplus \mathsf{H}_{2r}^*$, $W' = W''$, $\mathsf{H}'_{1g} = \mathsf{H}''_{1g}$, and $\mathsf{H}'_{2g} = \mathsf{H}''_{2g}$. First, notice that

$$W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+}), \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1+})) \cup L_1, FL(\text{cod}(\mathsf{H}_{2+})) \cup L_2))} W''$$

by Lemma C.0.2. One can see

$$(W'', (\mathsf{H}_{1l}^* \uplus \mathsf{H}_{1r}^*, (v_{1l}, v_{1r})), (\mathsf{H}_{2l}^* \uplus \mathsf{H}_{2r}^*, (v_{2l}, v_{2r}))) \in \mathcal{V}[\![\tau_1 \otimes \tau_2]\!]_\rho$$

because we have $(W'', (\mathsf{H}_{1l}^*, v_{1l}), (\mathsf{H}_{2l}^*, v_{2l})) \in \mathcal{V}[\![\tau_1]\!]_\rho$ (by Lemma C.0.3) and $(W'', (\mathsf{H}_{1r}^*, v_{1r}), (\mathsf{H}_{2r}^*, v_{2r})) \in \mathcal{V}[\![\tau_2]\!]_\rho$. This suffices to finish the proof. \square

Lemma C.0.34 (Compat let ($\mathbf{x}_1, \mathbf{x}_2$)). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash \mathbf{e}_1 \preceq \mathbf{e}_1 : \tau_1 \otimes \tau_2$ and $\Delta; \Gamma; \Delta; \Gamma_2, \mathbf{x}_1 : \tau_1, \mathbf{x}_2 : \tau_2 \vdash \mathbf{e}_2 \preceq \mathbf{e}_2 : \tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{let } (\mathbf{x}_1, \mathbf{x}_2) = \mathbf{e}_1 \text{ in } \mathbf{e}_2 \preceq \text{let } (\mathbf{x}_1, \mathbf{x}_2) = \mathbf{e}_1 \text{ in } \mathbf{e}_2 : \tau$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\![\cdot]\!]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists H'_1, H'_{1g}. \forall H_{2+} : MHeap. \exists H'_2, W', H'_{2g}, v_2. \\ & H_{1*} = H'_{1g} \uplus H'_1 \uplus H_{1+} \wedge H'_{1g}, H'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, FL(\text{cod}(H_{1+})) \cup L_1, FL(\text{cod}(H_{2+})) \cup L_2)} W' \wedge \\ & (W', (H'_1, v_1), (H'_2, v_2)) \in \mathcal{V}[\![\tau]\!]_\rho \wedge \\ & (H_{2g} \uplus H_2 \uplus H_{2+}, \text{let } p = \gamma_L^2(\gamma_T^2(e_1^+)) \text{ in let } x_1 = \text{fst } p \text{ in let } x_2 = \text{snd } p \text{ in } \gamma_L^2(\gamma_T^2(e_2^+))) \xrightarrow{*_{L_2}} \\ & (H'_{2g} \uplus H'_2 \uplus H_{2+}, v_2) \rightsquigarrow \end{aligned}$$

given arbitrary $\rho, \gamma_L, \gamma_T, W, L_1, L_2, H_{1g+}, H_{2g+} : W, v_1, H_1, H_2, H_{1+} : MHeap, H_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!], \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!], (W, \gamma_T) \in \mathcal{G}[\![\Gamma]\!]_\rho, (W, H_1, H_2, \gamma_L) \in \mathcal{G}[\![\Gamma_1 \uplus \Gamma_2]\!]_\rho$$

and

$$(H_{1g+} \uplus H_1 \uplus H_{1+}, \text{let } p = \gamma_L^1(\gamma_T^1(e_1^+)) \text{ in let } x_1 = \text{fst } p \text{ in let } x_2 = \text{snd } p \text{ in } \gamma_L^1(\gamma_T^1(e_2^+))) \xrightarrow{*_{L_1}} (H_{1*}, v_1) \rightsquigarrow$$

Then, by Lemma C.0.9, there exist $\gamma_{\mathbf{L}1}, \gamma_{\mathbf{L}2}, H_{1l}, H_{1r}, H_{2l}, H_{2r}$ such that $\gamma_{\mathbf{L}} = \gamma_{\mathbf{L}1} \uplus \gamma_{\mathbf{L}2}$, $H_1 = H_{1l} \uplus H_{1r}$, $H_2 = H_{2l} \uplus H_{2r}$,

$$(W, H_{1l}, H_{2l}, \gamma_{\mathbf{L}1}) \in \mathcal{G}[\![\Gamma_1]\!]_\rho$$

$$(W, H_{1r}, H_{2r}, \gamma_{\mathbf{L}2}) \in \mathcal{G}[\![\Gamma_2]\!]_\rho$$

and for all $j \in \{1, 2\}$,

$$\gamma_{\mathbf{L}}^j(\gamma_T^j(e_1^+)) = \gamma_{\mathbf{L}1}^j(\gamma_T^j(e_1^+))$$

$$\gamma_{\mathbf{L}}^j(\gamma_T^j(e_2^+)) = \gamma_{\mathbf{L}2}^j(\gamma_T^j(e_2^+))$$

Then, by instantiating the first induction hypothesis with $\rho, \gamma_T, \gamma_{\mathbf{L}1}, W, H_{1l}, H_{2l}$, we find

$$(W, (H_{1l}, \gamma_{\mathbf{L}1}^1(\gamma_T^1(e_1^+))), (H_{2l}, \gamma_{\mathbf{L}1}^2(\gamma_T^2(e_1^+)))) \in \mathcal{E}[\![\tau_1 \otimes \tau_2]\!]_\rho$$

Thus, by Lemma C.0.14, we have

$$(H_{1g} \uplus H_{1l} \uplus H_{1r} \uplus H_{1+}, \gamma_{\mathbf{L}1}^1(\gamma_T^1(e_1^+))) \xrightarrow{*_{L_1 \cup FL(\gamma_{\mathbf{L}2}^1(\gamma_T^1(e_2^+)))}} (H'_{1g} \uplus H_{1r} \uplus H_{1+} \uplus H_{1l}^*, v_1) \rightsquigarrow$$

and, for any H_{2+} ,

$$(H_{2g} \uplus H_{2l} \uplus H_{2r} \uplus H_{2+}, \gamma_{\mathbf{L}1}^2(\gamma_T^2(e_1^+))) \xrightarrow{*_{L_2 \cup FL(\gamma_{\mathbf{L}2}^2(\gamma_T^2(e_2^+)))}} (H'_{2g} \uplus H_{2r} \uplus H_{2+} \uplus H_{2l}^*, v_2) \rightsquigarrow$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some

$$\begin{aligned} W \sqsubseteq & (\text{dom}(\mathsf{H}_{1+} \uplus \mathsf{H}_{1r}), \text{dom}(\mathsf{H}_{2+} \uplus \mathsf{H}_{2r})), \\ & \text{rchgclocs}(W, \text{FL}(\text{cod}(\mathsf{H}_{1r})) \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})) \cup \text{FL}(\gamma_{\text{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \cup L_1, \\ & \text{FL}(\text{cod}(\mathsf{H}_{2r})) \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})) \cup \text{FL}(\gamma_{\text{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \cup L_2) \end{aligned} \quad W'$$

and

$$(W', (\mathsf{H}'_{1l}, v_1), (\mathsf{H}'_{2l}, v_2)) \in \mathcal{V}[\![\tau_1 \otimes \tau_2]\!]_{\rho}$$

By expanding the value relation, we find $\mathsf{H}'_{1l} = \mathsf{H}_{1ll} \uplus \mathsf{H}_{1lr}$, $\mathsf{H}'_{2l} = \mathsf{H}_{2ll} \uplus \mathsf{H}_{2lr}$, $v_1 = (v_{1l}, v_{1r})$, and $v_2 = (v_{2l}, v_{2r})$ where $(W', (\mathsf{H}_{1ll}, v_{1l}), (\mathsf{H}_{2ll}, v_{2l})) \in \mathcal{V}[\![\tau_1]\!]_{\rho}$ and $(W', (\mathsf{H}_{1lr}, v_{1r}), (\mathsf{H}_{2lr}, v_{2r})) \in \mathcal{V}[\![\tau_2]\!]_{\rho}$.

Thus, the original configuration steps as follows:

$$\begin{aligned} & (\mathsf{H}_{1g} \uplus \mathsf{H}_{1l} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+}, \text{let } p = \gamma_{\text{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+)) \text{ in let } x_1 = \text{fst } p \\ & \quad \text{in let } x_2 = \text{snd } p \text{ in } \gamma_{\text{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{*} *_{L_1} \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1ll} \uplus \mathsf{H}_{1lr}, \text{let } p = (v_{1l}, v_{1r}) \text{ in let } x_1 = \text{fst } p \\ & \quad \text{in let } x_2 = \text{snd } p \text{ in } \gamma_{\text{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{*} *_{L_1} \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1ll} \uplus \mathsf{H}_{1lr}, [x_1 \mapsto v_{1l}, x_2 \mapsto v_{1r}] \gamma_{\text{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \end{aligned}$$

and the original configuration on the other side steps to:

$$(\mathsf{H}'_{2g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2ll} \uplus \mathsf{H}_{2lr}, [x_1 \mapsto v_{2l}, x_2 \mapsto v_{2r}] \gamma_{\text{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)))$$

Next, notice that

$$\begin{aligned} & (W', \mathsf{H}_{1ll} \uplus \mathsf{H}_{1lr} \uplus \mathsf{H}_{1r}, \mathsf{H}_{2ll} \uplus \mathsf{H}_{2lr} \uplus \mathsf{H}_{2r}, \gamma_{\text{L}}^2[x_1 \mapsto (v_{1l}, v_{2l}), x_2 \mapsto (v_{1r}, v_{2r})]) \\ & \in \mathcal{G}[\![\Gamma_2, x_1 : \tau_1, x_2 : \tau_2]\!]_{\rho} \end{aligned}$$

because $(W', (\mathsf{H}_{1ll}, v_{1l}), (\mathsf{H}_{2ll}, v_{2l})) \in \mathcal{V}[\![\tau_1]\!]_{\rho}$, $(W', (\mathsf{H}_{1lr}, v_{1r}), (\mathsf{H}_{2lr}, v_{2r})) \in \mathcal{V}[\![\tau_2]\!]_{\rho}$, and $(W', \mathsf{H}_{1r}, \mathsf{H}_{2r}, \gamma_{\text{L}}^2) \in \mathcal{G}[\![\Gamma_2]\!]_{\rho}$ (by Lemma C.0.3).

Let $\gamma'_{\text{L}} = \gamma_{\text{L}}^2[x_1 \mapsto (v_{1l}, v_{2l}), x_2 \mapsto (v_{1r}, v_{2r})]$.

Thus, we can instantiate the second induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma'_{\text{L}}, \mathsf{H}_{1ll} \uplus \mathsf{H}_{1lr} \uplus \mathsf{H}_{1r}, \mathsf{H}_{2ll} \uplus \mathsf{H}_{2lr} \uplus \mathsf{H}_{2r}$ to find that

$$\begin{aligned} & (W', (\mathsf{H}_{1ll} \uplus \mathsf{H}_{1lr} \uplus \mathsf{H}_{1r}, [x_1 \mapsto v_{1l}, x_2 \mapsto v_{1r}] \gamma_{\text{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))), \\ & (\mathsf{H}_{2ll} \uplus \mathsf{H}_{2lr} \uplus \mathsf{H}_{2r}, [x_1 \mapsto v_{2l}, x_2 \mapsto v_{2r}] \gamma_{\text{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)))) \in \mathcal{E}[\![\tau]\!]_{\rho} \end{aligned} \quad (41)$$

Next, by the assumption that the configuration on the left-hand side terminates, we have

$$(\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1ll} \uplus \mathsf{H}_{1lr}, [x_1 \mapsto v_{1l}, x_2 \mapsto v_{1r}] \gamma_{\text{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{*} L_1 (\mathsf{H}_{1*}, v_1)$$

Then, by applying (41), we find

$$(\mathsf{H}_{1*}, v_1) = (\mathsf{H}''_{1g} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1f}^*, v_{1f})$$

and

$$(\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2ll} \uplus \mathsf{H}_{2lr}, [x_1 \mapsto v_{2l}, x_2 \mapsto v_{2r}] \gamma_{\textcolor{red}{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \xrightarrow{*_{L_2}} (\mathsf{H}''_{2g} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2f}^*, v_{2f})$$

where $\mathsf{H}''_{1g}, \mathsf{H}''_{2g} : W''$ for some

$$W' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_1, \text{FL}(\text{cod}(\mathsf{H}_{2+})), L_2)} W''$$

and

$$(W'', (\mathsf{H}_{1f}^*, v_{1f}), (\mathsf{H}_{2f}^*, v_{2f})) \in \mathcal{V}[\![\tau]\!]_\rho$$

Then, choose $\mathsf{H}'_1 = \mathsf{H}_{1f}^*$, $\mathsf{H}'_2 = \mathsf{H}_{2f}^*$, $W' = W''$, $\mathsf{H}'_{1g} = \mathsf{H}''_{1g}$, and $\mathsf{H}'_{2g} = \mathsf{H}''_{2g}$. Notice that

$$W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_1, \text{FL}(\text{cod}(\mathsf{H}_{2+})), L_2)} W''$$

by Lemma C.0.2. This suffices to finish the proof. \square

Lemma C.0.35 (Compat !v). *If $\Delta; \Gamma; \Delta; !\Gamma \vdash v \preceq v : \tau$, then*

$$\Delta; \Gamma; \Delta; !\Gamma \vdash !v \preceq !v : !\tau$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\![\cdot]\!]$ and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : MHeap. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\![\!\tau]\!]_\rho \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \gamma_{\textcolor{red}{L}}^2(\gamma_{\Gamma}^2(v^+))) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \rightsquigarrow \end{aligned}$$

given arbitrary $\rho, \gamma_{\textcolor{red}{L}}, \gamma_{\Gamma}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : MHeap, \mathsf{H}_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!], \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!], (W, \gamma_{\Gamma}) \in \mathcal{G}[\![\Gamma]\!]_\rho, (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\textcolor{red}{L}}) \in \mathcal{G}[\![\!\Gamma]\!]_\rho$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\Gamma}^1(v^+))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \rightsquigarrow$$

By Lemma C.0.10, $(W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\textcolor{red}{L}}) \in \mathcal{G}[\![\!\Gamma]\!]_\rho$ implies $\mathsf{H}_1 = \mathsf{H}_2 = \emptyset$. Then, by instantiating the first induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma_{\textcolor{red}{L}}, W, \emptyset, \emptyset$, we find

$$(W, (\emptyset, \gamma_{\textcolor{red}{L}}^1(\gamma_{\Gamma}^1(v^+))), (\emptyset, \gamma_{\textcolor{red}{L}}^2(\gamma_{\Gamma}^2(v^+)))) \in \mathcal{E}[\![\tau]\!]_\rho$$

Therefore,

$$(\mathsf{H}_{1*}, v_1) = (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1f} \uplus \mathsf{H}_{1+}, v_1)$$

and

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{v}^+))) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2f} \uplus \mathsf{H}_{2+}, v_2) \dashv_{L_2}$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some

$$W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'$$

and

$$(W', (\mathsf{H}_{1f}, v_1), (\mathsf{H}_{2f}, v_2)) \in \mathcal{V}[\![\tau]\!]_\rho$$

However, by Lemma C.0.11, $\gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{v}^+))$ and $\gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{v}^+))$ are target values, so the original configurations $(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{v}^+)))$ and $(\mathsf{H}_{2g+} \uplus \mathsf{H}_{2+}, \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{v}^+)))$ must be irreducible. Ergo, the heaps that these configurations step to must be the initial configurations, so $\mathsf{H}_{1g+} = \mathsf{H}'_{1g} \uplus \mathsf{H}_{1f}$ and $\mathsf{H}_{2g+} = \mathsf{H}'_{2g} \uplus \mathsf{H}_{2f}$.

Now, notice that, by the definition of Atom_n , $\mathsf{H}_{1f} : MHeap$ and $\mathsf{H}_{2f} : MHeap$. However, since $\mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W$, we also have $\mathsf{H}_{1g+} : GCHeap$ and $\mathsf{H}_{2g+} : GCHeap$. Thus, H_{1f} and H_{2f} has only manually mapped locations while H_{1g+} and H_{2g+} have only garbage collectable locations. However, the observation above implies $\mathsf{H}_{1f} \subseteq \mathsf{H}_{1g+}$ and $\mathsf{H}_{2f} \subseteq \mathsf{H}_{2g+}$, so this must imply $\mathsf{H}_{1f} = \mathsf{H}_{2f} = \emptyset$.

Ergo, $(W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![\tau]\!]_\rho$. From here, it follows that $(W', (\emptyset, v_1), (\emptyset, v_2)) \in \mathcal{V}[\![!\tau]\!]_\rho$, which suffices to finish the proof. \square

Lemma C.0.36 (Compat let !x). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : !\tau_1$ and $\Delta; \Gamma; \Delta; \Gamma_2, x : \tau_1 \vdash e_2 \preceq e_2 : \tau_2$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{let } !x = e_1 \text{ in } e_2 \preceq \text{let } !x = e_1 \text{ in } e_2 : \tau_2$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\![\cdot]\!]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : MHeap. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\![\tau_2]\!]_\rho \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } x = \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(e_1^+)) \text{ in } \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(e_2^+))) \xrightarrow{*_{L_2}} \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \dashv \end{aligned}$$

given arbitrary $\rho, \gamma_{\textcolor{red}{L}}, \gamma_{\textcolor{blue}{T}}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : MHeap, \mathsf{H}_{1*}$, such that

$$\begin{aligned} & \rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!], \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!], (W, \gamma_{\textcolor{blue}{T}}) \in \mathcal{G}[\![\Gamma]\!]_\rho, \\ & (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\textcolor{red}{L}}. \Gamma) \in \mathcal{G}[\![\Gamma_1 \uplus \Gamma_2]\!]_\rho, \gamma_{\textcolor{red}{L}}. \Delta = \gamma_{\text{locs}}(\rho. \mathbf{L3}) \end{aligned}$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } x = \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+)) \text{ in } \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow$$

Then, by Lemma C.0.9, there exist $\gamma_{\mathbf{L}1}, \gamma_{\mathbf{L}2}, \mathsf{H}_{1l}, \mathsf{H}_{1r}, \mathsf{H}_{2l}, \mathsf{H}_{2r}$ such that $\gamma_{\mathbf{L}} = \gamma_{\mathbf{L}1} \uplus \gamma_{\mathbf{L}2}$, $\mathsf{H}_1 = \mathsf{H}_{1l} \uplus \mathsf{H}_{1r}$, $\mathsf{H}_2 = \mathsf{H}_{2l} \uplus \mathsf{H}_{2r}$,

$$(W, \mathsf{H}_{1l}, \mathsf{H}_{2l}, \gamma_{\mathbf{L}1}) \in \mathcal{G}[\![\Gamma_1]\!]_\rho$$

$$(W, \mathsf{H}_{1r}, \mathsf{H}_{2r}, \gamma_{\mathbf{L}2}) \in \mathcal{G}[\![\Gamma_2]\!]_\rho$$

and for all $j \in \{1, 2\}$,

$$\gamma_{\mathbf{L}}^j(\gamma_{\Gamma}^j(\mathbf{e}_1^+)) = \gamma_{\mathbf{L}1}^j(\gamma_{\Gamma}^j(\mathbf{e}_1^+))$$

$$\gamma_{\mathbf{L}}^j(\gamma_{\Gamma}^j(\mathbf{e}_2^+)) = \gamma_{\mathbf{L}2}^j(\gamma_{\Gamma}^j(\mathbf{e}_2^+))$$

Then, by instantiating the first induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma_{\mathbf{L}1}, W, \mathsf{H}_{1l}, \mathsf{H}_{2l}$, we find

$$(W, (\mathsf{H}_{1l}, \gamma_{\mathbf{L}1}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+))), (\mathsf{H}_{2l}, \gamma_{\mathbf{L}1}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+)))) \in \mathcal{E}[\![\!\tau_1]\!]_\rho$$

Thus, by Lemma C.0.14, we have

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1l} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+}, \gamma_{\mathbf{L}1}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+))) \xrightarrow{*_{L_1 \cup FL(\gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+)))}} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}'_{1l}, v_{1l}) \not\rightarrow$$

and, for any H_{2+} ,

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_{2l} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+}, \gamma_{\mathbf{L}1}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+))) \xrightarrow{*_{L_2 \cup FL(\gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)))}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}'_{2l}, v_{2l}) \not\rightarrow$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some

$$\begin{aligned} W \sqsubseteq & (\text{dom}(\mathsf{H}_{1r} \uplus \mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2r} \uplus \mathsf{H}_{2+})), \\ & \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1r})) \cup FL(\text{cod}(\mathsf{H}_{1+})) \cup FL(\gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \cup L_1, \\ & FL(\text{cod}(\mathsf{H}_{2r})) \cup FL(\text{cod}(\mathsf{H}_{2+})) \cup FL(\gamma_{\mathbf{L}2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \cup L_2) \\ W' & \end{aligned}$$

and

$$(W', (\mathsf{H}'_{1l}, v_{1l}), (\mathsf{H}'_{2l}, v_{2l})) \in \mathcal{V}[\![\!\tau_1]\!]_\rho$$

By expanding the value relation, we find $\mathsf{H}'_{1l} = \mathsf{H}'_{2l} = \emptyset$ and $(W', (\emptyset, v_1^*), (\emptyset, v_2^*)) \in \mathcal{V}[\![\!\tau_1]\!]_\rho$.

Thus, the original configuration steps as follows:

$$\begin{aligned} & (\mathsf{H}_{1g+} \uplus \mathsf{H}_{1l} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+}, \text{let } x = \gamma_{\mathbf{L}1}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+)) \text{ in } \gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1}} *_{L_1} \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+}, \text{let } x = v_1^* \text{ in } \gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{L_1} \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+}, [x \mapsto v_1^*] \gamma_{\mathbf{L}2}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \end{aligned}$$

and

$$\begin{aligned} & (\mathsf{H}_{2g+} \uplus \mathsf{H}_{2l} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+}, \text{let } x = \gamma_{\mathbf{L}_1^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+))} \text{ in } \gamma_{\mathbf{L}_2^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))}) \xrightarrow{*} *_L \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+}, \text{let } x = v_2^* \text{ in } \gamma_{\mathbf{L}_2^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))}) \xrightarrow{L_2} \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+}, [x \mapsto v_2^*] \gamma_{\mathbf{L}_2^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))}) \end{aligned}$$

Then, notice that

$$(W', \mathsf{H}_{1r}, \mathsf{H}_{2r}, \gamma_{\mathbf{L}_2}[x \mapsto (v_1^*, v_2^*)]) \in \mathcal{G}[\![\Gamma, \mathbf{x} : \tau_1]\!]_\rho$$

because, by Lemma C.0.3, $(W', \mathsf{H}_{1r}, \mathsf{H}_{2r}, \gamma_{\mathbf{L}_2}) \in \mathcal{G}[\![\Gamma, \mathbf{x} : \tau_1]\!]_\rho$ and $(W', (\emptyset, v_1^*), (\emptyset, v_2^*)) \in \mathcal{V}[\![\tau_1]\!]_\rho$.

$$\text{Let } \gamma'_{\mathbf{L}_2} = \gamma_{\mathbf{L}_2}[x \mapsto (v_1^*, v_2^*)].$$

Ergo, we instantiate the second induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma'_{\mathbf{L}_2}, \mathsf{H}_{1r}, \mathsf{H}_{2r}$ to find that:

$$(W', (\mathsf{H}_{1r}, [x \mapsto v_1^*] \gamma_{\mathbf{L}_2^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))}), (\mathsf{H}_{2r}, [x \mapsto v_2^*] \gamma_{\mathbf{L}_2^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))})) \in \mathcal{E}[\![\tau_2]\!]_\rho \quad (42)$$

Next, by the assumption that the configuration on the left-hand side terminates, we have

$$(\mathsf{H}'_{1g} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1+}, [x \mapsto v_1^*] \gamma_{\mathbf{L}_2^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))}) \xrightarrow{*}_{L_1} (\mathsf{H}_{1*}, v_1) \rightsquigarrow_{L_1}$$

Then, by applying (42), we find

$$(\mathsf{H}_{1*}, v_1) = (\mathsf{H}''_{1g} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1f}^*, v_{1f})$$

and

$$(\mathsf{H}'_{2g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+}, [x \mapsto v_2^*] \gamma_{\mathbf{L}_2^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))}) \xrightarrow{*}_{L_2} (\mathsf{H}''_{2g} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2f}^*, v_{2f})$$

where $\mathsf{H}''_{1g}, \mathsf{H}''_{2g} : W''$ for some

$$W' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W''$$

and

$$(W'', (\mathsf{H}_{1f}^*, v_{1f}), (\mathsf{H}_{2f}^*, v_{2f})) \in \mathcal{V}[\![\tau_2]\!]_\rho$$

Then, choose $\mathsf{H}'_1 = \mathsf{H}_{1f}^*$, $\mathsf{H}'_2 = \mathsf{H}_{2f}^*$, $W' = W''$, $\mathsf{H}'_{1g} = \mathsf{H}''_{1g}$, and $\mathsf{H}'_{2g} = \mathsf{H}''_{2g}$. Notice that

$W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W''$ by Lemma C.0.2. This suffices to finish the proof. \square

Lemma C.0.37 (Compat **dupl e**). *If $\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{e} \preceq \mathbf{e} : !\tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{dupl e} \preceq \mathbf{dupl e} : !\tau \otimes !\tau$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\![\cdot]\!]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : MHeap. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\![\mathsf{!}\tau \otimes \mathsf{!}\tau]\!]_\rho \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } x = \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}^+)) \text{ in } (x, x)) \xrightarrow{*_{L_2}} \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \rightsquigarrow \end{aligned}$$

given arbitrary $\rho, \gamma_{\textcolor{red}{L}}, \gamma_{\textcolor{blue}{T}}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : MHeap, \mathsf{H}_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!], \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!], (W, \gamma_{\textcolor{blue}{T}}) \in \mathcal{G}[\![\Gamma]\!]_\rho, (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\textcolor{red}{L}}) \in \mathcal{G}[\![\Gamma]\!]_\rho$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } x = \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}_1^+)) \text{ in } (x, x)) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \rightsquigarrow$$

We can instantiate the first induction hypothesis with $\rho, \gamma_{\textcolor{blue}{T}}, \gamma_{\textcolor{red}{L}}, \mathsf{H}_1, \mathsf{H}_2$ to find

$$(W, (\mathsf{H}_1, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+))), (\mathsf{H}_2, \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}^+)))) \in \mathcal{E}[\![\mathsf{!}\tau]\!]_\rho$$

Thus, we find

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+))) \xrightarrow{*_{L_1}} (\mathsf{H}'_{1g} \uplus \mathsf{H}_1^* \uplus \mathsf{H}_{1+}, v_1^*) \rightsquigarrow_{L_1}$$

and

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}^+))) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_2^* \uplus \mathsf{H}_{2+}, v_2^*) \rightsquigarrow_{L_2}$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some

$$W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'$$

and

$$(W', (\mathsf{H}_1^*, v_1^*), (\mathsf{H}_2^*, v_2^*)) \in \mathcal{V}[\![\mathsf{!}\tau]\!]_\rho$$

By expanding the value relation, we find $\mathsf{H}_1^* = \mathsf{H}_2^* = \emptyset$.

Thus, the original configuration steps as follows:

$$\begin{aligned} & (\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } x = \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+)) \text{ in } (x, x)) \xrightarrow{*} *_{L_1} \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, \text{let } x = v_1^* \text{ in } (x, x)) \xrightarrow{*} *_{L_1} \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, (v_1^*, v_1^*)) \end{aligned}$$

and

$$\begin{aligned} & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } x = \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}^+)) \text{ in } (x, x)) \xrightarrow{*} *_{L_2} \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, \text{let } x = v_2^* \text{ in } (x, x)) \xrightarrow{*} *_{L_2} \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+}, (v_2^*, v_2^*)) \end{aligned}$$

Notice that both of these configurations are irreducible because (v_1^*, v_1^*) and (v_2^*, v_2^*) are both values. Next, choose $\mathsf{H}'_1 = \emptyset$, $\mathsf{H}'_{1g} = \mathsf{H}'_{1g}$, $\mathsf{H}'_2 = \emptyset$, and $\mathsf{H}'_{2g} = \mathsf{H}'_{2g}$. Finally, we find $(W', (\emptyset, (v_1^*, v_1^*)), (\emptyset, (v_2^*, v_2^*))) \in \mathcal{V}[\![\mathbf{!}\tau \otimes \mathbf{!}\tau]\!]_\rho$ because $(W', (\emptyset, v_1^*), (\emptyset, v_2^*)) \in \mathcal{V}[\![\mathbf{!}\tau]\!]_\rho$, which suffices to finish the proof. \square

Lemma C.0.38 (Compat **drop e**). *If $\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{e} \preceq \mathbf{e} : \mathbf{!}\tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{drop e} \preceq \mathbf{drop e} : \mathbf{Unit}$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\![\cdot]\!]$, and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : MHeap. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\![\mathbf{Unit}]\!]_\rho \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } _- = \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}^+)) \text{ in } ()) \xrightarrow{*} *_{L_2} \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \not\rightarrow \end{aligned}$$

given arbitrary $\rho, \gamma_{\textcolor{red}{L}}, \gamma_{\textcolor{blue}{T}}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : MHeap, \mathsf{H}_{1*}$, such that

$$\begin{aligned} & \rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!], \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!], (W, \gamma_{\textcolor{blue}{T}}) \in \mathcal{G}[\![\Gamma]\!]_\rho, \\ & (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\textcolor{red}{L}}.\Gamma) \in \mathcal{G}[\![\Gamma]\!]_\rho, \gamma_{\textcolor{red}{L}}.\Delta = \gamma_{\text{locs}}(\rho.\mathbf{L3}) \end{aligned}$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } _- = \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+)) \text{ in } ()) \xrightarrow{*} *_{L_1} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

We can instantiate the first induction hypothesis with $\rho, \gamma_{\textcolor{blue}{T}}, \gamma_{\textcolor{red}{L}}, \mathsf{H}_1, \mathsf{H}_2$ to find

$$(W, (\mathsf{H}_1, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+))), (\mathsf{H}_2, \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}^+)))) \in \mathcal{E}[\![\mathbf{!}\tau]\!]_\rho$$

Thus, we find

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+))) \xrightarrow{*} *_{L_1} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_1^*, v_1^*) \not\rightarrow_{L_1}$$

and

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}^+))) \xrightarrow{*} *_{L_2} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_2^*, v_2^*) \not\rightarrow_{L_2}$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some

$$W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'$$

and

$$(W', (\mathsf{H}_1^*, v_1^*), (\mathsf{H}_2^*, v_2^*)) \in \mathcal{V}[\![\mathbf{!}\tau]\!]_\rho$$

By expanding the value relation, we find $\mathsf{H}_1^* = \mathsf{H}_2^* = \emptyset$.

Thus, the original configuration steps as follows:

$$\begin{aligned} & (\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } _- = \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{\Gamma}}^1(\mathbf{e}^+)) \text{ in } ()) \xrightarrow{*} *_{L_1} \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, \text{let } _- = v_1^* \text{ in } ()) \xrightarrow{*} *_{L_1} \\ & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+}, ()) \end{aligned}$$

and

$$\begin{aligned} & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } _- = \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{\Gamma}}^2(\mathbf{e}^+)) \text{ in } ()) \xrightarrow{*} *_{L_2} \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2+}^*, \text{let } _- = v_2^* \text{ in } ()) \xrightarrow{*} *_{L_2} \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2+}^*, ()) \end{aligned}$$

Next, choose $\mathsf{H}'_1 = \emptyset$, $\mathsf{H}'_{1g} = \mathsf{H}'_{1g}$, $\mathsf{H}'_2 = \emptyset$, and $\mathsf{H}'_{2g} = \mathsf{H}'_{2g}$. Then, we find $(W'.(\emptyset,()),(\emptyset,())) \in \mathcal{V}[\![\mathbf{Unit}]\!]_\rho$ by definition, which suffices to finish the proof. \square

Lemma C.0.39 (Compat new e). *If $\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{e} \preceq \mathbf{e} : \tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{new} \mathbf{e} \preceq \mathbf{new} \mathbf{e} : \exists \zeta. \mathbf{cap} \zeta \tau \otimes \mathbf{!ptr} \zeta$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\![\cdot]\!]$ and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : MHeap. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\![\exists \zeta. \mathbf{cap} \zeta \tau \otimes \mathbf{!ptr} \zeta]\!]_\rho \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } _- = \text{callgc in let } x_\ell = \text{ref } \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{\Gamma}}^2(\mathbf{e}^+)) \text{ in } (((), x_\ell)) \xrightarrow{*} L_2 \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \rightsquigarrow \end{aligned}$$

given arbitrary $\rho, \gamma_{\textcolor{red}{L}}, \gamma_{\textcolor{blue}{\Gamma}}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : MHeap, \mathsf{H}_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!], \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!], (W, \gamma_{\textcolor{blue}{\Gamma}}) \in \mathcal{G}[\![\Gamma]\!]_\rho, (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\textcolor{red}{L}}) \in \mathcal{G}[\![\Gamma]\!]_\rho$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } _- = \text{callgc in let } x_\ell = \text{ref } \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{\Gamma}}^1(\mathbf{e}^+)) \text{ in } (((), x_\ell)) \xrightarrow{*} L_1 (\mathsf{H}_{1*}, v_1) \rightsquigarrow_{L_1}$$

First, notice that

$$\begin{aligned} & (\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } _ = \text{callgc} \text{ in let } x_\ell = \text{ref } \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+)) \text{ in } (((), x_\ell)) \rightarrow_{L_1} \\ & (\mathsf{H}_{1ga} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } _ = () \text{ in let } x_\ell = \text{ref } \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+)) \text{ in } (((), x_\ell)) \rightarrow_{L_1} \\ & (\mathsf{H}_{1ga} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } x_\ell = \text{ref } \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+)) \text{ in } (((), x_\ell)) \end{aligned}$$

and similarly,

$$\begin{aligned} & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } _ = \text{callgc} \text{ in let } x_\ell = \text{ref } \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}^+)) \text{ in } (((), x_\ell)) \xrightarrow{*} L_2 \\ & (\mathsf{H}_{2ga} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } x_\ell = \text{ref } \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}^+)) \text{ in } (((), x_\ell)) \end{aligned}$$

for some heaps $\mathsf{H}_{1ga} : GCH\!eap$, $\mathsf{H}_{2ga} : GCH\!eap$. By Lemma C.0.4, there exists a world

$$\begin{aligned} W \sqsubseteq & (\text{dom}(\mathsf{H}_1) \uplus \text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_2) \uplus \text{dom}(\mathsf{H}_{2+})), \quad W_a \\ & \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1+})) \cup FL(\gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}))) \cup L_1, \\ & \quad FL(\text{cod}(\mathsf{H}_{2+})) \cup FL(\gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}))) \cup L_2) \end{aligned}$$

such that $\mathsf{H}_{1ga}, \mathsf{H}_{2ga} : W_a$.

Then, since $\mathcal{G}[\Gamma]_\rho$, $\mathcal{G}[\Gamma]_\rho$ are closed under world extension by Lemma C.0.3, we can instantiate the first induction hypothesis with $\rho, \gamma_{\textcolor{blue}{T}}, \gamma_{\textcolor{red}{L}}, W_a, \mathsf{H}_1, \mathsf{H}_2$, so we find

$$(W_a, (\mathsf{H}_1, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}+))), (\mathsf{H}_2, \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}+)))) \in \mathcal{E}[\tau]_\rho$$

Ergo,

$$(\mathsf{H}_{1ga} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}+))) \xrightarrow{*} L_1 (\mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+}, v_1)$$

and

$$(\mathsf{H}_{2ga} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}+))) \xrightarrow{*} L_2 (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2)$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some

$$W_a \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1+})) \cup L_1, FL(\text{cod}(\mathsf{H}_{2+})) \cup L_2)} W'$$

and

$$(W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\tau]_\rho$$

Thus, the original configuration steps as follows:

$$\begin{aligned}
 & (\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } _ = \text{callgc} \text{ in let } x_\ell = \text{ref } \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+)) \text{ in } (((), x_\ell)) \xrightarrow{*_{L_1}} \\
 & (\mathsf{H}_{1ga} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } x_\ell = \text{ref } \gamma_{\textcolor{red}{L}}^1(\gamma_{\textcolor{blue}{T}}^1(\mathbf{e}^+)) \text{ in } (((), x_\ell)) \xrightarrow{*_{L_1}} \\
 & (\mathsf{H}'_{1g} \uplus \mathsf{H}_1^* \uplus \mathsf{H}_{1+}, \text{let } x_\ell = \text{ref } v_1 \text{ in } (((), x_\ell)) \xrightarrow{*_{L_1}} \\
 & (\mathsf{H}'_{1g} \uplus \mathsf{H}_1^*[\ell_1 \xrightarrow{m} v_1] \uplus \mathsf{H}_{1+}, \text{let } x_\ell = \ell_1 \text{ in } (((), x_\ell)) \xrightarrow{*_{L_1}} \\
 & (\mathsf{H}'_{1g} \uplus \mathsf{H}_1^*[\ell_1 \xrightarrow{m} v_1] \uplus \mathsf{H}_{1+}, (((), \ell_1)))
 \end{aligned}$$

and, by similar logic,

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } x_\ell = \text{ref } \gamma_{\textcolor{red}{L}}^2(\gamma_{\textcolor{blue}{T}}^2(\mathbf{e}^+)) \text{ in } (((), x_\ell)) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_2^*[\ell_2 \mapsto v_2] \uplus \mathsf{H}_{2+}, (((), \ell_2)))$$

for some locations $\ell_1 \notin \text{dom}(\mathsf{H}'_{1g} \uplus \mathsf{H}_1^* \uplus \mathsf{H}_{1+})$ and $\ell_2 \notin \text{dom}(\mathsf{H}'_{2g} \uplus \mathsf{H}_2^* \uplus \mathsf{H}_{2+})$.

Now, we can choose $\mathsf{H}'_1 = \mathsf{H}_1^*[\ell_1 \mapsto v_1]$, $\mathsf{H}'_2 = \mathsf{H}_2^*[\ell_2 \mapsto v_2]$, $W' = W'$, $\mathsf{H}'_{1g} = \mathsf{H}'_{1g}$, and $\mathsf{H}'_{2g} = \mathsf{H}'_{2g}$. Thus, it suffices to show:

$$(W', (\mathsf{H}_1^*[\ell_1 \mapsto v_1], (((), \ell_1)), (\mathsf{H}_2^*[\ell_2 \mapsto v_2], (((), \ell_2)))) \in \mathcal{V}[\exists \zeta. \text{cap } \zeta \tau \otimes !\text{ptr } \zeta]_\rho$$

By expanding the value relation, it suffices to show:

$$(W', (\mathsf{H}_1^*[\ell_1 \mapsto v_1], (((), \ell_1)), (\mathsf{H}_2^*[\ell_2 \mapsto v_2], (((), \ell_2)))) \in \mathcal{V}[\text{cap } \zeta \tau \otimes !\text{ptr } \zeta]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_1, \ell_2)]}$$

By expanding the value relation and splitting the heaps appropriately, it suffices to show

$$(W', (\mathsf{H}_1^*[\ell_1 \mapsto v_1], ()), (\mathsf{H}_2^*[\ell_2 \mapsto v_2], ())) \in \mathcal{V}[\text{cap } \zeta \tau]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_1, \ell_2)]} \quad (43)$$

and

$$(W', (\emptyset, \ell_1), (\emptyset, \ell_2)) \in \mathcal{V}[!\text{ptr } \zeta]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_1, \ell_2)]} \quad (44)$$

We first prove (44). By expanding the value relation, it suffices to show:

$$(W', (\emptyset, \ell_1), (\emptyset, \ell_2)) \in \mathcal{V}[\text{ptr } \zeta]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_1, \ell_2)]}$$

Then, since ζ clearly maps to (ℓ_1, ℓ_2) in the environment in the above value relation, we are done.

Next, we prove (43). By expanding the value relation, since ζ clearly maps to (ℓ_1, ℓ_2) in the environment in the value relation, it suffices to show

$$(W', (\mathsf{H}_1^*, v_1), (\mathsf{H}_2, v_2)) \in \mathcal{V}[\tau]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_1, \ell_2)]}$$

However, we have $(W', (\mathsf{H}_1^*, v_1), (\mathsf{H}_2^*, v_2)) \in \mathcal{V}[\tau]_\rho$, and extending ρ does not remove any atoms from the value relation, so this suffices to finish the proof. \square

Lemma C.0.40 (Compat $\mathbf{free} \mathbf{e}$). *If $\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{e} \preceq \mathbf{e} : \exists \zeta. \mathbf{cap} \zeta \tau \otimes !\mathbf{ptr} \zeta$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{free} \mathbf{e} \preceq \mathbf{free} \mathbf{e} : \exists \zeta. \tau$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\cdot]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists H'_1, H'_{1g}, \forall H_{2+} : MHeap. \exists H'_2, W', H'_{2g}, v_2. \\ & H_{1*} = H'_{1g} \uplus H'_1 \uplus H_{1+} \wedge H'_{1g}, H'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, FL(\text{cod}(H_{1+})) \cup L_1, FL(\text{cod}(H_{2+})) \cup L_2)} W' \wedge \\ & (W', (H'_1, v_1), (H'_2, v_2)) \in \mathcal{V}[\exists \zeta. \tau]_\rho \wedge \\ & (H_{2g+} \uplus H_2 \uplus H_{2+}, \\ & \quad \text{let } x = \gamma_L^2(\gamma_F^2(e^+)) \text{ in let } x_r = !(\text{snd } x) \text{ in let } _- = \text{free} (\text{snd } x) \text{ in } x_r \xrightarrow{*}_{L_2} \\ & (H'_{2g} \uplus H'_2 \uplus H_{2+}, v_2) \not\rightarrow \end{aligned}$$

given arbitrary $\rho, \gamma_L, \gamma_F, W, L_1, L_2, H_{1g+}, H_{2g+} : W, v_1, H_1, H_2, H_{1+} : MHeap, H_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\Delta], \rho.\mathbf{F} \in \mathcal{D}[\Delta], (W, \gamma_F) \in \mathcal{G}[\Gamma]_\rho, (W, H_1, H_2, \gamma_L) \in \mathcal{G}[\Gamma]_\rho$$

and

$$\begin{aligned} & (H_{1g+} \uplus H_1 \uplus H_{1+}, \\ & \quad \text{let } x = \gamma_L^1(\gamma_F^1(e^+)) \text{ in let } x_r = !(\text{snd } x) \text{ in let } _- = \text{free} (\text{snd } x) \text{ in } x_r \xrightarrow{*}_{L_1} \\ & (H_{1*}, v_1) \not\rightarrow_{L_1} \end{aligned}$$

By instantiating the first induction hypothesis with $\rho, \gamma_F, \gamma_L, H_1, H_2$, we find

$$(W, (H_1, \gamma_L^1(\gamma_F^1(e^+))), (H_2, \gamma_L^2(\gamma_F^2(e^+)))) \in \mathcal{E}[\exists \zeta. \mathbf{cap} \zeta \tau \otimes !\mathbf{ptr} \zeta]_\rho$$

Ergo, by Lemma C.0.14,

$$(H_{1g+} \uplus H_1 \uplus H_{1+}, \gamma_L^1(\gamma_F^1(e^+))) \xrightarrow{*}_{L_1} (H'_{1g} \uplus H_1^* \uplus H_{1+}, v_1) \not\rightarrow_{L_1}$$

and

$$(H_{2g+} \uplus H_2 \uplus H_{2+}, \gamma_L^2(\gamma_F^2(e^+))) \xrightarrow{*}_{L_2} (H'_{2g} \uplus H_2^* \uplus H_{2+}, v_2) \not\rightarrow_{L_2}$$

where $H'_{1g}, H'_{2g} : W'$ for some

$$W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, FL(\text{cod}(H_{1+})) \cup L_1, FL(\text{cod}(H_{2+})) \cup L_2)} W'$$

and

$$(W', (H_1^*, v_1), (H_2^*, v_2)) \in \mathcal{V}[\exists \zeta. \mathbf{cap} \zeta \tau \otimes !\mathbf{ptr} \zeta]_\rho$$

By expanding the value relation, there exist some locations ℓ_1, ℓ_2 and, for any $i \in \{1, 2\}$,

$$v_i = (((), \ell_i)$$

and

$$H_i^* = H_i^v \uplus \{\ell_i \mapsto v_{hi}\}$$

where

$$(W', (H_1^v, v_{h1}), (H_2^v, v_{h2})) \in \mathcal{V}[\tau]_{\rho[L3(\zeta) \mapsto (\ell_1, \ell_2)]}$$

Thus, the original configuration steps as follows:

$$\begin{aligned} & (H_{1g+} \uplus H_1 \uplus H_{1+}, \text{let } x = \gamma_L^1(\gamma_F^1(e^+)) \text{ in let } x_r = !(\text{snd } x) \text{ in let } _- = \text{free } (\text{snd } x) \text{ in } x_r) \xrightarrow{*} *_{L1} \\ & (H'_{1g} \uplus H_1^v \uplus \{\ell_1 \mapsto v_{h1}\} \uplus H_{1+}, \\ & \quad \text{let } x = (((), \ell_1) \text{ in let } x_r = !(\text{snd } x) \text{ in let } _- = \text{free } (\text{snd } x) \text{ in } x_r) \xrightarrow{*} *_{L1} \\ & (H'_{1g} \uplus H_1^v \uplus \{\ell_1 \mapsto v_{h1}\} \uplus H_{1+}, \text{let } x_r = !\ell_1 \text{ in let } _- = \text{free } \ell_1 \text{ in } x_r) \xrightarrow{*} *_{L1} \\ & (H'_{1g} \uplus H_1^v \uplus \{\ell_1 \mapsto v_{h1}\} \uplus H_{1+}, \text{let } x_r = v_{h1} \text{ in let } _- = \text{free } \ell_1 \text{ in } x_r) \xrightarrow{*} *_{L1} \\ & (H'_{1g} \uplus H_1^v \uplus \{\ell_1 \mapsto v_{h1}\} \uplus H_{1+}, \text{let } _- = \text{free } \ell_1 \text{ in } v_{h1}) \xrightarrow{*} *_{L1} \\ & (H'_{1g} \uplus H_1^v \uplus H_{1+}, v_{h1}) \end{aligned}$$

and by similar logic,

$$\begin{aligned} & (H_{2g+} \uplus H_2 \uplus H_{2+}, \text{let } x = \gamma_L^2(\gamma_F^2(e^+)) \text{ in let } x_r = !(\text{snd } x) \text{ in let } _- = \text{free } (\text{snd } x) \text{ in } x_r) \xrightarrow{*} *_{L2} \\ & (H'_{2g} \uplus H_2^v \uplus H_{2+}, v_{h2}) \end{aligned}$$

Then, we can take $W' = W'$, $H'_1 = H_1^v$, $H'_2 = H_2^v$, $H'_{1g} = H'_{1g}$, and $H'_{2g} = H'_{2g}$. Thus, it suffices to show

$$(W', (H_1^v, v_{h1}), (H_2^v, v_{h2})) \in \mathcal{V}[\exists \zeta. \tau]_{\rho}$$

Because we have $(W', (H_1^v, v_{h1}), (H_2^v, v_{h2})) \in \mathcal{V}[\tau]_{\rho[L3(\zeta) \mapsto (\ell_1, \ell_2)]}$, the above statement clearly follows, which suffices to finish the proof. \square

Lemma C.0.41 (Compat swap). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : \text{cap } \zeta \tau_1$, $\Delta; \Gamma; \Delta; \Gamma_2 \vdash e_2 \preceq e_2 : \text{ptr } \zeta$, and $\Delta; \Gamma; \Delta; \Gamma_3 \vdash e_3 \preceq e_3 : \tau_3$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \text{swap } e_1 \ e_2 \ e_3 \preceq \text{swap } e_1 \ e_2 \ e_3 : \text{cap } \zeta \tau_3 \otimes \tau_1$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\![\cdot]\!]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists H'_1, H'_{1g}. \forall H_{2+} : MHeap. \exists H'_2, W', H'_{2g}, v_2. \\ & H_{1*} = H'_{1g} \uplus H'_1 \uplus H_{1+} \wedge H'_{1g}, H'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(H_{1+}), \text{dom}(H_{2+})), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(H_{1+})), L_2 \cup FL(\text{cod}(H_{2+})))} W' \wedge \\ & (W', (H'_1, v_1), (H'_2, v_2)) \in \mathcal{V}[\![\text{cap } \zeta \tau_3 \otimes \tau_1]\!]_\rho \wedge \\ & (H_{2g+} \uplus H_2 \uplus H_{2+}, \\ & \text{let } x_p = \gamma_L^2(\gamma_\Gamma^2(e_2^+)) \text{ in let } _- = \gamma_L^2(\gamma_\Gamma^2(e_1)) \text{ in let } x_{v'} = !x_p \text{ in} \\ & \quad \text{let } _- = (x_p := \gamma_L^2(\gamma_\Gamma^2(e_3^+))) \text{ in } (((), x_{v'})) \xrightarrow{*}_{L_2} \\ & (H'_{2g} \uplus H'_2 \uplus H_{2+}, v_2) \not\rightarrow \end{aligned}$$

given arbitrary $\rho, \gamma_L, \gamma_\Gamma, W, L_1, L_2, H_{1g+}, H_{2g+} : W, v_1, H_1, H_2, H_{1+} : MHeap, H_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\![\Delta]\!], \rho.\mathbf{F} \in \mathcal{D}[\![\Delta]\!], (W, \gamma_\Gamma) \in \mathcal{G}[\![\Gamma]\!]_\rho, (W, H_1, H_2, \gamma_L) \in \mathcal{G}[\![\Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3]\!]_\rho$$

and

$$\begin{aligned} & (H_{1g+} \uplus H_1 \uplus H_{1+}, \\ & \text{let } x_p = \gamma_L^1(\gamma_\Gamma^1(e_2^+)) \text{ in let } _- = \gamma_L^1(\gamma_\Gamma^1(e_1)) \text{ in let } x_{v'} = !x_p \text{ in} \\ & \quad \text{let } _- = (x_p := \gamma_L^1(\gamma_\Gamma^1(e_3^+))) \text{ in } (((), x_{v'})) \xrightarrow{*}_{L_1} \\ & (H_{1*}, v_1) \not\rightarrow_{L_1} \end{aligned}$$

Then, by applying Lemma C.0.9 twice, there exist $\gamma_{\mathbf{L}1}, \gamma_{\mathbf{L}2}, \gamma_{\mathbf{L}3}, H_{1a}, H_{1b}, H_{1c}, H_{2a}, H_{2b}, H_{2c}$ such that $\gamma_{\mathbf{L}}.\mathbf{F} = \gamma_{\mathbf{L}1} \uplus \gamma_{\mathbf{L}2} \uplus \gamma_{\mathbf{L}3}$, $H_1 = H_{1a} \uplus H_{1b} \uplus H_{1c}$, $H_2 = H_{2a} \uplus H_{2b} \uplus H_{2c}$,

$$(W, H_{1a}, H_{2a}, \gamma_{\mathbf{L}1}) \in \mathcal{G}[\![\Gamma_1]\!]_\rho$$

$$(W, H_{1b}, H_{2b}, \gamma_{\mathbf{L}2}) \in \mathcal{G}[\![\Gamma_2]\!]_\rho$$

$$(W, H_{1c}, H_{2c}, \gamma_{\mathbf{L}3}) \in \mathcal{G}[\![\Gamma_3]\!]_\rho$$

and for all $j \in \{1, 2\}$,

$$\begin{aligned} \gamma_{\mathbf{L}}^j(\gamma_\Gamma^j(e_1^+)) &= \gamma_{\mathbf{L}1}^j(\gamma_\Gamma^j(e_1^+)) \\ \gamma_{\mathbf{L}}^j(\gamma_\Gamma^j(e_2^+)) &= \gamma_{\mathbf{L}2}^j(\gamma_\Gamma^j(e_2^+)) \\ \gamma_{\mathbf{L}}^j(\gamma_\Gamma^j(e_3^+)) &= \gamma_{\mathbf{L}3}^j(\gamma_\Gamma^j(e_3^+)) \end{aligned}$$

Then, by instantiating the second induction hypothesis with $\rho, \gamma_\Gamma, \gamma_{\mathbf{L}2}, W, H_{1b}, H_{2b}$, we find

$$(W, (H_{1b}, \gamma_{\mathbf{L}2}^1(\gamma_\Gamma^1(e_2^+))), (H_{2b}, \gamma_{\mathbf{L}2}^2(\gamma_\Gamma^2(e_2^+)))) \in \mathcal{E}[\![\text{ptr } \zeta]\!]_\rho$$

Thus, by Lemma C.0.14, we have

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{1a} \uplus \mathsf{H}_{1b} \uplus \mathsf{H}_{1c} \uplus \mathsf{H}_{1+}, \gamma_{\mathbf{L}_2^1}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1 \cup FL(\gamma_{\mathbf{L}_1^1}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+))) \cup FL(\gamma_{\mathbf{L}_3^1}^1(\gamma_{\Gamma}^1(\mathbf{e}_3^+)))}} (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1a} \uplus \mathsf{H}_{1c} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1b}^*, v_{1b}) \not\rightarrow$$

and, for any H_{2+} ,

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_{2a} \uplus \mathsf{H}_{2b} \uplus \mathsf{H}_{2c} \uplus \mathsf{H}_{2+}, \gamma_{\mathbf{L}_2^2}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \xrightarrow{*_{L_2 \cup FL(\gamma_{\mathbf{L}_1^2}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+))) \cup FL(\gamma_{\mathbf{L}_3^2}^2(\gamma_{\Gamma}^2(\mathbf{e}_3^+)))}} (\mathsf{H}'_{2g} \uplus \mathsf{H}_{2a} \uplus \mathsf{H}_{2c} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2b}^*, v_{2b}) \not\rightarrow$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some

$$\begin{aligned} W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1a} \uplus \mathsf{H}_{1c} \uplus \mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2a} \uplus \mathsf{H}_{2c} \uplus \mathsf{H}_{2+}))} & \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1a})) \cup FL(\text{cod}(\mathsf{H}_{1c})) \cup FL(\text{cod}(\mathsf{H}_{1+})) \cup FL(\gamma_{\mathbf{L}_1^1}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+))) \cup FL(\gamma_{\mathbf{L}_3^1}^1(\gamma_{\Gamma}^1(\mathbf{e}_3^+))) \cup L_1, \\ & FL(\text{cod}(\mathsf{H}_{2a})) \cup FL(\text{cod}(\mathsf{H}_{2c})) \cup FL(\text{cod}(\mathsf{H}_{2+})) \cup FL(\gamma_{\mathbf{L}_1^2}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+))) \cup FL(\gamma_{\mathbf{L}_3^2}^2(\gamma_{\Gamma}^2(\mathbf{e}_3^+))) \cup L_2) \quad W' \end{aligned}$$

and

$$(W', (\mathsf{H}_{1b}^*, v_{1b}), (\mathsf{H}_{2b}^*, v_{2b})) \in \mathcal{V}[\![\text{ptr } \zeta]\!]_\rho$$

Expanding the value relation, we find that $\mathsf{H}_{1b}^* = \mathsf{H}_{2b}^* = \emptyset$ and there exist locations ℓ_1, ℓ_2 such that $\rho.\mathbf{L3}(\zeta) = (\ell_1, \ell_2) = (v_{1b}, v_{2b})$.

Then, since $\mathcal{G}[\![\Gamma]\!]_\rho, \mathcal{G}[\![\Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3]\!]_\rho$ are closed under world extension by Lemma C.0.3, we can instantiate the first induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma_{\mathbf{L}_1}, W', \mathsf{H}_{1a}, \mathsf{H}_{2a}$:

$$(W', (\mathsf{H}_{1a}, \gamma_{\mathbf{L}_1^1}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+))), (\mathsf{H}_{2a}, \gamma_{\mathbf{L}_1^2}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+)))) \in \mathcal{E}[\![\text{cap } \zeta \tau_1]\!]_\rho$$

Thus, by Lemma C.0.14, we have

$$(\mathsf{H}'_{1g} \uplus \mathsf{H}_{1a} \uplus \mathsf{H}_{1c} \uplus \mathsf{H}_{1+}, \gamma_{\mathbf{L}_1^1}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+))) \xrightarrow{*_{L_1 \cup FL(\gamma_{\mathbf{L}_3^1}^1(\gamma_{\Gamma}^1(\mathbf{e}_3^+)))}} (\mathsf{H}''_{1g} \uplus \mathsf{H}_{1c} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1a}^*, v_{1a}) \not\rightarrow$$

and, for any H_{2+} ,

$$(\mathsf{H}'_{2g} \uplus \mathsf{H}_{2a} \uplus \mathsf{H}_{2b} \uplus \mathsf{H}_{2c} \uplus \mathsf{H}_{2+}, \gamma_{\mathbf{L}_1^2}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+))) \xrightarrow{*_{L_2 \cup FL(\gamma_{\mathbf{L}_3^2}^2(\gamma_{\Gamma}^2(\mathbf{e}_3^+)))}} (\mathsf{H}''_{2g} \uplus \mathsf{H}_{2c} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2a}^*, v_{2a}) \not\rightarrow$$

where $\mathsf{H}''_{1g}, \mathsf{H}''_{2g} : W''$ for some

$$\begin{aligned} W' \sqsubseteq & \quad (\text{dom}(\mathsf{H}_{1c} \uplus \mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2c} \uplus \mathsf{H}_{2+})), \quad W'' \\ & \text{rchgclocs}(W, FL(\text{cod}(\mathsf{H}_{1c})) \cup FL(\text{cod}(\mathsf{H}_{1+})) \cup FL(\gamma_{\mathbf{L}_3^1}^1(\gamma_{\Gamma}^1(\mathbf{e}_3^+))) \cup L_1, \\ & FL(\text{cod}(\mathsf{H}_{2c})) \cup FL(\text{cod}(\mathsf{H}_{2+})) \cup FL(\gamma_{\mathbf{L}_3^2}^2(\gamma_{\Gamma}^2(\mathbf{e}_3^+))) \cup L_2) \end{aligned}$$

and

$$(W'', (\mathsf{H}_{1a}^*, v_{1a}), (\mathsf{H}_{2a}^*, v_{2a})) \in \mathcal{V}[\![\text{cap } \zeta \tau_1]\!]_\rho$$

Expanding the value relation, we find that $v_{1a} = v_{2a} = ()$ and there exist values v_1, v_2 such that $H_{1a}^* = H_{1av} \uplus \{\ell_1 \xrightarrow{m} v_1\}$, $H_{2a}^* = H_{2av} \uplus \{\ell_2 \xrightarrow{m} v_2\}$, and

$$(W'', (H_{1av}, v_1), (H_{2av}, v_2)) \in \mathcal{V}[\tau_1]_\rho$$

Then, since $\mathcal{G}[\Gamma]_\rho, \mathcal{G}[\Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3]_\rho$ are closed under world extension by Lemma C.0.3, we can instantiate the third induction hypothesis with $\rho, \gamma_\Gamma, \gamma_{\text{L3}}, W'', H_{1c}, H_{2c}$:

$$(W', (H_{1c}, \gamma_{\text{L3}}^1(\gamma_\Gamma^1(e_3^+))), (H_{2c}, \gamma_{\text{L3}}^2(\gamma_\Gamma^2(e_3^+)))) \in \mathcal{E}[\tau_3]_\rho$$

Thus, by Lemma C.0.14, we have

$$\begin{aligned} & (H_{1g}'' \uplus H_{1av} \uplus \{\ell_1 \xrightarrow{m} v_1\} \uplus H_{1c} \uplus H_{1+}, \gamma_{\text{L3}}^1(\gamma_\Gamma^1(e_3^+))) \xrightarrow{*_{L_1}} \\ & (H_{1g}''' \uplus H_{1av} \uplus \{\ell_1 \xrightarrow{m} v_1\} \uplus H_{1+} \uplus H_{1c}^*, v_{1c}) \not\rightarrow \end{aligned}$$

and, for any H_{2+} ,

$$\begin{aligned} & (H_{2g}'' \uplus H_{2av} \uplus \{\ell_2 \xrightarrow{m} v_2\} \uplus H_{2c} \uplus H_{2+}, \gamma_{\text{L3}}^2(\gamma_\Gamma^2(e_3^+))) \xrightarrow{*_{L_2}} \\ & (H_{2g}''' \uplus H_{2av} \uplus \{\ell_2 \xrightarrow{m} v_2\} \uplus H_{2+} \uplus H_{2c}^*, v_{2c}) \not\rightarrow \end{aligned}$$

where $H_{1g}''', H_{2g}''' : W'''$ for some

$$\begin{aligned} W'' \sqsubseteq & (\text{dom}(H_{1av} \uplus H_{1+}), \text{dom}(H_{2av} \uplus H_{2+})), \\ & \text{rchgclocs}(W'', L_1 \cup \text{FL}(\text{cod}(H_{1+})) \cup \text{FL}(\text{cod}(H_{1av})) \cup \text{FL}(v_1), \\ & L_2 \cup \text{FL}(\text{cod}(H_{2+})) \cup \text{FL}(\text{cod}(H_{2av})) \cup \text{FL}(v_2)) \end{aligned} \quad W'''$$

and

$$(W''', (H_{1c}^*, v_{1c}), (H_{2c}^*, v_{2c})) \in \mathcal{V}[\tau_3]_\rho$$

Thus, the original configuration steps as follows:

$$\begin{aligned}
 & (\mathsf{H}_{1g+} \uplus \mathsf{H}_{1a} \uplus \mathsf{H}_{1b} \uplus \mathsf{H}_{1c} \uplus \mathsf{H}_{1+}, \\
 & \quad \text{let } x_p = \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+)) \text{ in let } _- = \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_1)) \text{ in let } x_{v'} = !x_p \text{ in} \\
 & \quad \quad \text{let } _- = (x_p := \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_3+))) \text{ in } (((), x_{v'})) \xrightarrow{*_{L_1}} \\
 & (\mathsf{H}'_{1g} \uplus \mathsf{H}_{1a} \uplus \mathsf{H}_{1c} \uplus \mathsf{H}_{1+}, \\
 & \quad \text{let } x_p = \ell_1 \text{ in let } _- = \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_1)) \text{ in let } x_{v'} = !x_p \text{ in let } _- = (x_p := \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_3+))) \text{ in } (((), x_{v'})) \xrightarrow{*_{L_1}} \\
 & (\mathsf{H}''_{1g} \uplus \mathsf{H}_{1av} \uplus \{\ell_1 \xrightarrow{m} v_1\} \uplus \mathsf{H}_{1c} \uplus \mathsf{H}_{1+}, \\
 & \quad \text{let } _- = () \text{ in let } x_{v'} = !\ell_1 \text{ in let } _- = (\ell_1 := \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_3+))) \text{ in } (((), x_{v'})) \xrightarrow{*_{L_1}} \\
 & (\mathsf{H}'''_{1g} \uplus \mathsf{H}_{1av} \uplus \{\ell_1 \xrightarrow{m} v_1\} \uplus \mathsf{H}_{1c} \uplus \mathsf{H}_{1+}, \\
 & \quad \text{let } x_{v'} = !\ell_1 \text{ in let } _- = (\ell_1 := \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_3+))) \text{ in } (((), x_{v'})) \xrightarrow{*_{L_1}} \\
 & (\mathsf{H}''''_{1g} \uplus \mathsf{H}_{1av} \uplus \{\ell_1 \xrightarrow{m} v_1\} \uplus \mathsf{H}_{1c} \uplus \mathsf{H}_{1+}, \\
 & \quad \text{let } x_{v'} = v_1 \text{ in let } _- = (\ell_1 := \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_3+))) \text{ in } (((), x_{v'})) \xrightarrow{*_{L_1}} \\
 & (\mathsf{H}''''_{1g} \uplus \mathsf{H}_{1av} \uplus \{\ell_1 \xrightarrow{m} v_1\} \uplus \mathsf{H}_{1c} \uplus \mathsf{H}_{1+}, \text{let } _- = (\ell_1 := \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_3+))) \text{ in } (((), v_1)) \xrightarrow{*_{L_1}} \\
 & (\mathsf{H}''''_{1g} \uplus \mathsf{H}_{1av} \uplus \{\ell_1 \xrightarrow{m} v_{1c}\} \uplus \mathsf{H}_{1c}^* \uplus \mathsf{H}_{1+}, \text{let } _- = (v_{1c} := \ell_1) \text{ in } (((), v_1)) \xrightarrow{*_{L_1}} \\
 & (\mathsf{H}''''_{1g} \uplus \mathsf{H}_{1av} \uplus \{\ell_1 \xrightarrow{m} v_{1c}\} \uplus \mathsf{H}_{1c}^* \uplus \mathsf{H}_{1+}, \text{let } _- = () \text{ in } (((), v_1)) \xrightarrow{*_{L_1}} \\
 & (\mathsf{H}''''_{1g} \uplus \mathsf{H}_{1av} \uplus \{\ell_1 \xrightarrow{m} v_{1c}\} \uplus \mathsf{H}_{1c}^* \uplus \mathsf{H}_{1+}, (((), v_1)) \rightsquigarrow
 \end{aligned}$$

and similarly, on the other side, the configuration steps to:

$$(\mathsf{H}''''_{2g} \uplus \mathsf{H}_{2av} \uplus \{\ell_2 \xrightarrow{m} v_{2c}\} \uplus \mathsf{H}_{2c}^* \uplus \mathsf{H}_{2+}, (((), v_2)))$$

Then, choose $\mathsf{H}_{1'} = \mathsf{H}_{1av} \uplus \{\ell_1 \xrightarrow{m} v_{1c}\} \uplus \mathsf{H}_{1c}^*$, $\mathsf{H}_{2'} = \mathsf{H}_{2av} \uplus \{\ell_2 \xrightarrow{m} v_{2c}\} \uplus \mathsf{H}_{2c}^*$, $W' = W'''$, $\mathsf{H}'_{1g} = \mathsf{H}''''_{1g}$, and $\mathsf{H}'_{2g} = \mathsf{H}''''_{2g}$. First, notice that $W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup FL(\text{cod}(\mathsf{H}_{1+})), L_2 \cup FL(\text{cod}(\mathsf{H}_{2+})))} W'''$ by Lemma C.0.2. Then, to finish the proof, we must show that

$$(W''', (\mathsf{H}_{1av} \uplus \{\ell_1 \xrightarrow{m} v_{1c}\} \uplus \mathsf{H}_{1c}^*, (((), v_1))), (\mathsf{H}_{2av} \uplus \{\ell_2 \xrightarrow{m} v_{2c}\} \uplus \mathsf{H}_{2c}^*, (((), v_2)))) \in \mathcal{V}[\![\text{cap } \zeta \tau_3 \otimes \tau_1]\!]_\rho$$

First, we have $(W''', (\mathsf{H}_{1av}, v_1), (\mathsf{H}_{2av}, v_2)) \in \mathcal{V}[\![\tau_1]\!]_\rho$ by Lemma C.0.3. Thus, it suffices to show:

$$(W''', (\{\ell_1 \xrightarrow{m} v_{1c}\} \uplus \mathsf{H}_{1c}^*, ()), (\{\ell_2 \xrightarrow{m} v_{2c}\} \uplus \mathsf{H}_{2c}^*, v_2)) \in \mathcal{V}[\![\text{cap } \zeta \tau_3]\!]_\rho$$

This follows from the fact that $\rho.\text{L3}(\zeta) = (\ell_1, \ell_2)$ and that $(W''', (\mathsf{H}_{1c}^*, v_{1c}), (\mathsf{H}_{2c}^*, v_{2c})) \in \mathcal{V}[\![\tau_3]\!]_\rho$, which suffices to finish the proof. \square

Lemma C.0.42 (Compat $\Lambda\zeta.e$). *If $\Delta; \Gamma; \Delta, \zeta; \Gamma \vdash e \preceq e : \tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \Lambda\zeta.e \preceq \Lambda\zeta.e : \forall\zeta.\tau$$

Proof. Expanding the conclusion, we must show that given

$$\begin{aligned} & \forall \rho, \gamma_{\Gamma}, \gamma_{\mathbf{L}}, W, \mathsf{H}_1, \mathsf{H}_2. \\ & \rho.\mathbf{F} \in \mathcal{D}[\Delta] \wedge \rho.\mathbf{L3} \in \mathcal{D}[\Delta] \wedge (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho} \wedge (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}.\Gamma) \in \mathcal{G}[\Gamma]_{\rho} \\ & \wedge \gamma_{\mathbf{L}}.\Delta = \gamma_{\text{locs}}(\rho.\mathbf{L3}) \end{aligned}$$

it holds that:

$$(W, (\mathsf{H}_1, \lambda x_{\zeta}. \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^+))), (\mathsf{H}_2, \lambda x_{\zeta}. \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^+)))) \in \mathcal{E}[\forall \zeta. \tau]_{\rho}$$

By Lemma C.0.8, it suffices to show that:

$$(W, (\mathsf{H}_1, \lambda x_{\zeta}. \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^+))), (\mathsf{H}_2, \lambda x_{\zeta}. \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^+)))) \in \mathcal{V}[\forall \zeta. \tau]_{\rho}$$

By expanding the value relation, for any locations ℓ_1, ℓ_2 , we must show

$$(W, (\mathsf{H}_1, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^+))), (\mathsf{H}_2, \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^+)))) \in \mathcal{E}[\tau]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_1, \ell_2)]}$$

Let ρ' be a record such that $\rho'.\mathbf{F} = \rho.\mathbf{F}$ and $\rho'.\mathbf{L3} = \rho.\mathbf{L3}[\zeta \mapsto (\ell_1, \ell_2)]$. It is easy to see $\rho'.\mathbf{L3} \in \mathcal{D}[\Delta, \zeta]$, given that $\rho.\mathbf{L3} \in \mathcal{D}[\Delta]$. Thus, we can instantiate the first induction hypothesis with $\rho', \gamma_{\Gamma}, \gamma_{\mathbf{L}}, W, \mathsf{H}_1, \mathsf{H}_2$, which suffices to show the above statement. \square

Lemma C.0.43 (Compat $\mathbf{e}[\zeta']$). *If $\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{e} \preceq \mathbf{e} : \forall \zeta. \tau$ and $\zeta' \in \Delta$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \mathbf{e}[\zeta'] \preceq \mathbf{e}[\zeta'] : [\zeta \mapsto \zeta']\tau$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\cdot]$, and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{1+} : MHeap. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\zeta \mapsto \zeta']\tau]_{\rho} \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^+)) ()) \xrightarrow{*} L_2 \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \not\rightarrow \end{aligned}$$

given arbitrary $\rho, \gamma_{\mathbf{L}}, \gamma_{\Gamma}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : MHeap, \mathsf{H}_{1*}$, such that

$$\begin{aligned} & \rho.\mathbf{L3} \in \mathcal{D}[\Delta], \rho.\mathbf{F} \in \mathcal{D}[\Delta], (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho}, \\ & (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}.\Gamma) \in \mathcal{G}[\Gamma]_{\rho}, \gamma_{\mathbf{L}}.\Delta = \gamma_{\text{locs}}(\rho.\mathbf{L3}) \end{aligned}$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}^+)) ()) \xrightarrow{*} L_1 (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

First, we can instantiate the first induction hypothesis with $\rho, \gamma_{\text{L}}, \gamma_{\text{L}}^1, \mathbf{H}_1, \mathbf{H}_2$ to find that:

$$(W, (\mathbf{H}_1, \gamma_{\text{L}}^1(\gamma_{\text{L}}^1(\mathbf{e}^+))), (\mathbf{H}_2, \gamma_{\text{L}}^2(\gamma_{\text{L}}^2(\mathbf{e}^+)))) \in \mathcal{E}[\![\forall \zeta. \tau]\!]_{\rho}$$

Thus, we find

$$(\mathbf{H}_{1g+} \uplus \mathbf{H}_1 \uplus \mathbf{H}_{1+}, \gamma_{\text{L}}^1(\gamma_{\text{L}}^1(\mathbf{e}^+))) \xrightarrow{*_{L_1}} (\mathbf{H}'_{1g} \uplus \mathbf{H}_1^* \uplus \mathbf{H}_{1+}, v_1^*) \not\rightarrow_{L_1}$$

and

$$(\mathbf{H}_{2g+} \uplus \mathbf{H}_2 \uplus \mathbf{H}_{2+}, \gamma_{\text{L}}^2(\gamma_{\text{L}}^2(\mathbf{e}^+))) \xrightarrow{*_{L_2}} (\mathbf{H}'_{2g} \uplus \mathbf{H}_2^* \uplus \mathbf{H}_{2+}, v_2^*) \not\rightarrow_{L_2}$$

where $\mathbf{H}'_{1g}, \mathbf{H}'_{2g} : W'$ for some

$$W \sqsubseteq_{(\text{dom}(\mathbf{H}_{1+}), \text{dom}(\mathbf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathbf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathbf{H}_{2+})))} W'$$

and

$$(W', (\mathbf{H}_1^*, v_1^*), (\mathbf{H}_2^*, v_2^*)) \in \mathcal{V}[\![\forall \zeta. \tau]\!]_{\rho}$$

By expanding the value relation, we find $v_1^* = \lambda_{-}.\mathbf{e}_b^*$ and $v_2^* = \lambda_{-}.\mathbf{e}_b^{\dagger}$ where

$$(W', (\mathbf{H}_1^*, \mathbf{e}_b^*), (\mathbf{H}_2^*, \mathbf{e}_b^{\dagger})) \in \mathcal{E}[\![\tau]\!]_{\rho[\text{L3}(\zeta) \mapsto (\ell_1, \ell_2)]} \quad (45)$$

Ergo, the original configuration steps as follows:

$$\begin{aligned} & (\mathbf{H}_{1g+} \uplus \mathbf{H}_1 \uplus \mathbf{H}_{1+}, \gamma_{\text{L}}^1(\gamma_{\text{L}}^1(\mathbf{e}^+)) ()) \xrightarrow{*_{L_1}} *_{L_1} \\ & (\mathbf{H}'_{1g} \uplus \mathbf{H}_1^* \uplus \mathbf{H}_{1+}, \lambda_{-}.\mathbf{e}_b^* ()) \xrightarrow{*_{L_1}} \\ & (\mathbf{H}'_{1g} \uplus \mathbf{H}_1^* \uplus \mathbf{H}_{1+}, \mathbf{e}_b^*) \end{aligned}$$

and

$$\begin{aligned} & (\mathbf{H}_{2g+} \uplus \mathbf{H}_2 \uplus \mathbf{H}_{2+}, \gamma_{\text{L}}^2(\gamma_{\text{L}}^2(\mathbf{e}^+)) ()) \xrightarrow{*_{L_2}} *_{L_2} \\ & (\mathbf{H}'_{2g} \uplus \mathbf{H}_2^* \uplus \mathbf{H}_{2+}, \mathbf{e}_b^{\dagger}) \end{aligned}$$

Next, by the fact that the configuration on the left-hand side terminates, we have

$$(\mathbf{H}'_{1g} \uplus \mathbf{H}_1^* \uplus \mathbf{H}_{1+}, \mathbf{e}_b^*) \xrightarrow{*_{L_1}} (\mathbf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

Then, by applying (45), we find that

$$(\mathbf{H}_{1*}, v_1) = (\mathbf{H}''_{1g} \uplus \mathbf{H}_1^{**} \uplus \mathbf{H}_{1+}, v_1^f)$$

and

$$(\mathbf{H}'_{2g} \uplus \mathbf{H}_2^* \uplus \mathbf{H}_{2+}, \mathbf{e}_b^{\dagger}) \xrightarrow{*_{L_2}} (\mathbf{H}''_{2g} \uplus \mathbf{H}_2^{**} \uplus \mathbf{H}_{2+}, v_2^f) \not\rightarrow_{L_2}$$

where $\mathbf{H}''_{1g}, \mathbf{H}''_{2g} : W''$ for some

$$W' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W', L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W''$$

and

$$(W'', (\mathsf{H}_1^{**}, v_1^f), (\mathsf{H}_2^{**}, v_2^f)) \in \mathcal{V}[[\tau]]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_1, \ell_2)]}$$

Then, by Lemma C.0.6, we find

$$(W'', (\mathsf{H}_1^{**}, v_1^f), (\mathsf{H}_2^{**}, v_2^f)) \in \mathcal{V}[[\zeta \mapsto \zeta']\tau]_{\rho}$$

Finally, we can take $\mathsf{H}'_1 = \mathsf{H}_1^{**}$, $\mathsf{H}'_2 = \mathsf{H}_2^{**}$, $W' = W''$, $\mathsf{H}'_{1g} = \mathsf{H}_{1g}''$, and $\mathsf{H}'_{2g} = \mathsf{H}_{2g}''$. Notice that $W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W''$ by Lemma C.0.2. This suffices to finish the proof. \square

Lemma C.0.44 (Compat $\vdash \zeta, e \sqsupseteq e$). *If $\Delta; \Gamma; \Delta; \Gamma \vdash e \sqsupseteq e : [\zeta \mapsto \zeta']\tau$, then*

$$\Delta; \Gamma; \Delta; \Gamma \vdash \vdash \zeta', e \sqsupseteq \zeta', e : \exists \zeta. \tau$$

Proof. Expanding the definition of \sqsupseteq , \cdot^+ , $\mathcal{E}[[\cdot]]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : MHeap. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[[\zeta \mapsto \zeta']\tau]_{\rho} \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(e^+))) \xrightarrow{*} \mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \xrightarrow{*} \end{aligned}$$

given arbitrary $\rho, \gamma_{\mathbf{L}}, \gamma_{\Gamma}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : W, v_1, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_{1+} : MHeap, \mathsf{H}_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[[\Delta]], \rho.\mathbf{F} \in \mathcal{D}[[\Delta]], (W, \gamma_{\Gamma}) \in \mathcal{G}[[\Gamma]]_{\rho}, (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}) \in \mathcal{G}[[\Gamma]]_{\rho}$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(e^+))) \xrightarrow{*} (\mathsf{H}_{1*}, v_1) \xrightarrow{*} \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+}, v_1)$$

First, we can instantiate the first induction hypothesis with $\rho, \gamma_{\Gamma}, \gamma_{\mathbf{L}}, W, \mathsf{H}_1, \mathsf{H}_2$ to find that:

$$(W, (\mathsf{H}_1, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(e^+))), (\mathsf{H}_2, \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(e^+)))) \in \mathcal{E}[[\zeta \mapsto \zeta']\tau]_{\rho}$$

Thus, by Lemma C.0.14, we find

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(e^+))) \xrightarrow{*} (\mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+}, v_1) \xrightarrow{*} \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+}, v_1)$$

and

$$(\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}^+))) \xrightarrow{*_{L_2}} (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2^*) \not\rightarrow_{L_2}$$

where $\mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W'$ for some $W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W'$ and

$$(W', (\mathsf{H}_1^*, v_1^*), (\mathsf{H}_2^*, v_2^*)) \in \mathcal{V}[[\zeta \mapsto \zeta']\tau]_{\rho}$$

Then, we can take $\mathsf{H}'_1 = \mathsf{H}_1^*$, $\mathsf{H}'_2 = \mathsf{H}_2^*$, $W' = W$, $\mathsf{H}'_{1g} = \mathsf{H}'_{1g}$, and $\mathsf{H}'_{2g} = \mathsf{H}'_{2g}$. Thus, it suffices to show:

$$(W', (\mathsf{H}_1^*, v_1^*), (\mathsf{H}_2^*, v_2^*)) \in \mathcal{V}[\exists \zeta. \tau]_{\rho}$$

By expanding the value relation, it suffices to show:

$$(W', (\mathsf{H}_1^*, v_1^*), (\mathsf{H}_2^*, v_2^*)) \in \mathcal{V}[\tau]_{\rho[\mathbf{L3}(\zeta) \mapsto (\ell_1, \ell_2)]}$$

The above statement must hold by Lemma C.0.6 because we have that $(W', (\mathsf{H}_1^*, v_1^*), (\mathsf{H}_2^*, v_2^*)) \in \mathcal{V}[[\zeta \mapsto \zeta']\tau]_{\rho}$ from earlier, which suffices to finish the proof. \square

Lemma C.0.45 (Compat let $\lceil \zeta, x \rceil$). *If $\Delta; \Gamma; \Delta; \Gamma_1 \vdash e_1 \preceq e_1 : \exists \zeta. \tau_1$, $\Delta; \Gamma; \Delta, \zeta; \Gamma_2, x : \tau_1 \vdash e_2 \preceq e_2 : \tau_2$ and $\text{FLV}(\tau_2) \subseteq \Delta$, then*

$$\Delta; \Gamma; \Delta; \Gamma_1 \uplus \Gamma_2 \vdash \text{let } \lceil \zeta, x \rceil = e_1 \text{ in } e_2 \preceq \text{let } \lceil \zeta, x \rceil = e_1 \text{ in } e_2 : \tau_2$$

Proof. Expanding the definition of \preceq , \cdot^+ , $\mathcal{E}[\cdot]$. and pushing substitutions in the goal, we are to show that

$$\begin{aligned} & \exists \mathsf{H}'_1, \mathsf{H}'_{1g}. \forall \mathsf{H}_{2+} : M\text{Heap}. \exists \mathsf{H}'_2, W', \mathsf{H}'_{2g}, v_2. \\ & \mathsf{H}_{1*} = \mathsf{H}'_{1g} \uplus \mathsf{H}'_1 \uplus \mathsf{H}_{1+} \wedge \mathsf{H}'_{1g}, \mathsf{H}'_{2g} : W' \wedge \\ & W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W' \wedge \\ & (W', (\mathsf{H}'_1, v_1), (\mathsf{H}'_2, v_2)) \in \mathcal{V}[\tau_2]_{\rho} \wedge \\ & (\mathsf{H}_{2g+} \uplus \mathsf{H}_2 \uplus \mathsf{H}_{2+}, \text{let } x = \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+)) \text{ in } \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \xrightarrow{*_{L_2}} \\ & (\mathsf{H}'_{2g} \uplus \mathsf{H}'_2 \uplus \mathsf{H}_{2+}, v_2) \end{aligned}$$

given arbitrary $\rho, \gamma_{\mathbf{L}}, \gamma_{\Gamma}, W, L_1, L_2, \mathsf{H}_{1g+}, \mathsf{H}_{2g+} : M\text{Heap}, \mathsf{H}_{1*}$, such that

$$\rho.\mathbf{L3} \in \mathcal{D}[\Delta], \rho.\mathbf{F} \in \mathcal{D}[\Delta], (W, \gamma_{\Gamma}) \in \mathcal{G}[\Gamma]_{\rho}, (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}) \in \mathcal{G}[\Gamma_1 \uplus \Gamma_2]_{\rho}$$

and

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_1 \uplus \mathsf{H}_{1+}, \text{let } x = \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_1^+)) \text{ in } \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1}} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

Then, by Lemma C.0.9, there exist $\gamma_{\mathbf{L}1}, \gamma_{\mathbf{L}2}, \mathbf{H}_{1l}, \mathbf{H}_{1r}, \mathbf{H}_{2l}, \mathbf{H}_{2r}$ such that $\gamma_{\mathbf{L}} = \gamma_{\mathbf{L}1} \uplus \gamma_{\mathbf{L}2}$, $\mathbf{H}_1 = \mathbf{H}_{1l} \uplus \mathbf{H}_{1r}$, $\mathbf{H}_2 = \mathbf{H}_{2l} \uplus \mathbf{H}_{2r}$,

$$(W, \mathbf{H}_{1l}, \mathbf{H}_{2l}, \gamma_{\mathbf{L}1}) \in \mathcal{G}[\![\boldsymbol{\Gamma}_1]\!]_\rho$$

$$(W, \mathbf{H}_{1r}, \mathbf{H}_{2r}, \gamma_{\mathbf{L}2}) \in \mathcal{G}[\![\boldsymbol{\Gamma}_2]\!]_\rho$$

and for all $j \in \{1, 2\}$,

$$\gamma_{\mathbf{L}}^j(\gamma_{\boldsymbol{\Gamma}}^j(\mathbf{e}_1^+)) = \gamma_{\mathbf{L}1}^j(\gamma_{\boldsymbol{\Gamma}}^j(\mathbf{e}_1^+))$$

$$\gamma_{\mathbf{L}}^j(\gamma_{\boldsymbol{\Gamma}}^j(\mathbf{e}_2^+)) = \gamma_{\mathbf{L}2}^j(\gamma_{\boldsymbol{\Gamma}}^j(\mathbf{e}_2^+))$$

Then, by instantiating the first induction hypothesis with $\rho, \gamma_{\boldsymbol{\Gamma}}, \gamma_{\mathbf{L}1}, W, \mathbf{H}_{1l}, \mathbf{H}_{2l}$, we find

$$(W, (\mathbf{H}_{1l}, \gamma_{\mathbf{L}1}^1(\gamma_{\boldsymbol{\Gamma}}^1(\mathbf{e}_1^+))), (\mathbf{H}_{2l}, \gamma_{\mathbf{L}1}^2(\gamma_{\boldsymbol{\Gamma}}^2(\mathbf{e}_1^+)))) \in \mathcal{E}[\![\exists \boldsymbol{\zeta}. \boldsymbol{\tau}_1]\!]_\rho$$

Thus, by Lemma C.0.14, we have

$$(\mathbf{H}_{1g} \uplus \mathbf{H}_{1l} \uplus \mathbf{H}_{1r} \uplus \mathbf{H}_{1+}, \gamma_{\mathbf{L}1}^1(\gamma_{\boldsymbol{\Gamma}}^1(\mathbf{e}_1^+))) \xrightarrow{*_{L_1 \cup FL(\gamma_{\mathbf{L}2}^1(\gamma_{\boldsymbol{\Gamma}}^1(\mathbf{e}_2^+)))}} (\mathbf{H}'_{1g} \uplus \mathbf{H}_{1r} \uplus \mathbf{H}_{1l}^* \uplus \mathbf{H}_{1+}, v_1^*) \not\rightarrow$$

and, for any \mathbf{H}_{2+} ,

$$(\mathbf{H}_{1g} \uplus \mathbf{H}_{2l} \uplus \mathbf{H}_{2r} \uplus \mathbf{H}_{2+}, \gamma_{\mathbf{L}1}^2(\gamma_{\boldsymbol{\Gamma}}^2(\mathbf{e}_1^+))) \xrightarrow{*_{L_2 \cup FL(\gamma_{\mathbf{L}2}^2(\gamma_{\boldsymbol{\Gamma}}^2(\mathbf{e}_2^+)))}} (\mathbf{H}'_{1g} \uplus \mathbf{H}_{2r} \uplus \mathbf{H}_{2l}^* \uplus \mathbf{H}_{2+}, v_2^*) \not\rightarrow$$

where $\mathbf{H}'_{1g}, \mathbf{H}'_{2g} : W'$ for some

$$W \sqsubseteq_{(\text{dom}(\mathbf{H}_{1r} \uplus \mathbf{H}_{1+}), \text{dom}(\mathbf{H}_{2r} \uplus \mathbf{H}_{2+})), \text{rchgclocs}(W, FL(\text{cod}(\mathbf{H}_{1r})) \cup FL(\text{cod}(\mathbf{H}_{1+})) \cup FL(\gamma_{\mathbf{L}1}^1(\gamma_{\boldsymbol{\Gamma}}^1(\mathbf{e}_2^+)))) \cup L_1, \\ FL(\text{cod}(\mathbf{H}_{2r})) \cup FL(\text{cod}(\mathbf{H}_{2+})) \cup FL(\gamma_{\mathbf{L}2}^2(\gamma_{\boldsymbol{\Gamma}}^2(\mathbf{e}_2^+))) \cup L_2)} W'$$

and

$$(W', (\mathbf{H}_{1l}^*, v_1^*), (\mathbf{H}_{2l}^*, v_2^*)) \in \mathcal{V}[\![\exists \boldsymbol{\zeta}. \boldsymbol{\tau}_1]\!]_\rho$$

By expanding the value relation, we find there exist locations ℓ_1, ℓ_2 such that, for any $i \in \{1, 2\}$,

$$(W', (\mathbf{H}_{1l}^*, v_1^*), (\mathbf{H}_{2l}^*, v_2^*)) \in \mathcal{V}[\![\boldsymbol{\tau}_1]\!]_{\rho[\mathbf{L}3(\boldsymbol{\zeta}) \mapsto (\ell_1, \ell_2)]}$$

Thus, the original configuration steps as follows:

$$(\mathbf{H}_{1g} \uplus \mathbf{H}_{1l} \uplus \mathbf{H}_{1r} \uplus \mathbf{H}_{1+}, \text{let } x = \gamma_{\mathbf{L}}^1(\gamma_{\boldsymbol{\Gamma}}^1(\mathbf{e}_1^+)) \text{ in } \gamma_{\mathbf{L}}^1(\gamma_{\boldsymbol{\Gamma}}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1}} *_{L_1} \\ (\mathbf{H}'_{1g} \uplus \mathbf{H}_{1r} \uplus \mathbf{H}_{1+} \uplus \mathbf{H}_{1l}^*, \text{let } x = v_1^* \text{ in } \gamma_{\mathbf{L}}^1(\gamma_{\boldsymbol{\Gamma}}^1(\mathbf{e}_2^+))) \xrightarrow{*_{L_1}} *_{L_1} \\ (\mathbf{H}'_{1g} \uplus \mathbf{H}_{1r} \uplus \mathbf{H}_{1+} \uplus \mathbf{H}_{1l}^*, [x \mapsto v_1^*] \gamma_{\mathbf{L}}^1(\gamma_{\boldsymbol{\Gamma}}^1(\mathbf{e}_2^+)))$$

and similarly

$$(\mathsf{H}_{1g+} \uplus \mathsf{H}_{2l} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+}, \text{let } x = \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_1^+)) \text{ in } \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \xrightarrow{*} *_{L_2} \\ (\mathsf{H}'_{1g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2+} \uplus \mathsf{H}_{2l}^*, [x \mapsto v_2^*] \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+)))$$

Let $\gamma'_{\mathbf{L}2} = \gamma_{\mathbf{L}2}[x \mapsto (v_1^*, v_2^*)]$.

First, one can see that

$$(W', \mathsf{H}_{1r} \uplus \mathsf{H}_{1l}^*, \mathsf{H}_{2r} \uplus \mathsf{H}_{2l}^*, \gamma'_{\mathbf{L}2}) \in \mathcal{G}[\![\mathbf{T}_1, x : \tau_1]\!]_{\rho[\mathbf{L}3(\zeta) \mapsto (\ell_1, \ell_2)]}$$

because $(W', (\mathsf{H}_{1l}^*, v_1^*), (\mathsf{H}_{2l}^*, v_2^*)) \in \mathcal{V}[\![\tau_1]\!]_{\rho[\mathbf{L}3(\zeta) \mapsto (\ell_1, \ell_2)]}$ and $(W', \mathsf{H}_{1r}, \mathsf{H}_{2r}, \gamma_{\mathbf{L}2}) \in \mathcal{G}[\![\mathbf{T}_2]\!]_{\rho}$ (by Lemma C.0.3, and extending ρ with ζ does not invalidate any atoms in the substitution).

Thus, since $\mathcal{G}[\![\Gamma]\!]_{\rho}, \mathcal{G}[\![\mathbf{T}_1 \uplus \mathbf{T}_2]\!]_{\rho}$ are closed under world extension by Lemma C.0.3, we can instantiate the second induction hypothesis with $\rho[\mathbf{L}3(\zeta) \mapsto (\ell_1, \ell_2)], \gamma_{\Gamma}, \gamma'_{\mathbf{L}2}, W', \mathsf{H}_{1r} \uplus \mathsf{H}_{1l}^*, \mathsf{H}_{2r} \uplus \mathsf{H}_{2l}^*$ to find

$$(W', (\mathsf{H}_{1r} \uplus \mathsf{H}_{1l}^*, [x \mapsto v_1^*] \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))), (\mathsf{H}_{2r} \uplus \mathsf{H}_{2l}^*, [x \mapsto v_2^*] \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \in \mathcal{E}[\![\tau_2]\!]_{\rho[\mathbf{L}3(\zeta) \mapsto (\ell_1, \ell_2)]} \quad (46)$$

Next, by the assumption that the configuration on the left-hand side terminates, we have

$$(\mathsf{H}'_{1g} \uplus \mathsf{H}_{1+} \uplus \mathsf{H}_{1r} \uplus \mathsf{H}_{1l}^*, [x \mapsto v_1^*] \gamma_{\mathbf{L}}^1(\gamma_{\Gamma}^1(\mathbf{e}_2^+))) \xrightarrow{*} *_{L_1} (\mathsf{H}_{1*}, v_1) \not\rightarrow_{L_1}$$

Ergo, by applying (46), we have

$$(\mathsf{H}_{1*}, v_1) = (\mathsf{H}_{1g}'' \uplus \mathsf{H}_{1f} \uplus \mathsf{H}_{1+}, v_1^{\mathsf{f}})$$

and

$$(\mathsf{H}'_{2g} \uplus \mathsf{H}_{2r} \uplus \mathsf{H}_{2l}^* \uplus \mathsf{H}_{2+}, [x \mapsto v_2^*] \gamma_{\mathbf{L}}^2(\gamma_{\Gamma}^2(\mathbf{e}_2^+))) \xrightarrow{*} *_{L_2} (\mathsf{H}_{2g}'' \uplus \mathsf{H}_{2f} \uplus \mathsf{H}_{2+}, v_2^{\mathsf{f}}) \not\rightarrow_{L_2}$$

where H_{1g}'' , $\mathsf{H}_{2g}'' : W''$ for some $W' \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W', L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W''$ and

$$(W'', (\mathsf{H}_{1f}, v_1^{\mathsf{f}}), (\mathsf{H}_{2f}, v_2^{\mathsf{f}})) \in \mathcal{V}[\![\tau_2]\!]_{\rho[\mathbf{L}3(\zeta) \mapsto (\ell_1, \ell_2)]}$$

Then, by Lemma C.0.7, since $\text{FLV}(\tau_2) \subseteq \Delta$,

$$(W'', (\mathsf{H}_{1f}, v_1^{\mathsf{f}}), (\mathsf{H}_{2f}, v_2^{\mathsf{f}})) \in \mathcal{V}[\![\tau_2]\!]_{\rho}$$

Finally, we can take $\mathsf{H}'_1 = \mathsf{H}_{1f}$, $\mathsf{H}'_2 = \mathsf{H}_{2f}$, $W' = W''$, $\mathsf{H}'_{1g} = \mathsf{H}_{1g}''$, and $\mathsf{H}'_{2g} = \mathsf{H}_{2g}''$. Notice that $W \sqsubseteq_{(\text{dom}(\mathsf{H}_{1+}), \text{dom}(\mathsf{H}_{2+})), \text{rchgclocs}(W, L_1 \cup \text{FL}(\text{cod}(\mathsf{H}_{1+})), L_2 \cup \text{FL}(\text{cod}(\mathsf{H}_{2+})))} W''$ by Lemma C.0.2. This suffices to finish the proof. \square

Lemma C.0.46 (Compat $(\llbracket e \rrbracket_\tau)$). *If $\Delta; !\Gamma; \Delta; \Gamma \vdash e \preceq e : \tau$ and $\tau \sim \tau$, then*

$$\Delta; \Gamma; \Delta; !\Gamma \vdash (\llbracket e \rrbracket_\tau) \preceq (\llbracket e \rrbracket_\tau) : \tau$$

Proof. Expanding the definition of \preceq and \cdot^+ and pushing substitutions in the goal, we are to show that

$$(W, (\mathsf{H}_1, C_{\tau \mapsto \tau}(\gamma_{\mathbf{L}}^1(\gamma_{\mathbf{L}}^1(e^+)))), (\mathsf{H}_2, C_{\tau \mapsto \tau}(\gamma_{\mathbf{L}}^2(\gamma_{\mathbf{L}}^2(e^+))))) \in \mathcal{E}[\llbracket \tau \rrbracket_\rho] \quad (47)$$

given $\rho, \gamma_{\mathbf{L}}, W, \mathsf{H}_1, \mathsf{H}_2$ such that

$$\begin{aligned} \rho.\mathbf{F} &\in \mathcal{D}[\llbracket \Delta \rrbracket], \rho.\mathbf{L3} \in \mathcal{D}[\llbracket \Delta \rrbracket], (W, \gamma_{\mathbf{L}}) \in \mathcal{G}[\llbracket \Gamma \rrbracket_\rho], \\ (W, \mathsf{H}_1, \mathsf{H}_2, \gamma_{\mathbf{L}}.\Gamma) &\in \mathcal{G}[\llbracket !\Gamma \rrbracket_\rho], \gamma_{\mathbf{L}}.\Delta = \gamma_{\text{locs}}(\rho.\mathbf{L3}) \end{aligned}$$

Our first induction hypothesis, appropriately instantiated, tells us that:

$$(W, (\mathsf{H}_1, \gamma_{\mathbf{L}}^1(\gamma_{\mathbf{L}}^1(e^+))), (\mathsf{H}_2, \gamma_{\mathbf{L}}^2(\gamma_{\mathbf{L}}^2(e^+)))) \in \mathcal{E}[\llbracket \tau \rrbracket_\rho]$$

Since $\tau \sim \tau$, we have (47) by Theorem C.0.15. \square

Lemma C.0.47 (Fundamental Property). *If $\Delta; \Gamma; \Delta; \Gamma \vdash e : \tau$, then $\Delta; \Gamma; \Delta; \Gamma \vdash e \preceq e : \tau$ and if $\Delta; \Gamma; \Delta; \Gamma \vdash e : \tau$, then $\Delta; \Gamma; \Delta; \Gamma \vdash e \preceq e : \tau$.*

Proof. By induction on typing derivation, relying on the following compatibility lemmas, which have to exist for every typing rule in both source languages. \square

Theorem C.0.48 (Type Safety for **Miniml**). *If $\cdot; \cdot; \cdot; \cdot \vdash e : \tau$, then for any heap H , if $(H, e^+) \xrightarrow{*} (H', e')$, either there exist H'', e'' such that $(H', e') \rightarrow (H'', e'')$ or e' is a value.*

Proof. By the fundamental property, since the environments under which e is typechecked are empty, $(\cdot, (\emptyset, e^+), (\emptyset, e^+)) \in \mathcal{E}[\llbracket \tau \rrbracket]$.

Then, either $(H', e') \rightarrow (H'', e'')$ or (H', e') is irreducible. If (H', e') is irreducible, we can apply the expression relation and find that there exists a world W and expression v_2 such that $(W, (\emptyset, e'), (\emptyset, v_2)) \in \mathcal{V}[\llbracket \tau \rrbracket]$. Since expressions in the value relation are target values, this suffices to show that e' is a value. \square

Theorem C.0.49 (Type Safety for **L3**). *If $\cdot; \cdot; \cdot; \cdot \vdash e : \tau$, then for any heap H , if $(H, e^+) \xrightarrow{*} (H', e')$, either there exist H'', e'' such that $(H', e') \rightarrow (H'', e'')$ or e' is a value.*

Proof. By the fundamental property, since the environments under which e is typechecked are empty, $(\cdot, (\emptyset, e^+), (\emptyset, e^+)) \in \mathcal{E}[\llbracket \tau \rrbracket]$.

Then, either $(H', e') \rightarrow (H'', e'')$ or (H', e') is irreducible. If (H', e') is irreducible, we can apply the expression relation and find that

there exists a world W , heaps H'_1, H'_2 , and an expression v_2 such that $(W, (H'_1, e'), (H'_2, v_2)) \in \mathcal{V}[\tau]..$ Since expressions in the value relation are target values, this suffices to show that e' is a value. \square

D

B E H A V I O R I N T E R O P E R A B I L I T Y : M U T A B L E S T A T E

D.0.1 *Supporting Lemmas*

Lemma D.0.1 (relevant locations subset). *If $\varphi_1 \subset \varphi$, $\varphi_2 \subset \varphi$, and $H :_{\varphi} W$ then if $\varphi_1 = \text{flocs}(P)$, $\langle H ; S ; P \rangle \xrightarrow{*} \langle H^1 ; S^1 ; P^1 \rangle$, and for some φ'_1 , $W^1 \sqsubseteq W$, $H^1 :_{\varphi'_1 \cup \varphi_1} W^1$, then $H_1 :_{\varphi_2} W^1$.*

Proof. Consider what needs to be true for $H^1 :_{\varphi_2} W^1$. For every location in φ_2 , either it is marked as dead in W^1 , or the location must be in H^1 and must map to a value in the relation described by W^1 . Since we know that $\varphi_2 \subset \varphi$ and $H :_{\varphi} W$, we have a starting point at which these facts held. Since $W^1 \sqsubseteq W$, we know the only changes to the world can be adding locations or marking existing locations as dead. Since $H^1 :_{\varphi_1} W^1$, we know that anything in $\varphi_2 \cap \varphi_1$ is satisfied. What about locations not in that set? Since $\varphi_1 = \text{flocs}(P)$, we know the program only knew about the locations in φ_1 —there is no way for an existing location to be synthesized out of thin air—and thus any locations in $\varphi_2 \setminus \varphi_1$ will have been unchanged between H and H_1 , so we are done. \square

Corollary D.0.2 (Antireduction λ).

If $\forall k \varphi' H H', S. (k - j, \varphi', \text{push } v_1; \text{push } v_2; \dots \text{push } v_n; P) \in \mathcal{E}^{\lambda}[\tau]$, and $\langle H ; S ; P' ; P \rangle \xrightarrow{j} \langle H' ; S, v_1, v_2, \dots, v_n ; P \rangle$ then $(k, \varphi, P' ; P) \in \mathcal{E}^{\lambda}[\tau]$.

Proof. Our obligation is to show that

$$\begin{aligned} & \forall H, H', S, S', j < k. \langle H ; S ; P' ; P \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle \\ & \implies (S' = \text{Fail } c \wedge c \in \text{OKERR}) \vee \exists v. (S' = S, v \wedge (k - j, v) \in \mathcal{V}^{\lambda}[\tau]) \end{aligned}$$

From our second hypothesis, we know that

$$\langle H ; S ; P' ; P \rangle \xrightarrow{j'} \langle H^* ; v_1, \dots, v_n ; P \rangle \xrightarrow{j-j'} \langle H' ; S' ; \cdot \rangle$$

Our first hypothesis then tells us that

$$(S' = \text{Fail } c \wedge c \in \text{OKERR}) \vee \exists v. (S' = S, v \wedge ((k - j') - (j - j'), v) \in \mathcal{V}^{\lambda}[\tau])$$

which suffices to complete the proof.

□

Corollary D.0.3 (Antireduction S).

If $\forall W' \varphi' H H', S. (W', \varphi', \text{push } v_1; \text{push } v_2; \dots \text{push } v_n; P) \in \mathcal{E}^S[\tau]$ and $W' \sqsubseteq W, H :_\varphi W, H' :_{\varphi \cup \varphi'} W'$, and $\langle H ; S ; P' ; P \rangle \xrightarrow{*} \langle H' ; S, v_1, v_2, \dots, v_n ; P \rangle$ then $(W, \varphi, P'; P) \in \mathcal{E}^S[\tau]$.

Proof. We consider heap $H :_\varphi W$, arbitrary stack S . We know that if the term in question does not run forever (which, if it does, then the suffix P does as well, so we are done), then it steps to a terminal configuration $\langle H^F ; S^F ; \cdot \rangle$. We need to show that, assuming that is not an error, $S^F = S, v$ and for some φ^F and $W^F \sqsubseteq W, H^F :_{\varphi^F} W^F$ and $(W^F, \varphi^F, v) \in \mathcal{V}^S[\tau]$. We know that $\langle H ; S ; P' ; P \rangle \xrightarrow{*} \langle H' ; S, v_1, \dots, v_n ; P \rangle$ and that for some $W' \sqsubseteq W, H' :_{\varphi \cup \varphi'} W'$. So we instantiate our first hypothesis with H' and S . After n steps, it is in exactly the configuration our term left off in. We know it doesn't run forever, and if it errors, similarly, our overall term must error, so we conclude that it runs to a terminal configuration which due to confluence, will be the same one. Thus, we know $H^F :_{\varphi \cup \varphi' \cup \varphi^F} W^F$, which is stronger than we need, and $(W^F, \varphi^F, v) \in \mathcal{V}^S[\tau]$, exactly as needed. □

D.0.2 *FunLang Compatibility Lemmas*

$$\begin{aligned} \llbracket I; \Gamma \vdash P : \tau \rrbracket &\equiv \\ \forall k \gamma. \forall ((k, \emptyset), \emptyset, \gamma^{I^S}) \in \mathcal{G}^S[I^S]. \forall ((k, \emptyset), \emptyset, \gamma^{I^X}) \in \mathcal{G}^X[I^X]. (k, \gamma) \in \mathcal{G}^\lambda[\Gamma] &\implies \\ (k, \gamma^{I^X}(\gamma^{I^S}(\gamma(P)))) \in \mathcal{E}^\lambda[\tau] \end{aligned}$$

We now state and prove all the compatibility lemmas for our source language. Note that we have to prove these *three times*: once for each model, though the boundary terms only exist at the top level, and they are the most challenging/interesting.

Lemma D.0.4 (unit). *Show that $\llbracket I; \Gamma \vdash \text{push } 0 : \text{unit} \rrbracket$.*

Proof. Since 0 has no free variables, what we need to show is that $(k, \text{push } 0) \in \mathcal{E}^\lambda[\text{unit}]$. Given any H, γ , we can see that we take one step from $\langle H ; \gamma ; \text{push } 0 \rangle$ to $\langle H ; \gamma, 0 ; \cdot \rangle$, and thus provided k was larger than 1 (else, trivial), what remains to show is that $(k-1, 0) \in \mathcal{V}^\lambda[\text{unit}]$. But this is trivial by the definition of the value relation.

□

Lemma D.0.5 (bool). *Show for any n , $\llbracket I; \Gamma \vdash \text{push } n : \text{bool} \rrbracket$.*

Proof. This proof is identical to that of *unit*. □

Lemma D.0.6 (if). *If $\llbracket I; \Gamma \vdash P : \text{bool} \rrbracket$, $\llbracket I; \Gamma \vdash P_1 : \tau \rrbracket$, and $\llbracket I; \Gamma \vdash P_2 : \tau \rrbracket$ then*

$$\llbracket I; \Gamma \vdash P; \text{if } 0 \ P_1 \ P_2 : \tau \rrbracket.$$

Proof. We are given $(k, \gamma) \in \mathcal{G}^\lambda[\Gamma]$, $((k, \emptyset), \emptyset, \gamma^{\text{I}^S}) \in \mathcal{G}^S[\text{I}^S]$, $((k, \emptyset), \emptyset, \gamma^{\text{I}^X}) \in \mathcal{G}^X[\text{I}^X]$, and need to show that $(k, \gamma^{\text{I}^X}(\gamma^{\text{I}^S}(\gamma(P; \text{if0 } P_1 P_2)))) \in \mathcal{E}^\lambda[\tau]$.

Pushing the substitutions in and combining $\gamma^{\text{I}^X} \circ \gamma^{\text{I}^S} \circ \gamma$ (for compactness) to γ^{I} , we refine this slightly to:

$$(k, \gamma^{\text{I}}(P); \text{if0 } \gamma^{\text{I}}(P_1) \gamma^{\text{I}}(P_2)) \in \mathcal{E}^\lambda[\tau]$$

Applying Lemma D.0.2, it suffices to show

$$(k - j, \text{push } v_1; \text{if0 } \gamma^{\text{I}}(P_1) \gamma^{\text{I}}(P_2)) \in \mathcal{E}^\lambda[\tau]$$

, since from the first hypothesis we know $\gamma^{\text{I}}(P)$ will reduce to some value v_1 in $\mathcal{V}^\lambda[\text{bool}]$. We now appeal to Lemma D.0.2 again, finishing the proof by noting that if v_1 is 0 then the induction hypothesis on $\gamma^{\text{I}}(P_1)$ suffices, and if it is not, the induction hypothesis on $\gamma^{\text{I}}(P_2)$ suffices.

□

Lemma D.0.7 (int). *For any n, show $[\![I; \Gamma \vdash \text{push } n : \text{int}]\!]$.*

Proof. This case is analogous to **unit** and **bool**. □

Lemma D.0.8 (op=). *If $[\![I; \Gamma \vdash P_1 : \text{int}]\!]$ and $[\![I; \Gamma \vdash P_2 : \text{int}]\!]$, show that $[\![I; \Gamma \vdash P_1; P_2; \text{equal?} : \text{bool}]\!]$.*

Proof. We are given $(k, \gamma) \in \mathcal{G}^\lambda[\Gamma]$, $((k, \emptyset), \emptyset, \gamma^{\text{I}^S}) \in \mathcal{G}^S[\text{I}^S]$, $((k, \emptyset), \emptyset, \gamma^{\text{I}^X}) \in \mathcal{G}^X[\text{I}^X]$, and need to show that $(k, \gamma^{\text{I}^X}(\gamma^{\text{I}^S}(\gamma(P_1; P_2; \text{equal?})))) \in \mathcal{E}^\lambda[\text{bool}]$.

Pushing the substitutions in and combining $\gamma^{\text{I}^X} \circ \gamma^{\text{I}^S} \circ \gamma$ (for compactness) to γ^{I} , we refine this slightly to:

$$(k, \gamma^{\text{I}}(P_1); \gamma^{\text{I}}(P_2); \text{equal?}) \in \mathcal{E}^\lambda[\text{bool}]$$

We apply Lemma D.0.2 twice, appealing to our inductive hypotheses to reduce our obligation to showing that

$$(k', \text{push } v_1; \text{push } v_2; \text{equal?}) \in \mathcal{E}^\lambda[\text{bool}]$$

for some v_1 and v_2 in $\mathcal{V}^\lambda[\text{int}]$. Since v_1 and v_2 are both integers, the term steps to either 0 or 1 on the stack, which means we satisfy our requirement to be in $\mathcal{V}^\lambda[\text{bool}]$, sufficient to complete the proof. □

Lemma D.0.9 (op- i). *If $[\![I; \Gamma \vdash P_1 : \text{int}]\!]$ and $[\![I; \Gamma \vdash P_2 : \text{int}]\!]$, show that $[\![I; \Gamma \vdash P_1; P_2; \text{less?} : \text{bool}]\!]$.*

Proof. This proof is identical to that of $=$. □

Lemma D.0.10 (op+). *If $[\![I; \Gamma \vdash P_1 : \text{int}]\!]$ and $[\![I; \Gamma \vdash P_2 : \text{int}]\!]$, show that $[\![I; \Gamma \vdash P_1; P_2; \text{add} : \text{int}]\!]$.*

Proof. This proof is identical to that of $=$. \square

Lemma D.0.11 (var). *For any $x : \tau \in \Gamma$, show that $\llbracket I; \Gamma \vdash \text{push } x : \tau \rrbracket$.*

Proof. We are given $(k, \gamma) \in \mathcal{G}^\lambda[\Gamma]$, $((k, \emptyset), \emptyset, \gamma^{\text{I}^S}) \in \mathcal{G}^S[\text{I}^S]$, $((k, \emptyset), \emptyset, \gamma^{\text{I}^X}) \in \mathcal{G}^X[\text{I}^X]$, and need to show that $(k, \gamma^{\text{I}^X}(\gamma^{\text{I}^S}(\gamma(\text{push } x)))) \in \mathcal{E}^\lambda[\tau]$.

Since $x \in \Gamma$, it isn't in I , and thus we can eliminate the other substitutions. Further, we know from the definition of $\mathcal{G}^\lambda[\Gamma]$ that there exists some v with $(k, v) \in \mathcal{V}^\lambda[\tau]$ such that $\gamma(x) = v$. This means we can substitute, yielding this as a goal:

$$(k, \text{push } v) \in \mathcal{E}^\lambda[\tau]$$

Now we can choose an arbitrary heap H and stack S , take one step, and end up in a terminal state with stack S, v . Since $(k, v) \in \mathcal{V}^\lambda[\tau]$, we are done.

\square

Lemma D.0.12 (pair). *If $\llbracket I; \Gamma \vdash P_1 : \tau_1 \rrbracket$ and $\llbracket I; \Gamma \vdash P_2 : \tau_2 \rrbracket$, show that $\llbracket I; \Gamma \vdash P_1; P_2; \text{lam } x_2. \text{lam } x_1. \text{push } [x_1, x_2] : \tau_1 \times \tau_2 \rrbracket$.*

Proof. We are given $(k, \gamma) \in \mathcal{G}^\lambda[\Gamma]$, $((k, \emptyset), \emptyset, \gamma^{\text{I}^S}) \in \mathcal{G}^S[\text{I}^S]$, $((k, \emptyset), \emptyset, \gamma^{\text{I}^X}) \in \mathcal{G}^X[\text{I}^X]$, and need to show that $(k, \gamma^{\text{I}^X}(\gamma^{\text{I}^S}(\gamma(P_1; P_2; \text{lam } x_2. \text{lam } x_1. \text{push } [x_1, x_2])))) \in \mathcal{E}^\lambda[\tau_1 \times \tau_2]$.

Pushing the substitutions in and combining $\gamma^{\text{I}^X} \circ \gamma^{\text{I}^S} \circ \gamma$ (for compactness) to γ^{I} , we refine this slightly to:

$$(k, \gamma^{\text{I}}(P_1); \gamma^{\text{I}}(P_2); \text{lam } x_2. \text{lam } x_1. \text{push } [x_1, x_2]) \in \mathcal{E}^\lambda[\tau_1 \times \tau_2]$$

We apply Lemma D.0.2 twice, appealing to both induction hypotheses, to reduce our obligation to showing

$$(k', \text{push } v_1; \text{push } v_2; \text{lam } x_2. \text{lam } x_1. \text{push } [x_1, x_2]) \in \mathcal{E}^\lambda[\tau_1 \times \tau_2]$$

where (k', v_1) is in $\mathcal{V}^\lambda[\tau_1]$ and (k', v_2) is in $\mathcal{V}^\lambda[\tau_2]$. The term then takes three steps, resulting in the value $[v_1, v_2]$ on the stack, which suffices to finish the proof.

\square

Lemma D.0.13 (fst). *If $\llbracket I; \Gamma \vdash P : \tau_1 \times \tau_2 \rrbracket$, show that $\llbracket I; \Gamma \vdash P_1; \text{push } 0; \text{idx} : \tau_1 \rrbracket$.*

Proof. We are given $(k, \gamma) \in \mathcal{G}^\lambda[\Gamma]$, $((k, \emptyset), \emptyset, \gamma^{\text{I}^S}) \in \mathcal{G}^S[\text{I}^S]$, $((k, \emptyset), \emptyset, \gamma^{\text{I}^X}) \in \mathcal{G}^X[\text{I}^X]$, and need to show that $(k, \gamma^{\text{I}^X}(\gamma^{\text{I}^S}(\gamma(P_1; \text{push } 0; \text{idx})))) \in \mathcal{E}^\lambda[\tau_1]$.

Pushing the substitutions in and combining $\gamma^{\text{I}^X} \circ \gamma^{\text{I}^S} \circ \gamma$ (for compactness) to γ^{I} , we refine this slightly to:

$$(k, \gamma^I(P_1); \text{push } 0; \text{idx}) \in \mathcal{E}^\lambda[\tau_1]$$

We apply Lemma D.0.2 to reduce this to showing

$$(k', \text{push } v; \text{push } 0; \text{idx}) \in \mathcal{E}^\lambda[\tau_1]$$

where $(k', v) \in \mathcal{V}^\lambda[\tau_1 \times \tau_2]$, and thus has shape $[v_1, v_2]$. The term takes three steps to result in v_1 on top of the stack, which suffices to finish the proof.

□

Lemma D.0.14 (snd). *If $\llbracket I; \Gamma \vdash P : \tau_1 \times \tau_2 \rrbracket$, show that $\llbracket I; \Gamma \vdash P_1; \text{push } 1; \text{idx} : \tau_2 \rrbracket$.*

Proof. This proof is nearly identical to that of `fst`. □

Lemma D.0.15 (inl). *If $\llbracket I; \Gamma \vdash P : \tau_1 \rrbracket$, show that $\llbracket I; \Gamma \vdash P; \text{lam } x. \text{push } [0, x] : \tau_1 + \tau_2 \rrbracket$.*

Proof. We are given $(k, \gamma) \in \mathcal{G}^\lambda[\Gamma]$, $((k, \emptyset), \emptyset, \gamma^I)$ $\in \mathcal{G}^S[\mathbb{I}^S]$, $((k, \emptyset), \emptyset, \gamma^I)$ $\in \mathcal{G}^X[\mathbb{I}^X]$, and need to show that $(k, \gamma^I(\gamma^I(\gamma(P; \text{lam } x. \text{push } [0, x])))) \in \mathcal{E}^\lambda[\tau_1 + \tau_2]$.

Pushing the substitutions in and combining $\gamma^I \circ \gamma^I \circ \gamma$ (for compactness) to γ^I , we refine this slightly to:

$$(k, \gamma^I(P_1); \text{lam } x. \text{push } [0, x]) \in \mathcal{E}^\lambda[\tau_1 + \tau_2]$$

We apply Lemma D.0.2 to reduce this to

$$(k', \text{push } v_1; \text{lam } x. \text{push } [0, x]) \in \mathcal{E}^\lambda[\tau_1 + \tau_2]$$

where $(k', v_1) \in \mathcal{V}^\lambda[\tau_1]$. This takes three steps to result in $[0, v_1]$ on the stack, which suffices to complete the proof. □

Lemma D.0.16 (inr). *If $\llbracket I; \Gamma \vdash P : \tau_2 \rrbracket$, show that $\llbracket I; \Gamma \vdash P; \text{lam } x. \text{push } [1, x] : \tau_1 + \tau_2 \rrbracket$.*

Proof. This proof is nearly identical to that of `inl`. □

Lemma D.0.17 (match). *If $\llbracket I; \Gamma \vdash P : \tau_1 + \tau_2 \rrbracket$, $\llbracket I; \Gamma, x : \tau_1 \vdash P_1 : \tau_1 \rrbracket$, and $\llbracket I; \Gamma, y : \tau_2 \vdash P_2 : \tau_2 \rrbracket$, show that $\llbracket I; \Gamma \vdash P; \text{DUP}; \text{push } 1; \text{idx}; \text{SWAP}; \dots : \tau \rrbracket$.*

$\text{push } 0; \text{idx}; \text{if0 } (\text{lam } x. P_1) (\text{lam } y. P_2)$

Proof. We are given $(k, \gamma) \in \mathcal{G}^\lambda[\Gamma]$, $((k, \emptyset), \emptyset, \gamma^I)$ $\in \mathcal{G}^S[\mathbb{I}^S]$, $((k, \emptyset), \emptyset, \gamma^I)$ $\in \mathcal{G}^X[\mathbb{I}^X]$, and need to show that, after pushing substitutions and combining $\gamma^I \circ \gamma^I \circ \gamma$ (for compactness) to γ^I ,

$$(k, \gamma^I(P); \text{DUP}; \text{push } 1; \text{idx}; \text{SWAP}; \text{push } 0; \text{idx}; \text{if0 } (\text{lam } x. \gamma^I(P_1)) (\text{lam } y. \gamma^I(P_2))) \\ \in \mathcal{E}^\lambda[\tau]$$

We apply Lemma D.0.2 to reduce this to showing

$(k', v; \text{DUP}; \text{push } 1; \text{idx}; \text{SWAP}; \text{push } 0; \text{idx}; \text{if0 } (\text{lam } x. \gamma^I(P_1)) (\text{lam } y. \gamma^I(P_2))) \in \mathcal{E}^\lambda[\tau]$

where $(k', \text{push } v) \in \mathcal{V}^\lambda[\tau_1 + \tau_2]$. We appeal to Lemma D.0.2 again, noting that after seven steps we will either have a $v_1, 0$ where v_1 is in $\mathcal{V}^\lambda[\tau_1]$ or $v_2, 1$ where v_2 is in $\mathcal{V}^\lambda[\tau_2]$ on the top of the stack, and thus in either case, after two more steps we can appeal to one of our induction hypotheses to complete the proof.

□

Lemma D.0.18 (fold). *If $[\![\mathbf{I}; \Gamma \vdash P : \tau[\mu\alpha.\tau/\alpha]]\!]$, show that $[\![\mathbf{I}; \Gamma \vdash P : \mu\alpha.\tau]\!]$.*

Proof. We are given $(k, \gamma) \in \mathcal{G}^\lambda[\Gamma]$, $((k, \emptyset), \emptyset, \gamma^{I^S}) \in \mathcal{G}^S[\mathbf{I}^S]$, $((k, \emptyset), \emptyset, \gamma^{I^X}) \in \mathcal{G}^X[\mathbf{I}^X]$, and need to show that $(k, \gamma^{I^S}(\gamma^I(\gamma(P)))) \in \mathcal{E}^\lambda[\mu\alpha.\tau]$.

This means we need to pick an arbitrary heap H and stack S and show that this runs down to a value in the value relation (or else runs forever or to a well-defined error).

We can instantiate our hypothesis with the same substitutions, combining $\gamma^{I^X} \circ \gamma^{I^S} \circ \gamma$ (for compactness) to γ^I , and heap and stack. This means that (assuming no divergence beyond k , or error, which would finish the proof immediately):

$$\langle H ; S ; \gamma^I(P) \rangle \xrightarrow{j} \langle H ; S, v ; \cdot \rangle$$

Now, we know from the hypothesis that $(k - j, v) \in \mathcal{V}^\lambda[\tau[\mu\alpha.\tau/\alpha]]$. What we need to show is that $(k - j, v)$ is also in $\mathcal{V}^\lambda[\mu\alpha.\tau]$. But this is fine, since that definition only requires that the value be in $\mathcal{V}^\lambda[\tau[\mu\alpha.\tau/\alpha]]$ for smaller step index, and our relations are closed under smaller step index.

□

Lemma D.0.19 (unfold). *If $[\![\mathbf{I}; \Gamma \vdash P : \mu\alpha.\tau]\!]$, show that $[\![\mathbf{I}; \Gamma \vdash P; \text{noop} : \tau[\mu\alpha.\tau/\alpha]]\!]$.*

Proof. We are given $(k, \gamma) \in \mathcal{G}^\lambda[\Gamma]$, $((k, \emptyset), \emptyset, \gamma^{I^S}) \in \mathcal{G}^S[\mathbf{I}^S]$, $((k, \emptyset), \emptyset, \gamma^{I^X}) \in \mathcal{G}^X[\mathbf{I}^X]$, and need to show that $(k, \gamma^{I^S}(\gamma^I(\gamma(P; \text{noop})))) \in \mathcal{E}^\lambda[\tau[\mu\alpha.\tau/\alpha]]$.

This means we need to pick an arbitrary heap H and stack S and show that this runs down to a value in the value relation (or else runs forever or to a well-defined error).

We can instantiate our hypothesis with the same substitutions, combining $\gamma^{I^X} \circ \gamma^{I^S} \circ \gamma$ (for compactness) to γ^I , and heap and stack. This means

that (assuming no divergence beyond k , or error, which would finish the proof immediately):

$$\langle H ; S ; \gamma^I(P) \rangle \xrightarrow{j} \langle H ; S, v ; \cdot \rangle$$

Now, we return to our original program, which runs as:

$$\langle H ; S ; \gamma^I(P) ; \text{noop} \rangle \xrightarrow{j} \langle H ; S, v ; \text{noop} \rangle \rightarrow \langle H ; S, v ; \cdot \rangle$$

Now, we know from the hypothesis that $(k - j, v) \in \mathcal{V}^\lambda[\mu\alpha.\tau]$. What we need to show is that $(k - j - 1, v)$ (since we took one more step) is in $\mathcal{V}^\lambda[\tau[\mu\alpha.\tau/\alpha]]$. But, the definition of $\mathcal{V}^\lambda[\mu\alpha.\tau]$ gives us this immediately, as our step index is lower. \square

Lemma D.0.20 (fun). *If $\llbracket I; \Gamma, f : (\tau_1, \dots, \tau_n) \rightarrow \tau' ; x_i : \tau_i \vdash P : \tau' \rrbracket$, show that $\llbracket I; \Gamma \vdash \text{push}(\text{thunk push}(\text{thunk lam } f.\text{lam } x_n. \dots . \text{lam } x_1.P); \text{fix}) : (\tau_1, \dots, \tau_n) \rightarrow \tau' \rrbracket$*

Proof. We are given $(k, \gamma) \in \mathcal{G}^\lambda[\Gamma]$, $((k, \emptyset), \emptyset, \gamma^{IS}) \in \mathcal{G}^S[\text{IS}]$, $((k, \emptyset), \emptyset, \gamma^{IX}) \in \mathcal{G}^X[\text{IX}]$, and need to show, after pushing in substitutions and combining $\gamma^{IX} \circ \gamma^{IS} \circ \gamma$ (for compactness) to γ^I :

$$(k, \text{push}(\text{thunk push}(\text{thunk lam } f.\text{lam } x_n. \dots . \text{lam } x_1.\gamma^I(P)); \text{fix})) \\ \in \mathcal{E}^\lambda[(\tau_1, \dots, \tau_n) \rightarrow \tau']$$

Following the definition of $\mathcal{E}^\lambda[\tau]$, we choose an arbitrary H and S and run the term, which after one step, results in the thunk on the stack. That means what we need to show is:

$$(k, \text{thunk push}(\text{thunk lam } f.\text{lam } x_n. \dots . \text{lam } x_1.\gamma^I(P)); \text{fix}) \in \mathcal{V}^\lambda[(\tau_1, \dots, \tau_n) \rightarrow \tau']$$

Syntactically, this clearly satisfies the value relation; that means what we need to show is:

$$\begin{aligned} & \forall v_i \ k' < k. \wedge \ (k', v_i) \in \mathcal{V}^\lambda[\tau_i] \\ & \implies (k', [x_1 \mapsto v_1, \dots, x_n \mapsto v_n, \\ & \quad f \mapsto (\text{thunk push}(\text{thunk lam } f.\text{lam } x_n. \dots . \text{lam } x_1.\gamma^I(P)); \text{fix})]P) \in \mathcal{E}^\lambda[\tau'] \end{aligned}$$

We do this by appeal to our hypothesis. Specifically, we construct an extended substitution γ' :

$$\gamma^I, x_1:v_1, \dots, x_n:v_n, f : (\text{thunk push}(\text{thunk lam } f.\text{lam } x_n. \dots . \text{lam } x_1.\gamma^I(P)); \text{fix})$$

Note that f can be given the needed type in the relation because we are only considering step $k' < k$, and our overall induction is over step indices. Further, our relations are closed under step indices, which means our substitution γ^I is still valid when restricted to k' . This means that we know:

$$(k', \gamma'(P)) \in \mathcal{E}^\lambda[\tau']$$

Which, expanding out γ' , is exactly what we needed to show. \square

Lemma D.0.21 (app). *If $\llbracket I; \Gamma \vdash P : (\tau_1, \dots, \tau_n) \rightarrow \tau' \rrbracket$ and for $i \in \{1, \dots, n\} \llbracket I; \Gamma \vdash P_i : \tau_i \rrbracket$ then*

$$\llbracket I; \Gamma \vdash P; P_1; \text{SWAP} \dots P_n; \text{SWAP}; \text{call} : \tau' \rrbracket$$

Proof. We are given $(k, \gamma) \in \mathcal{G}^\lambda[\Gamma]$, $((k, \emptyset), \emptyset, \gamma^b I^S) \in \mathcal{G}^S[I^S]$, $((k, \emptyset), \emptyset, \gamma^I X) \in \mathcal{G}^X[I^X]$, and need to show that $(k, \gamma^I X(\gamma(P; P_1; \text{SWAP} \dots P_n; \text{SWAP}; \text{call}))) \in \mathcal{E}^\lambda[\tau']$. Pushing the substitutions in and combining $\gamma^I X \circ \gamma^I S \circ \gamma$ (for compactness) to γ^I , we refine this slightly to:

$$(k, \gamma^I(P); \gamma^I(P_1); \text{SWAP} \dots \gamma^I(P_n); \text{SWAP}; \text{call}) \in \mathcal{E}^\lambda[\tau']$$

Following the definition of $\mathcal{E}^\lambda[\tau']$, we choose an arbitrary H and S and run the term. To figure out how it steps, we instantiate our first hypothesis with γ^I , H , and S . This tells us that either P runs forever (in which case, the term is in the relation trivially), or:

$$\langle H ; S ; \gamma^I(P) \rangle \xrightarrow{j} \langle H' ; S' ; \cdot \rangle$$

And either S' is a well-defined error (in which case, the entire program would have run to the same error, and we are again done), or S, v_f with $(k - j, v_f) \in \mathcal{V}^\lambda[(\tau_1, \dots, \tau_n) \rightarrow \tau']$.

Then, we instantiate the second hypothesis with γ^I , H' , and S , resulting in a similar result for a smaller step index k_1 and H_1 and value v_1 . We can repeat this process another $n - 1$ times. This results in an overall evaluation of:

$$\begin{aligned} & \langle H ; S ; \gamma^I(P); \gamma^I(P_1); \text{SWAP} \dots \gamma^I(P_n); \text{SWAP}; \text{call} \rangle \\ & \xrightarrow{*} \langle H' ; S, v_f ; \gamma^I(P_1); \text{SWAP} \dots \gamma^I(P_n); \text{SWAP}; \text{call} \rangle \\ & \xrightarrow{*} \langle H_1 ; S, v_f, v_1 ; \text{SWAP} \dots \gamma^I(P_n); \text{SWAP}; \text{call} \rangle \\ & \xrightarrow{*} \langle H_1 ; S, v_1, v_f ; \dots \gamma^I(P_n); \text{SWAP}; \text{call} \rangle \\ & \dots \\ & \xrightarrow{*} \langle H_n ; S, v_1, v_2, \dots, v_n, v_f ; \text{call} \rangle \end{aligned}$$

From $\mathcal{V}^\lambda \llbracket (\tau_1, \dots, \tau_n) \rightarrow \tau' \rrbracket$, we know the shape of v_f , so we can expand that out and step further:

$$\begin{aligned} & \langle H_n ; S, v_1, v_2, \dots, v_n, (\text{thunk push (thunk lam f.lam } x_n. \dots. \text{ lam } x_1.P); \text{fix}) ; \text{call} \rangle \\ & \rightarrow \langle H_n ; S, v_1, v_2, \dots, v_n ; \text{push (thunk lam f.lam } x_n. \dots. \text{ lam } x_1.P); \text{fix} \rangle \\ & \rightarrow \langle H_n ; S, v_1, v_2, \dots, v_n, \text{thunk lam f.lam } x_n. \dots. \text{ lam } x_1.P ; \text{fix} \rangle \\ & \rightarrow \langle H_n ; S, v_1, v_2, \dots, v_n, \text{thunk(push(thunk lam f.lam } x_n. \dots. \text{ lam } x_1.P); \text{fix}) ; \\ & \quad \text{lam f.lam } x_n. \dots. \text{ lam } x_1.P \rangle \\ & \xrightarrow{n+1} \langle H_n ; S ; [x_i \mapsto v_i, f \mapsto \text{thunk(push(thunk lam f.lam } x_n. \dots. \text{ lam } x_1.P); \text{fix})]P \rangle \end{aligned}$$

Now we can appeal to the definition of $\mathcal{V}^\lambda \llbracket (\tau_1, \dots, \tau_n) \rightarrow \tau' \rrbracket$, which tells us that this term is in $\mathcal{E}^\lambda \llbracket \tau' \rrbracket$, which is exactly what we need to complete the proof: we can instantiate that relation with H_n , S , and compose the two reductions together to produce the result needed. \square

Lemma D.0.22 (boundary S). $\llbracket I^S \uplus \Gamma \vdash_S P : \tau \rrbracket \implies \llbracket I ; \Gamma \vdash P ; \langle \downarrow \tau \rangle : \downarrow \tau \rrbracket$

Proof. Expanding the goal, we see we need to show:

$$\forall k \gamma. \forall ((k, \emptyset), \emptyset, \gamma^{I^S}) \in \mathcal{G}^S \llbracket I^S \rrbracket. \forall ((k, \emptyset), \emptyset, \gamma^{I^X}) \in \mathcal{G}^X \llbracket I^X \rrbracket. (k, \gamma) \in \mathcal{G}^\lambda \llbracket \Gamma \rrbracket \implies (k, \gamma^{I^X}(\gamma^{I^S}(\gamma(P; \langle \downarrow \tau \rangle)))) \in \mathcal{E}^\lambda \llbracket \downarrow \tau \rrbracket$$

From Lemma 9.5.2, we know $\langle \downarrow \tau \rangle$ is closed, so we can push the substitutions in to just over P . Further, from the hypothesis, we know that P has no free variables from I^X , so we can eliminate that substitution.

The hypothesis that we are working with says:

$$\forall W \varphi \gamma' (W, \varphi, \gamma') \in \mathcal{G}^S \llbracket I^S \uplus \Gamma \rrbracket \wedge \varphi = \text{flocs}(\gamma(P)) \implies (k, \varphi, \gamma'(P)) \in \mathcal{E}^S \llbracket \tau \rrbracket$$

To instantiate the hypothesis, we need an environment γ' that satisfies $\mathcal{G}^S \llbracket I^S \uplus \Gamma \rrbracket$. We argue that it is exactly γ composed with γ^{I^S} : we know they are disjoint, and we know the former can be lifted into the latter via Lemma 9.5.1. This means, in particular, that φ is \emptyset .

Since we have no relevant locations, any heap will satisfy the expression relation: in particular, the arbitrary H that we have to consider for our obligation, and we can similarly use the arbitrary stack S . This means that our hypothesis tells us that:

$$\langle H ; S ; (\gamma^{I^S}(\gamma(P))) \rangle \xrightarrow{*} \langle H' ; S' ; \cdot \rangle$$

Unless we run beyond our step budget, in which case we are trivially in the relation. Similarly, if we run to Fail c , we are also in our relation.

Otherwise, we know that $S' = S, v$ and, for a future world $W' \sqsubseteq W$ that H' satisfies with the relevant locations φ' , $(W', \varphi', v) \in \mathcal{V}^S[\tau]$.

Now, what we want to show is that this value is “contained” by the code in $\langle \downarrow \tau \rangle$ to behave like $\downarrow \tau$. But, clearly we can’t show that using the $\mathcal{E}^\lambda[\tau]$ logical relation, as the value still can have locations it is closing over, etc. So, we proceed by two steps. First, we appeal to Lemma 9.5.3

This will tell us that we can evaluate the whole program at question further, to get to a point with a world $W'' \sqsubseteq W'$, $\varphi'', H'' :_{\varphi'' \cup \varphi'} W''$ and $(W'', \varphi'', v') \in \mathcal{V}^S[\uparrow \downarrow \tau]$:

$$\begin{aligned} & \langle H ; S ; (\gamma^I(\gamma(P)); \langle \downarrow \tau \rangle) \rangle \xrightarrow{*} \\ & \langle H' ; S, v ; \langle \downarrow \tau \rangle \rangle \xrightarrow{*} \\ & \langle H'' ; S, v' ; \cdot \rangle \end{aligned}$$

Now, we appeal to Lemma 9.5.1

This means that the value that we ran down to is in $(W''.k, v') \in \mathcal{V}^\lambda[\downarrow \tau]$, which is exactly what we need to show.

□

D.0.3 FunLang with S Compatibility Lemmas

$$[\Gamma \vdash_S P : \tau] \equiv \forall W \varphi \gamma. (W, \varphi, \gamma) \in \mathcal{G}^S[\Gamma] \implies (W, \text{flocs}(\gamma(P)), \gamma(P)) \in \mathcal{E}^S[\tau]$$

Lemma D.0.23 (unit). *Show that $[\Gamma \vdash_S \text{push } 0 : \text{unit}]$.*

Proof. We expand the goal, pushing out substitution through and simplifying φ , given there are no free variables in push 0, to get an obligation:

$$(W, \emptyset, \text{push } 0) \in \mathcal{E}^S[\text{unit}]$$

To satisfy this, we note that we can take 1 step (if $W.k = 1$, we are in the relation trivially) to having 0 on top of the stack, with a world that has the same heap typing and, still, no relevant locations, thus satisfying $\mathcal{V}^S[\text{unit}]$, as needed. □

Lemma D.0.24 (bool). *Show for any n, $[\Gamma \vdash_S n : \text{bool}]$.*

Proof. This proof is identical to that of `unit`. □

Lemma D.0.25 (if). *If $[\Gamma \vdash_S P : \text{bool}]$, $[\Gamma \vdash_S P_1 : \tau]$, and $[\Gamma \vdash_S P_2 : \tau]$ then*

$$[\Gamma \vdash_S P; \text{if0 } P_1 \ P_2 : \tau].$$

Proof. We are given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^S[\gamma]$, where $\varphi = \text{flocs}(\gamma(P; \text{if0 } P_1 \ P_2))$, and need to show that $(W, \varphi, \gamma(P; \text{if0 } P_1 \ P_2)) \in \mathcal{E}^S[\tau]$. Pushing the substitutions in, we refine this slightly to:

$$(W, \varphi, \gamma(P); \text{if0 } \gamma(P_1) \ \gamma(P_2))) \in \mathcal{E}^S[\tau]$$

We appeal to Lemma D.0.3, which reduces our obligation to

$$(W', \varphi', v; \text{if0 } \gamma(P_1) \ \gamma(P_2))) \in \mathcal{E}^S[\tau]$$

where from our induction hypothesis we know that for $H :_\varphi W$ and arbitrary S , $\langle H ; S ; \gamma(P) \rangle \xrightarrow{*} \langle H^1 ; S^1 ; \cdot \rangle$ and either S^1 is a dynamic failure, in which case we are done, or it is v above, where for some $W' \sqsubseteq W$, φ' , $H^1 :_{\varphi \cup \varphi'} W'$.

From the definition of $\mathcal{V}^S[\text{bool}]$, we know v is either 0 or non-zero. In either case, we appeal to Lemma D.0.3 again, relying on the corresponding hypotheses in the corresponding case that the term reduces to. \square

Lemma D.0.26 (int). *For any n , show $\llbracket \Gamma \vdash_S \text{push } n : \text{int} \rrbracket$.*

Proof. This proof is essentially equivalent to that of `unit` and `bool`. \square

Lemma D.0.27 (op- $=$). *If $\llbracket \Gamma \vdash_S P_1 : \text{int} \rrbracket$ and $\llbracket \Gamma \vdash_S P_2 : \text{int} \rrbracket$, show that $\llbracket \Gamma \vdash_S P_1; P_2; \text{equal?} : \text{bool} \rrbracket$.*

Proof. We are given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^S[\gamma]$, where $\varphi = \text{flocs}(\gamma(P_1; P_2; \text{equal?}))$, and need to show that $(W, \varphi, \gamma(P_1; P_2; \text{equal?})) \in \mathcal{E}^S[\text{bool}]$. Pushing the substitutions in, we refine this slightly to:

$$(W, \varphi, \gamma(P_1); \gamma(P_2) \ \text{equal?}) \in \mathcal{E}^S[\text{bool}]$$

We then apply Lemma D.0.3 twice, relying on our two hypotheses to reduce our obligation to

$$(W', \varphi', \text{push } v_1; \text{push } v_2; \text{equal?}) \in \mathcal{E}^S[\text{bool}]$$

Note that we instantiate the second hypothesis with $\varphi'' = \text{flocs}(\gamma(P_2)) \subset \varphi$, noting that $H^1 :_{\varphi''} W^1$ via Lemma D.0.1.

Since v_1 and v_2 are integers, this takes three steps to either 0 or 1 on top of the stack (with unchanged heap), which is sufficient to complete the proof. \square

Lemma D.0.28 (op- $<$). *If $\llbracket \Gamma \vdash_S P_1 : \text{int} \rrbracket$ and $\llbracket \Gamma \vdash_S P_2 : \text{int} \rrbracket$, show that $\llbracket \Gamma \vdash_S P_1; P_2; \text{less?} : \text{bool} \rrbracket$.*

Proof. This proof is identical to that of `=`. \square

Lemma D.0.29 (op- $+$). *If $\llbracket \Gamma \vdash_S P_1 : \text{int} \rrbracket$ and $\llbracket \Gamma \vdash_S P_2 : \text{int} \rrbracket$, show that $\llbracket \Gamma \vdash_S P_1; P_2; \text{add} : \text{int} \rrbracket$.*

Proof. This proof is identical to that of $=$. \square

Lemma D.0.30 (var). *For any $x : \tau \in \Gamma$, show that $[\![\Gamma \vdash_S \text{push } x : \tau]\!]$.*

Proof. We are given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^S[\![\gamma]\!]$, where $\varphi = \text{flocs}(\gamma(\text{push } x))$, and need to show that $(W, \varphi, \gamma(\text{push } x)) \in \mathcal{E}^S[\![\tau]\!]$.

Based on the definition of $\mathcal{G}^S[\![\gamma]\!]$, we know that $\gamma(x) = v$ for some v where $(W, \varphi', v) \in \mathcal{V}^S[\![\tau]\!]$ and $\varphi' \subset \varphi$. Substituting, we can refine our proof goal to:

$$(W, \varphi, \text{push } v) \in \mathcal{E}^S[\![\tau]\!]$$

And since $\varphi = \text{flocs}(v)$, we know $\varphi' = \varphi$. This means that after one step starting from a heap satisfying W and φ' , we terminate with v on the top of the stack, and we are done. \square

Lemma D.0.31 (pair). *If $[\![\Gamma \vdash_S P_1 : \tau_1]\!]$ and $[\![\Gamma \vdash_S P_2 : \tau_2]\!]$ then $[\![\Gamma \vdash_S P_1; P_2; \text{lam } x_2.\text{lam } x_1.\text{push } [x_1, x_2] : \tau_1 \times \tau_2]\!]$*

Proof. We are given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^S[\![\gamma]\!]$, where $\varphi = \text{flocs}(\gamma(P_1; P_2; \text{lam } x_2.\text{lam } x_1.\text{push } [x_1, x_2]))$, and need to show that $(W, \varphi, \gamma(P_1; P_2; \text{lam } x_2.\text{lam } x_1.\text{push } [x_1, x_2])) \in \mathcal{E}^S[\![\tau_1 \times \tau_2]\!]$.

Pushing the substitutions in, we refine this slightly to:

$$(W, \varphi, \gamma(P_1); \gamma(P_2); \text{lam } x_2.\text{lam } x_1.\text{push } [x_1, x_2]) \in \mathcal{E}^S[\![\tau_1 \times \tau_2]\!]$$

This follows from two applications of Lemma D.0.3 and the operational semantics, relying on Lemma D.0.1 for the choice of relevant locations. \square

Lemma D.0.32 (fst). *If $[\![\Gamma \vdash_S P : \tau_1 \times \tau_2]\!]$, show that $[\![\Gamma \vdash_S P_1; \text{push } 0; \text{idx} : \tau_1]\!]$.*

Proof. We are given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^S[\![\gamma]\!]$, where $\varphi = \text{flocs}(\gamma(P_1; \text{push } 0; \text{idx}))$, and need to show that $(W, \varphi, \gamma(P_1; \text{push } 0; \text{idx})) \in \mathcal{E}^S[\![\tau_1]\!]$.

Pushing the substitutions in, we refine this slightly to:

$$(W, \varphi, \gamma(P_1); \text{push } 0; \text{idx}) \in \mathcal{E}^S[\![\tau_1]\!]$$

We apply Lemma D.0.3, which, combined with the hypothesis, the operational semantics, and definition of the value relation is sufficient to complete the proof. \square

Lemma D.0.33 (snd). *If $[\![\Gamma \vdash_S P : \tau_1 \times \tau_2]\!]$, show that $[\![\Gamma \vdash_S P_1; \text{push } 1; \text{idx} : \tau_2]\!]$.*

Proof. This proof is identical to `fst`. \square

Lemma D.0.34 (inl). *If $[\![\Gamma \vdash_S P : \tau_1]\!]$, show that $[\![\Gamma \vdash_S P; \text{lam } x.\text{push } [0, x] : \tau_1 + \tau_2]\!]$.*

Proof. We are given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^S[\gamma]$, where $\varphi = \text{flocs}(\gamma(P; \text{lam } x.\text{push } [0, x]))$, and need to show that $(W, \varphi, \gamma(P_1; \text{lam } x.\text{push } [0, x])) \in \mathcal{E}^S[\tau_1 + \tau_2]$.

Pushing the substitutions in, we refine this slightly to:

$$(W, \varphi, \gamma(P); \text{lam } x.\text{push } [0, x]) \in \mathcal{E}^S[\tau_1 + \tau_2]$$

As in other cases, this follows from Lemma D.0.3 and our hypothesis. \square

Lemma D.0.35 (inr). *If $[\Gamma \vdash_S P : \tau_2]$, show that $[\Gamma \vdash_S P; \text{lam } x.\text{push } [1, x] : \tau_1 + \tau_2]$.*

Proof. This proof is identical to that of `inl`. \square

Lemma D.0.36 (match). *If $[\Gamma \vdash_S P : \tau_1 + \tau_2]$, $[\Gamma, x : \tau_1 \vdash_S P_1 : \tau]$, and $[\Gamma, y : \tau_2 \vdash_S P_2 : \tau]$, show that $[\Gamma \vdash_S P; \text{DUP}; \text{push } 1; \text{idx}; \text{SWAP}; \text{push } 0; \text{idx}; \text{if0 } (\text{lam } x.P_1) (\text{lam } y.P_2) : \tau]$.*

Proof. We are given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^S[\gamma]$, where

$$\varphi = \text{flocs}(\gamma(P; \text{DUP}; \text{push } 1; \text{idx}; \text{SWAP}; \text{push } 0; \text{idx}; \text{if0 } (\text{lam } x.P_1) (\text{lam } y.P_2)))$$

and need to show that, after pushing in substitutions: Pushing the substitutions in, we refine this slightly to:

$$(W, \varphi, \gamma(P); \text{DUP}; \text{push } 1; \text{idx}; \text{SWAP}; \text{push } 0; \text{idx}; \text{if0 } (\text{lam } x.\gamma(P_1) (\text{lam } y.\gamma(P_2)))) \in \mathcal{E}^S[\tau_1 + \tau_2]$$

We appeal to Lemma D.0.3 and the operational semantics to reduce this to considering the two possible branches: when $\mathcal{V}^S[\tau_1 + \tau_1]$ is $[0, v]$ and when it is $[1, v]$. In both cases, we again appeal to Lemma D.0.3, but to the second or third hypothesis respectively, as operationally that is what we will reduce to, with appropriate substitution. \square

Lemma D.0.37 (fold). *If $[\Gamma \vdash_S P : \tau[\mu\alpha.\tau/\alpha]]$, show that $[\Gamma \vdash_S P : \mu\alpha.\tau]$.*

Proof. We are given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^S[\gamma]$, where $\varphi = \text{flocs}(\gamma(P))$, and need to show that $(W, \varphi, \gamma(P)) \in \mathcal{E}^S[\mu\alpha.\tau]$.

This means we are given an heap $H :_\varphi W$, stack γ , and, assuming we don't run forever or out of steps (in $W.k$ budget), we run down to $\langle H'; \gamma'; \cdot \rangle$. We instantiate our first hypothesis with W , H , γ , and φ , to get that:

$$\langle H ; S ; \gamma(P) \rangle \xrightarrow{j^1} \langle H^1 ; S^1 ; \cdot \rangle$$

Now, either S^1 is `Fail c` for appropriate c , in which case the entire program will be and we are done, or $S^1 = S, v$ and for $W^1 \sqsubseteq W$, $H^1 :_{\varphi^1 \cup \varphi} W^1$, and $(W^1, \varphi^1, v) \in \mathcal{V}^S[\tau[\mu\alpha.\tau/\alpha]]$. Now, our obligation only needs us to prove

that the resulting value, which is the same value, is in this relation at lower step index, so we are done. \square

Lemma D.0.38 (unfold). *If $\llbracket \Gamma \vdash_S P : \mu\alpha.\tau \rrbracket$, show that $\llbracket \Gamma \vdash_S P; \text{noop} : \tau[\mu\alpha.\tau] \rrbracket$.*

Proof. We are given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^S[\gamma]$, where $\varphi = \text{flocs}(\gamma(P; \text{noop}))$, and need to show that $(W, \varphi, \gamma(P; \text{noop})) \in \mathcal{E}^S[\tau[\mu\alpha.\tau]]$.

This means we are given an heap $H :_\varphi W$, stack γ , and, assuming we don't run forever or out of steps (in $W.k$ budget), we run down to $\langle H'; \gamma'; \cdot \rangle$. We instantiate our first hypothesis with W , H , γ , and φ , to get that:

$$\langle H ; S ; \gamma(P) \rangle \xrightarrow{j^1} \langle H^1 ; S^1 ; \cdot \rangle$$

Now, either S^1 is Fail c for appropriate c , in which case the entire program will be and we are done, or $S^1 = S, v$ and for $W^1 \sqsubseteq W$, $H^1 :_{\varphi^1 \cup \varphi} W^1$, and $(W^1, \varphi^1, v) \in \mathcal{V}^S[\mu\alpha.\tau]$. Now, our original term steps as follows:

$$\begin{aligned} & \langle H ; S ; \gamma(P; \text{noop}) \rangle \xrightarrow{j^1} \\ & \langle H^1 ; S, v ; \text{noop} \rangle \rightarrow \\ & \langle H^1 ; S, v ; \cdot \rangle \end{aligned}$$

We need to fulfill $\mathcal{E}^S[\tau[\mu\alpha.\tau]]$, which means we need to choose $W' \sqsubseteq W$, φ' such that $H^1 :_{\varphi'} W'$ and $(W', \varphi', v) \in \mathcal{V}^S[\tau[\mu\alpha.\tau]]$. We choose W' to be W^1 with the step index decreased by one. Because this is a strictly future world of W^1 , this follows directly from the definition of $\mathcal{V}^S[\mu\alpha.\tau]$. \square

Lemma D.0.39 (fun). *If $\llbracket \Gamma, f : (\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau', x_i : \tau_i \vdash_S P : \tau' \rrbracket$, show that $\llbracket \Gamma \vdash_S \text{push}(\text{thunk push}(\text{thunk lam } f.\text{lam } x_n. \dots. \text{ lam } x_1.\gamma(P)); \text{fix}) : (\tau_1, \dots, \tau_n) \xrightarrow{\bullet} \tau' \rrbracket$*

Proof. We are given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^S[\gamma]$, where

$$\varphi = \text{flocs}(\gamma(\text{push}(\text{thunk push}(\text{thunk lam } f.\text{lam } x_n. \dots. \text{ lam } x_1.P); \text{fix})))$$

and need to show that

$$(W, \varphi, \gamma(\text{push}(\text{thunk push}(\text{thunk lam } f.\text{lam } x_n. \dots. \text{ lam } x_1.P); \text{fix}))) \in \mathcal{E}^S[(\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau']$$

We can then push the substitution in to refine that to:

$$(W, \varphi, \text{push}(\text{thunk push}(\text{thunk lam } f.\text{lam } x_n. \dots. \text{ lam } x_1.\gamma(P); \text{fix})) \in \mathcal{E}^S[(\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau']$$

This means we are given a $H :_{\varphi} W$, stack S , and, assuming we don't run forever or out of steps (in $W.k$ budget), we run down to $\langle H' ; S' ; \cdot \rangle$.

This clearly takes a single step to put thunk push (thunk lam f.lam $x_1 \dots$ lam $x_n.\gamma(P)$; fix) on the stack. We can thus choose W' to be W with k decreased by 1, the same relevant location set \emptyset . Thus we need to satisfy the value relation, which amounts to:

$$\begin{aligned} & \forall v_i \varphi_i W' \sqsupseteq W. \\ & \varphi_i \subset \text{dom}(W'.\Psi) \wedge (W', \varphi_i, v_i) \in \mathcal{V}^S[\tau_i] \\ & \implies (W', \varphi \cup \bigcup_i \varphi_i, [x_1 \mapsto v_1, \dots, x_n \mapsto v_n, \\ & \quad f \mapsto (\text{thunk push (thunk lam f.lam } x_1 \dots \text{ lam } x_n.\gamma(P)); \text{fix})]) \in \mathcal{E}^S[\tau'] \end{aligned}$$

Thus we choose an arbitrary future world $W'' \sqsubset W'$, and construct an extended substitution $\gamma' = \gamma, x_1 \mapsto v_1, \dots, x_n \mapsto v_n, f \mapsto (\text{thunk...})$. We argue that $(W'', \varphi \cup \bigcup_i \varphi_i, \gamma') \in \mathcal{G}^S[\Gamma, f : (\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau', x_i : \tau_i]$. Clearly, all of the values v_i are in the value relation at the correct type. And, since W'' is a strict world extension, it has a smaller step index, which means that we can appeal to our inductive hypothesis to get that our function has the correct semantic type at that world.

That means we can instantiate our first hypothesis with W'' , $\varphi \cup \bigcup_i \varphi_i$ and γ' to complete the proof. \square

Lemma D.0.40 (app pure). *If $[\Gamma \vdash_S P : (\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau']$ and for $i \in \{1, \dots, n\} [\Gamma \vdash_S P_i : \tau_i]$ then*

$$[\Gamma \vdash_S P; P_1; \text{SWAP} \dots P_n; \text{SWAP}; \text{call} : \tau']$$

Proof. We are given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^S[\gamma]$, where

$$\varphi = \text{flocs}(\gamma(P; P_1; \text{SWAP} \dots P_n; \text{SWAP}; \text{call}))$$

and need to show that

$$(W, \varphi, \gamma(P; P_1; \text{SWAP} \dots P_n; \text{SWAP}; \text{call})) \in \mathcal{E}^S[\tau']$$

We can then push the substitution in to refine that to:

$$(W, \varphi, P; P_1; \text{SWAP} \dots P_n; \text{SWAP}; \text{call} \in \mathcal{E}^S[\tau'])$$

This means we are given a $H :_{\varphi} W$, stack S , and, assuming we don't run forever or out of steps (in $W.k$ budget), we run down to $\langle H' ; S' ; \cdot \rangle$.

To figure out how it steps, we instantiate our first hypothesis with W , γ , H , S and φ' , where $\varphi' = \text{flocs}(\gamma(P_1)) \subset \varphi$, noting that the heap will still satisfy the same world with the smaller φ' , to get that:

$$\langle H ; S ; \gamma(P) \rangle \xrightarrow{j^1} \langle H^1 ; S^1 ; \cdot \rangle$$

Now, either S^1 is $\text{Fail } c$ for appropriate c , in which case the entire program will be and we are done, or $S^1 = S, v_f$ and for $W^1 \sqsubseteq W$, $H^1 :_{\varphi^f \cup \varphi'} W^1$, and $(W^1, \varphi^f, v_f) \in \mathcal{V}^S \llbracket (\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau' \rrbracket$. From the value relation, we note that φ^f is \emptyset .

We then instantiate our second hypothesis with W^1, H^1, S^1 , and $\varphi'' = \text{flocs}(\gamma(P_2)) \subset \varphi$. Note that $H^1 :_{\varphi''} W^1$ from Lemma D.0.1.

This means that:

$$\langle H^1 ; S, v_f ; \gamma(P_2) \rangle \xrightarrow{j^2} \langle H^2 ; S^2 ; \cdot \rangle$$

Since this program began running in the same state as the previous one stopped, and the previous one began at the beginning of our whole program, again, we are either trivially in the relation or else we know that $S^2 = S^1, v_1 = S, v_f, v_1$ and for $W^2 \sqsubseteq W^1$, $H^2 :_{\varphi^1 \cup \varphi''} W^2$, and $(W^2, \varphi^1, v_1) \in \mathcal{V}^S \llbracket \tau_1 \rrbracket$.

We can repeat this process another $n - 1$ times. This results in an overall evaluation of:

$$\begin{aligned} & \langle H ; S ; \gamma(P); \gamma(P_1); \text{SWAP} \dots \gamma(P_n); \text{SWAP}; \text{call} \rangle \\ & \xrightarrow{*} \langle H' ; S, v_f ; \gamma(P_1); \text{SWAP} \dots \gamma(P_n); \text{SWAP}; \text{call} \rangle \\ & \xrightarrow{*} \langle H_1 ; S, v_f, v_1 ; \text{SWAP} \dots \gamma(P_n); \text{SWAP}; \text{call} \rangle \\ & \xrightarrow{*} \langle H_1 ; S, v_1, v_f ; \dots \gamma(P_n); \text{SWAP}; \text{call} \rangle \\ & \dots \\ & \xrightarrow{*} \langle H_n ; S, v_1, v_2, \dots, v_n, v_f ; \text{call} \rangle \end{aligned}$$

From $\mathcal{V}^S \llbracket (\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau' \rrbracket$, we know the shape of v_f , so we can expand that out and step further:

$$\begin{aligned} & \langle H_n ; S, v_1, v_2, \dots, v_n, (\text{thunk push} (\text{thunk lam } f.\text{lam } x_n \dots \text{ lam } x_1.P); \text{fix}) ; \text{call} \rangle \\ & \rightarrow \langle H_n ; S, v_1, v_2, \dots, v_n ; \text{push} (\text{thunk lam } f.\text{lam } x_n \dots \text{ lam } x_1.P); \text{fix} \rangle \\ & \rightarrow \langle H_n ; S, v_1, v_2, \dots, v_n, \text{thunk lam } f.\text{lam } x_n \dots \text{ lam } x_1.P ; \text{fix} \rangle \\ & \rightarrow \langle H_n ; S, v_1, v_2, \dots, v_n, \text{thunk}(\text{push}(\text{thunk lam } f.\text{lam } x_n \dots \text{ lam } x_1.P); \text{fix}) ; \\ & \quad \text{lam } f.\text{lam } x_n \dots \text{ lam } x_1.P \rangle \\ & \xrightarrow{n+1} \langle H_n ; S ; [x_i \mapsto v_i, f \mapsto \text{thunk}(\text{push}(\text{thunk lam } f.\text{lam } x_n \dots \text{ lam } x_1.P); \text{fix})]P \rangle \end{aligned}$$

Now we can appeal to the definition of $\mathcal{V}^S \llbracket (\tau_1, \dots, \tau_n) \xrightarrow{\circ} \tau' \rrbracket$, which tells us that this term is in $\mathcal{E}^S \llbracket \tau' \rrbracket$, given the values were in the value relation, which we know from each instantiated hypothesis. We then instantiate that relation with $W^{n+1}, \varphi^f \cup \bigcup_i \varphi^i$, and compose the reductions together to produce the result needed. \square

Lemma D.0.41 (app state). *If $\llbracket \Gamma \vdash_S P : (\tau_1, \dots, \tau_n) \xrightarrow{*} \tau' \rrbracket$ and for $i \in \{1, \dots, n\}$ $\llbracket \Gamma \vdash_S P_i : \tau_i \rrbracket$ then $\llbracket \Gamma \vdash_S P; P_1; \text{SWAP} \dots P_n; \text{SWAP}; \text{call} : \tau' \rrbracket$*

Proof. This proof is nearly identical to the previous one: the only difference is that φ^f is not empty, but that just carries down to the final instantiation which we left unsimplified in the above proof for clarity. \square

E

BEHAVIOR INTEROPERABILITY: EXCEPTIONS

E.0.1 Supporting Lemmas

Lemma E.0.1 ($\mathcal{E}^{\text{X}}[\tau]$ Embeds $\mathcal{V}^{\text{X}}[\tau]$). *If $(W, \varphi, v) \in \mathcal{V}^{\text{X}}[\tau]$, then $(W, \varphi, \text{push } v) \in \mathcal{E}^{\text{X}}[\tau]$.*

Proof. We choose heap $H :_{\varphi} W$, arbitrary stack S , take a single step and the result is immediate. \square

Lemma E.0.2 ($\mathcal{E}^{\text{X}}[\tau] \triangleleft$ Embeds $\mathcal{E}^{\text{X}}[\tau]$). *If $(W, \varphi, P) \in \mathcal{E}^{\text{X}}[\tau]$, then $(W, \varphi, P) \in \mathcal{E}^{\text{X}}[\tau] \triangleleft$.*

Proof. Our obligation is to show that for arbitrary τ', K , where $(W, \varphi^k, K) \in \mathcal{K}[\tau \Rightarrow \tau']$, $(W, \varphi \cup \varphi^k, K[P]) \in \mathcal{E}^{\text{X}}[\tau]$. We do this by appealing to our hypothesis, as we know that if we do not run forever, or result in an acceptable error, we will reduce to a final value on the stack. In that case, we simply appeal to the first case of $\mathcal{R}[\tau]$ and we are done. \square

Lemma E.0.3 (Monotonicity X). *If $(W, \varphi, v) \in \mathcal{V}^{\text{X}}[\tau]$ and $W' \sqsupseteq W$, then $(W', \varphi, v) \in \mathcal{V}^{\text{X}}[\tau]$*

Proof. This follows from the definition of world extension: step indices can decrease, which can only have the effect of bringing more terms into the relation, in the case that we run out of steps before we can rule our membership, and the heap typing can expand or mark existing locations as dead, neither of which rules out existing values being in the relation. \square

Lemma E.0.4 (Antireduction $_{\circ}$ X).

If $\forall W' \varphi' H H', S. (W', \varphi', \text{push } v_1; \text{push } v_2; \dots \text{push } v_n; P) \in \mathcal{E}^{\text{X}}[\tau] \circ$ and $W' \sqsubseteq W$, $H :_{\varphi} W$, $H' :_{\varphi \cup \varphi'} W'$, and $\langle H ; S ; P' ; P \rangle \xrightarrow{} \langle H' ; S, v_1, v_2, \dots, v_n ; P \rangle$ then $(W, \varphi, P'; P) \in \mathcal{E}^{\text{X}}[\tau] \circ$.*

Proof. We consider heap $H :_{\varphi} W$, arbitrary stack S . We know that if the term in question does not run forever (which, if it does, then the suffix P does as well, so we are done), then it steps to a terminal configuration $\langle H^F ; S^F ; \cdot \rangle$. We need to show that, assuming that is not an error, $S^F = S, v$ and for some φ^F and $W^F \sqsubseteq W$, $H^F :_{\varphi^F} W^F$ and $(W^F, \varphi^F, v) \in \mathcal{V}^{\text{X}}[\tau] \circ$. We know that $\langle H ; S ; P' ; P \rangle \xrightarrow{*} \langle H' ; S, v_1, \dots, v_n ; P \rangle$ and that for some $W' \sqsubseteq W$, $H' :_{\varphi \cup \varphi'} W'$. So we instantiate our first hypothesis with H' and S . After n steps, it is in exactly the configuration our term left off in. We know it doesn't run forever, and if it errors, similarly, our overall term must

error, so we conclude that it runs to a terminal configuration which due to confluence, will be the same one. Thus, we know $H^F :_{\varphi \cup \varphi' \cup \varphi_F} W^F$, which is stronger than we need, and $(W^F, \varphi^F, v) \in \mathcal{V}^X[\tau]$, exactly as needed. \square

Lemma E.0.5 (Monadic Bind X). *If $(W, \varphi_p, P) \in \mathcal{E}^X[\tau]_\bullet$, and $(W', \varphi_k \cup \varphi'_p, K[P']) \in \mathcal{E}^X[\tau']_\bullet$ whenever $(W', \varphi'_p, P') \in \mathcal{R}[\tau]$ and $W' \sqsupseteq W$, then $(W, \varphi_k \cup \varphi_p, K[P]) \in \mathcal{E}^X[\tau']_\bullet$.*

Proof. Given $(W, \varphi'_k, K') \in \mathcal{K}[\tau' \Rightarrow \tau'']$, we must show $(W, \varphi'_k \cup \varphi_k \cup \varphi_p, K'[K[P]]) \in \mathcal{E}^X[\tau'']_\circ$. Because $(W, \varphi_p, P) \in \mathcal{E}^X[\tau]_\bullet$, it suffices if $(W, \varphi_k \cup \varphi'_k, K'[K]) \in \mathcal{K}[\tau \Rightarrow \tau'']$. Given $(W', \varphi'_p, P') \in \mathcal{R}[\tau]$ where $W' \sqsupseteq W$, we must show $(W', \varphi'_k \cup \varphi_k \cup \varphi'_p, K'[K[P']]) \in \mathcal{E}^X[\tau'']_\circ$. By assumption, $(W', \varphi_k \cup \varphi'_p, K[P']) \in \mathcal{E}^X[\tau']_\bullet$, so $(W', \varphi'_k \cup \varphi_k \cup \varphi'_p, K'[K[P']]) \in \mathcal{E}^X[\tau'']_\circ$ by definition of $\mathcal{E}^X[\tau']_\bullet$. \square

Corollary E.0.6 (Antireduction_• X).

If $\forall W' \varphi' H H', S. (W', \varphi', \text{push } v_1; \text{push } v_2; \dots \text{push } v_n; P) \in \mathcal{E}^X[\tau]_\bullet$ and $W' \sqsubseteq W$, $H :_\varphi W$, $H' :_{\varphi \cup \varphi'} W'$, and $\langle H ; S ; P' ; P \rangle \xrightarrow{} \langle H' ; S, v_1, v_2, \dots, v_n ; P \rangle$ then $(W, \varphi, P'; P) \in \mathcal{E}^X[\tau]_\bullet$.*

Proof. In Lemma E.0.4, the only cases are divergence, (type-sound) termination, and failure. Here, we must also consider exceptions, but we can use Lemma E.0.5 as needed. Otherwise, the proof proceeds as in Lemma E.0.4. \square

Lemma E.0.7 (Thread X). *If $(W, \varphi_p, P) \in \mathcal{E}^X[\tau]_\bullet$, and $(W', \varphi_k \cup \varphi_v, K[push v]) \in \mathcal{E}^X[\tau']_\bullet$ whenever $(W', \varphi_v, v) \in \mathcal{V}^X[\tau]$ and $W' \sqsupseteq W$, then $(W, \varphi_k \cup \varphi_p, K[P]) \in \mathcal{E}^X[\tau']_\bullet$.*

Proof. By Lemma E.0.5, it suffices if $(W', \varphi_k \cup \varphi'_p, K[P']) \in \mathcal{E}^X[\tau']_\bullet$ given $(W', \varphi'_p, P') \in \mathcal{R}[\tau]$ where $W' \sqsupseteq W$. Unfolding $\mathcal{R}[\tau]$, there are two cases.

- $P' = \text{push } v$ for $(W', \varphi'_p, v) \in \mathcal{V}^X[\tau]$. Then apply the second premise.
- $P' = \text{push } [0, v]; \text{shift } _(); P''$ for $(W', \varphi_v, v) \in \mathcal{V}^X[\tau]$, $\varphi_v \subseteq \varphi'_p$. Given $(W', \varphi'_k, K') \in \mathcal{K}[\tau' \Rightarrow \tau'']$, we must show $(W', \varphi'_k \cup \varphi_k \cup \varphi'_p, K'[K[\text{push } [0, v]; \text{shift } _(); P'']]) \in \mathcal{E}^X[\tau'']_\circ$. Since $K = \overline{\text{push } v_k; [\cdot]; P_k}$, we must show $(W', \varphi'_k \cup \varphi_k \cup \varphi'_p, K'[\text{push } v; \text{push } [0, v]; \text{shift } _(); P''; P_k]) \in \mathcal{E}^X[\tau'']_\circ$. Applying Lemma E.0.4, it suffices if $(W'', \varphi'_k \cup \varphi_k \cup \varphi'_p, K'[\text{push } [0, v]; \text{shift } _(); P''; P_k]) \in \mathcal{E}^X[\tau'']_\circ$. But notice that $(W'', \varphi_k \cup \varphi'_p, \text{push } [0, v]; \text{shift } _(); P''; P_k) \in \mathcal{R}[\tau']$, so applying the definition of $\mathcal{K}[\tau' \Rightarrow \tau'']$ is sufficient.

\square

E.0.2 FunLang with \mathbf{X} Compatibility Lemmas

$$\llbracket \Gamma \vdash_{\mathbf{X}} P : \tau \equiv \forall W \varphi \gamma. (W, \varphi, \gamma) \in \mathcal{G}^{\mathbf{X}}[\Gamma] \implies (W, \text{flocs}(\gamma(P)), \gamma(P)) \in \mathcal{E}^{\mathbf{X}}[\tau] \rrbracket$$

Lemma E.0.8 (unit). *Show that $\llbracket \Gamma \vdash_{\mathbf{X}} \text{push } 0 : \text{unit} \rrbracket$.*

Proof. We are given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^{\mathbf{X}}[\gamma]$, where $\varphi = \text{flocs}(\gamma(\text{push } 0)) = \emptyset$, and need to show that $(W, \emptyset, \gamma(\text{push } 0)) \in \mathcal{E}^{\mathbf{X}}[\text{unit}]$.

Thus, we consider arbitrary continuation K with $(W, \varphi^k, K) \in \mathcal{K}[\text{unit} \Rightarrow \tau]$. We need to show that $(W, \varphi^k, K[\gamma(\text{push } 0)]) \in \mathcal{E}^{\mathbf{X}}[\tau]$. But this follows exactly from the definition of $\mathcal{R}[\tau]$. \square

Lemma E.0.9 (bool). *Show for any n , $\llbracket \Gamma \vdash_{\mathbf{X}} n : \text{bool} \rrbracket$.*

Proof. This proof is essentially identical to that of **unit**. \square

Lemma E.0.10 (if). *If $\llbracket \Gamma \vdash_{\mathbf{X}} P_1 : \text{bool} \rrbracket$, $\llbracket \Gamma \vdash_{\mathbf{X}} P_2 : \tau \rrbracket$, and $\llbracket \Gamma \vdash_{\mathbf{X}} P_3 : \tau \rrbracket$ then*

$$\llbracket \Gamma \vdash_{\mathbf{X}} P_1; \text{if} 0 P_2 P_3 : \tau \rrbracket$$

Proof. Unfolding $\llbracket \cdot \rrbracket$ and pushing substitutions, we must show

$$(W, \varphi, \gamma(P_1); \text{if} 0 \gamma(P_2) \gamma(P_3)) \in \mathcal{E}^{\mathbf{X}}[\tau]$$

given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^{\mathbf{X}}[\Gamma]$ where $\varphi = \bigcup \varphi_i$ and $\varphi_i = \text{flocs}(\gamma(P_i))$. Applying Lemma E.0.7 with the first premise, it suffices if

$$(W', \varphi_2 \cup \varphi_3, \text{push } n; \text{if} 0 \gamma(P_2) \gamma(P_3)) \in \mathcal{E}^{\mathbf{X}}[\tau]$$

given $(W', \emptyset, n) \in \mathcal{V}^{\mathbf{X}}[\text{bool}]$ and $W' \sqsupseteq W$. There are two cases.

- Suppose $n = 0$. Then by Lemma E.0.6, it suffices if

$$(W', \varphi_2, \gamma(P_2)) \in \mathcal{E}^{\mathbf{X}}[\tau]$$

Applying Lemma E.0.7 with the second premise, it suffices if

$$(W'', \varphi', \text{push } v) \in \mathcal{E}^{\mathbf{X}}[\tau]$$

where $(W'', \varphi', v) \in \mathcal{V}^{\mathbf{X}}[\tau]$, $W'' \sqsupseteq W'$. Then apply Lemmas E.0.1, E.0.2.

- Suppose $n \neq 0$. Then by Lemma E.0.6, it suffices if

$$(W', \varphi_3, \gamma(P_3)) \in \mathcal{E}^{\mathbf{X}}[\tau]$$

Applying Lemma E.0.7 with the third premise, it suffices if

$$(W'', \varphi', \text{push } v) \in \mathcal{E}^{\mathbf{X}}[\tau]$$

where $(W'', \varphi', v) \in \mathcal{V}^X[\tau]$, $W'' \sqsupseteq W'$. Then apply Lemmas E.0.1, E.0.2.

□

Lemma E.0.11 (int). *For any n , show $\llbracket \Gamma \vdash_X \text{push } n : \text{int} \rrbracket$.*

Proof. This proof is essentially identical to that of `unit`. □

Lemma E.0.12 (op- $=$). *If $\llbracket \Gamma \vdash_X P_1 : \text{int} \rrbracket$ and $\llbracket \Gamma \vdash_X P_2 : \text{int} \rrbracket$, then $\llbracket \Gamma \vdash_X P_1; P_2; \text{equal?} : \text{bool} \rrbracket$.*

Proof. Unfolding $\llbracket \cdot \rrbracket$ and pushing substitutions, we are to show

$$(W, \varphi, \gamma(P_1); \gamma(P_2); \text{equal?}) \in \mathcal{E}^X[\text{bool}]^\hookrightarrow$$

given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^X[\Gamma]$ where $\varphi = \bigcup \varphi_i$ and $\varphi_i = \text{flocs}(\gamma(P_i))$.

Applying Lemma E.0.7 twice, it sufficies if

$$(W', \emptyset, \text{push } n_1; \text{push } n_2; \text{equal?}) \in \mathcal{E}^X[\text{bool}]^\hookrightarrow$$

given $(W', \emptyset, n_i) \in \mathcal{V}^X[\text{int}]$ and $W' \sqsupseteq W$. Applying Lemma E.0.6, there are two cases:

- Suppose $n_1 = n_2$. Then we must show

$$(W', \emptyset, \text{push } 0) \in \mathcal{E}^X[\text{bool}]^\hookrightarrow$$

which we have by Lemmas E.0.1, E.0.2 and the definition of $\mathcal{V}^X[\text{bool}]$.

- Suppose $n_1 \neq n_2$. Then we must show

$$(W', \emptyset, \text{push } 1) \in \mathcal{E}^X[\text{bool}]^\hookrightarrow$$

which we have by Lemmas E.0.1, E.0.2 and the definition of $\mathcal{V}^X[\text{bool}]$.

□

Lemma E.0.13 (op- !). *If $\llbracket \Gamma \vdash_X P_1 : \text{int} \rrbracket$ and $\llbracket \Gamma \vdash_X P_2 : \text{int} \rrbracket$, then $\llbracket \Gamma \vdash_X P_1; P_2; \text{less?} : \text{bool} \rrbracket$.*

Proof. This proof is essentially identical to that of `=`. □

Lemma E.0.14 (op- $+$). *If $\llbracket \Gamma \vdash_X P_1 : \text{int} \rrbracket$ and $\llbracket \Gamma \vdash_X P_2 : \text{int} \rrbracket$, then $\llbracket \Gamma \vdash_X P_1; P_2; \text{add} : \text{int} \rrbracket$.*

Proof. This proof is essentially identical to that of `=`. □

Lemma E.0.15 (var). $\llbracket \Gamma \vdash_X \text{push } x : \tau \rrbracket$

Proof. Unfolding $\llbracket \cdot \rrbracket$ and pushing substitutions, we are to show

$$(W, \varphi, \text{push } v) \in \mathcal{E}^X[\tau] \downarrow$$

given $(W, \varphi^\dagger \cup \varphi, \gamma[x \mapsto v]) \in \mathcal{G}^X[\Gamma]$ where $(W, \varphi, v) \in \mathcal{V}^X[\tau]$. Then apply Lemmas E.0.1, E.0.2. \square

Lemma E.0.16 (pair). *If $\llbracket \Gamma \vdash_X P_1 : \tau_1$ and $\llbracket \Gamma \vdash_X P_2 : \tau_2$ then $\llbracket \Gamma \vdash_X P_1; P_2; \text{lam } x_2.\text{lam } x_1.\text{push } [x_1, x_2] : \tau_1 \times \tau_2$*

Proof. Unfolding $\llbracket \cdot \rrbracket$ and pushing substitutions, we are to show

$$(W, \varphi, \gamma(P_1); \gamma(P_2); \text{lam } x_2.\text{lam } x_1.\text{push } [x_1, x_2]) \in \mathcal{E}^X[\tau_1 \times \tau_2] \downarrow$$

given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^X[\Gamma]$ where $\varphi = \bigcup \varphi_i$ and $\varphi_i = \text{flocs}(\gamma(P_i))$.

Applying Lemma E.0.7 twice, it suffices if

$$(W', \varphi', \text{push } v_1; \text{push } v_2; \text{lam } x_2.\text{lam } x_1.\text{push } [x_1, x_2]) \in \mathcal{E}^X[\tau_1 \times \tau_2] \downarrow$$

given $(W', \varphi'_i, v_i) \in \mathcal{V}^X[\tau_i]$ and $\varphi' = \bigcup \varphi'_i$ and $W' \sqsupseteq W$. Applying Lemma E.0.6, it suffices if

$$(W'', \varphi', \text{push } [v_1, v_2]) \in \mathcal{E}^X[\tau_1 \times \tau_2] \downarrow$$

given $W'' \sqsupseteq W'$, which we have by Lemmas E.0.1, E.0.2 and the definition of $\mathcal{V}^X[\tau_1 \times \tau_2]$. \square

Lemma E.0.17 (fst). *If $\llbracket \Gamma \vdash_X P : \tau_1 \times \tau_2$, then $\llbracket \Gamma \vdash_X P; \text{push } 0; \text{idx} : \tau_1$.*

Proof. Unfolding $\llbracket \cdot \rrbracket$ and pushing substitutions, we are to show

$$(W, \varphi, \gamma(P); \text{push } 0; \text{idx}) \in \mathcal{E}^X[\tau_1] \downarrow$$

given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^X[\Gamma]$ where $\varphi = \text{flocs}(\gamma(P))$.

Applying Lemma E.0.7, it suffices if

$$(W', \varphi', \text{push } [v_1, v_2]; \text{push } 0; \text{idx}) \in \mathcal{E}^X[\tau_1] \downarrow$$

where $(W', \varphi'_i, v_i) \in \mathcal{V}^X[\tau_i]$ and $\varphi' = \bigcup \varphi'_i$ and $W' \sqsupseteq W$. Applying Lemma E.0.6, it suffices if

$$(W'', \varphi'_1, \text{push } v_1) \in \mathcal{E}^X[\tau_1] \downarrow$$

where $W'' \sqsupseteq W'$, which we have by Lemmas E.0.1, E.0.2. \square

Lemma E.0.18 (snd). *If $\llbracket \Gamma \vdash_X P : \tau_1 \times \tau_2$, then $\llbracket \Gamma \vdash_X P_1; \text{push } 1; \text{idx} : \tau_2$.*

Proof. As in Lemma E.0.17. □

Lemma E.0.19 (inl). *If* $\llbracket \Gamma \vdash x : \tau_1 \text{, then } \llbracket \Gamma \vdash x : P; \text{lam } x.\text{push } [0, x] : \tau_1 + \tau_2 \rrbracket$.

Proof. Unfolding $\llbracket \cdot \rrbracket$ and pushing substitutions, we are to show

$$(W, \varphi, \gamma(P); \text{lam } x.\text{push } [0, x]) \in \mathcal{E}^X[\tau_1 + \tau_2] \triangleleft$$

given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^X[\Gamma]$ where $\varphi = \text{flocs}(\gamma(P))$.

Applying Lemma E.0.7, it suffices if

$$(W', \varphi', \text{push } v; \text{lam } x.\text{push } [0, x]) \in \mathcal{E}^X[\tau_1 + \tau_2] \triangleleft$$

given $(W', \varphi', v) \in \mathcal{V}^X[\tau_1]$ and $W' \sqsupseteq W$. Applying Lemma E.0.6, it suffices if

$$(W'', \varphi', \text{push } [0, v]) \in \mathcal{E}^X[\tau_1 + \tau_2] \triangleleft$$

given $W'' \sqsupseteq W'$, which we have by Lemmas E.0.1, E.0.2 and the definition of $\mathcal{V}^X[\tau_1 + \tau_2]$. □

Lemma E.0.20 (inr). *If* $\llbracket \Gamma \vdash x : \tau_2 \text{, then } \llbracket \Gamma \vdash x : P; \text{lam } x.\text{push } [1, x] : \tau_1 + \tau_2 \rrbracket$.

Proof. As in Lemma E.0.19. □

Lemma E.0.21 (match). *If* $\llbracket \Gamma \vdash x : \tau_1 + \tau_2 \text{, } \llbracket \Gamma, x : \tau_1 \vdash x : P_1 : \tau, \text{ and } \llbracket \Gamma, y : \tau_2 \vdash x : P_2 : \tau \text{, then } \llbracket \Gamma \vdash x : P_0; \text{DUP; push } 1; \text{idx; SWAP; push } 0; \text{idx; if0 } (\text{lam } x.P_1) (\text{lam } y.P_2) : \tau \rrbracket$.

Proof. Unfolding $\llbracket \cdot \rrbracket$ and pushing substitutions, we must show

$$(W, \varphi, \gamma(P_0); \text{DUP; push } 1; \text{idx; SWAP; push } 0; \text{idx; if0 } (\text{lam } x.\gamma(P_1)) (\text{lam } y.\gamma(P_2))) \in \mathcal{E}^X[\tau] \triangleleft$$

given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^X[\Gamma]$ where $\varphi = \bigcup \varphi_i$ and $\varphi_i = \text{flocs}(\gamma(P_i))$. Applying Lemma E.0.7 with the first premise, it suffices if

$$(W', \varphi', \text{push } [n, v]; \text{DUP; push } 1; \text{idx; SWAP; push } 0; \text{idx; if0 } (\text{lam } x.\gamma(P_1)) (\text{lam } y.\gamma(P_2))) \in \mathcal{E}^X[\tau] \triangleleft$$

given $(W', \varphi'_0, [n, v]) \in \mathcal{V}^X[\tau_1 + \tau_2]$ and $W' \sqsupseteq W$ where $\varphi' = \varphi'_0 \cup \varphi_1 \cup \varphi_2$. There are two cases.

- Suppose $n = 0$ and $(W', \varphi'_0, v) \in \mathcal{V}^X[\tau_1]$. Then by Lemma E.0.6 and pushing substitutions, it suffices if

$$(W', \varphi_1, \gamma[x \mapsto v](P_1)) \in \mathcal{E}^X[\tau]'$$

Applying Lemma E.0.7 with the second premise, it suffices if

$$(W'', \varphi'', \text{push } v') \in \mathcal{E}^X[\tau]'$$

where $(W'', \varphi'', v) \in \mathcal{V}^X[\tau]$, $W'' \sqsupseteq W'$. Then apply Lemmas E.0.1, E.0.2.

- Suppose $n = 1$ and $(W', \varphi'_0, v) \in \mathcal{V}^X[\tau_2]$. Then by Lemma E.0.6 and pushing substitutions, it suffices if

$$(W', \varphi_2, \gamma[y \mapsto v](P_2)) \in \mathcal{E}^X[\tau]'$$

Applying Lemma E.0.7 with the third premise, it suffices if

$$(W'', \varphi'', \text{push } v') \in \mathcal{E}^X[\tau]'$$

where $(W'', \varphi'', v) \in \mathcal{V}^X[\tau]$, $W'' \sqsupseteq W'$. Then apply Lemmas E.0.1, E.0.2.

□

Lemma E.0.22 (fold). *If $\llbracket \Gamma \vdash_X P : \tau[\mu\alpha.\tau/\alpha]$, then $\llbracket \Gamma \vdash_X P : \mu\alpha.\tau$.*

Proof. Unfolding $\llbracket \cdot$ and pushing substitutions, we are to show

$$(W, \varphi, \gamma(P)) \in \mathcal{E}^X[\mu\alpha.\tau]'$$

given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^X[\Gamma]$ where $\varphi = \text{flocs}(\gamma(P))$.

Applying Lemma E.0.7 with the first premise, it suffices if

$$(W', \varphi', \text{push } v) \in \mathcal{E}^X[\mu\alpha.\tau]'$$

given $(W', \varphi', v) \in \mathcal{V}^X[\tau[\mu\alpha.\tau/\alpha]]$ and $W' \sqsupseteq W$. Applying Lemmas E.0.1, E.0.2, it suffices if

$$(W', \varphi', v) \in \mathcal{V}^X[\mu\alpha.\tau]$$

which is immediate from the assumption, the definition of $\mathcal{V}^X[\mu\alpha.\tau]$, and Lemma E.0.3. □

Lemma E.0.23 (unfold). *If $\llbracket \Gamma \vdash_X P : \mu\alpha.\tau$, then $\llbracket \Gamma \vdash_X P ; \text{noop} : \tau[\mu\alpha.\tau]$.*

Proof. Unfolding $\llbracket \cdot \rrbracket$ and pushing substitutions, we are to show

$$(W, \varphi, \gamma(\mathsf{P}); \text{noop}) \in \mathcal{E}^{\mathbf{X}} \llbracket \tau[\mu\alpha.\tau] \rrbracket \triangleleft$$

given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^{\mathbf{X}} \llbracket \Gamma \rrbracket$ where $\varphi = \text{flocs}(\gamma(\mathsf{P}))$.

Applying Lemma E.0.7 with the first premise, it suffices if

$$(W', \varphi', \text{push } v; \text{noop}) \in \mathcal{E}^{\mathbf{X}} \llbracket \tau[\mu\alpha.\tau] \rrbracket \triangleleft$$

given $(W', \varphi', v) \in \mathcal{V}^{\mathbf{X}} \llbracket \mu\alpha.\tau \rrbracket$ and $W' \sqsupseteq W$. Applying Lemma E.0.6, it suffices if

$$(W'', \varphi', \text{push } v) \in \mathcal{E}^{\mathbf{X}} \llbracket \tau[\mu\alpha.\tau] \rrbracket \triangleleft$$

given $W'' \sqsupseteq W'$ (N.B., we take care to strictly advance the world, here). Applying Lemmas E.0.1, E.0.2, it suffices if

$$(W'', \varphi', v) \in \mathcal{V}^{\mathbf{X}} \llbracket \tau[\mu\alpha.\tau] \rrbracket$$

which is immediate from the assumption, the definition of $\mathcal{V}^{\mathbf{X}} \llbracket \mu\alpha.\tau \rrbracket$, and Lemma E.0.3. \square

Lemma E.0.24 (fun). *If $\llbracket \Gamma, f : (\tau_1, \dots, \tau_n) \rightarrow \tau', x_i : \tau_i \vdash_{\mathbf{X}} P : \tau' \rrbracket$, then*

$$\llbracket \Gamma \vdash_{\mathbf{X}} \text{push } (\text{thunk push } (\text{thunk lam } f.\text{lam } x_n. \dots. \text{ lam } x_1.P); \text{fix}) : (\tau_1, \dots, \tau_n) \rightarrow \tau' \rrbracket$$

Proof. Unfolding $\llbracket \cdot \rrbracket$ and pushing substitutions, we are to show

$$\begin{aligned} & (W, \varphi, \text{push } (\text{thunk push } (\text{thunk lam } f.\text{lam } x_n. \dots. \text{ lam } x_1.\gamma(\mathsf{P})); \text{fix})) \\ & \in \mathcal{E}^{\mathbf{X}} \llbracket (\tau_1, \dots, \tau_n) \rightarrow \tau' \rrbracket \triangleleft \end{aligned}$$

given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^{\mathbf{X}} \llbracket \Gamma \rrbracket$ where $\varphi = \text{flocs}(\gamma(\mathsf{P}))$.

Applying Lemmas E.0.1, E.0.2, it suffices if

$$\begin{aligned} & (W, \varphi, \text{thunk push } (\text{thunk lam } f.\text{lam } x_n. \dots. \text{ lam } x_1.\gamma(\mathsf{P})); \text{fix}) \\ & \in \mathcal{V}^{\mathbf{X}} \llbracket (\tau_1, \dots, \tau_n) \rightarrow \tau' \rrbracket \end{aligned}$$

Unfolding the definition of $\mathcal{V}^{\mathbf{X}} \llbracket (\tau_1, \dots, \tau_n) \rightarrow \tau' \rrbracket$ and pushing substitutions, we must show

$$\begin{aligned} & (W', \varphi', \gamma[x_i \mapsto v_i, f \mapsto \text{thunk push } (\text{thunk lam } f.\text{lam } x_n. \dots. \text{ lam } x_1.\gamma(\mathsf{P})); \text{fix}](P)) \\ & \in \mathcal{E}^{\mathbf{X}} \llbracket \tau' \rrbracket \triangleleft \end{aligned}$$

given $W' \sqsupseteq W$ and $(W', \varphi_i, v_i) \in \mathcal{V}^{\mathbf{X}} \llbracket \tau_i \rrbracket$ where $\varphi' = \bigcup \varphi_i \cup \varphi \subset W'.\Psi$, which is immediate from the premise. \square

Lemma E.0.25 (app). *If $\llbracket \Gamma \vdash_{\mathbf{X}} P_0 : (\tau_1, \dots, \tau_n) \rightarrow \tau' \rrbracket$ and for $i \in \{1, \dots, n\} \llbracket \Gamma \vdash_{\mathbf{X}} P_i : \tau_i \rrbracket$ then $\llbracket \Gamma \vdash_{\mathbf{X}} P_0; P_1; \text{SWAP} \dots P_n; \text{SWAP}; \text{call} : \tau' \rrbracket$*

Proof. Unfolding $\llbracket \cdot \rrbracket$ and pushing substitutions, we are to show

$$(W, \varphi, \gamma(P_0); \gamma(P_1); \text{SWAP} \dots \gamma(P_n); \text{SWAP}; \text{call}) \in \mathcal{E}^{\mathbf{X}}[\tau'] \triangleleft$$

given $(W, \varphi^\dagger, \gamma) \in \mathcal{G}^{\mathbf{X}}[\Gamma]$ where $\varphi_i = \text{flocs}(P_i)$ and $\varphi = \bigcup P_i$.

Applying Lemmas E.0.7, E.0.6 with the premises in order, it suffices if

$$(W', \varphi', \text{push } v_1; \dots; \text{push } v_n; \text{push } (\text{thunk push } (\text{thunk lam } f.\text{lam } x_n. \dots. \text{lam } x_1.P); \text{fix}); \text{call}) \in \mathcal{E}^{\mathbf{X}}[\tau'] \triangleleft$$

given $W' \sqsupseteq W$, $(W', \varphi'_i, v_i) \in \mathcal{V}^{\mathbf{X}}[\tau_i]$ for $i > 0$, and

$$(W', \varphi'_0, \text{thunk push } (\text{thunk lam } f.\text{lam } x_n. \dots. \text{lam } x_1.P); \text{fix}) \in \mathcal{V}^{\mathbf{X}}[(\tau_1, \dots, \tau_n) \rightarrow \tau']$$

where $\varphi' = \bigcup \varphi'_i$. Applying Lemma E.0.6, it suffices if

$$(W', \varphi', [x_i \mapsto v_i, f \mapsto \text{thunk push } (\text{thunk lam } f.\text{lam } x_n. \dots. \text{lam } x_1.P); \text{fix}](P)) \in \mathcal{E}^{\mathbf{X}}[\tau'] \triangleleft$$

which is immediate from the definition of $\mathcal{V}^{\mathbf{X}}[(\tau_1, \dots, \tau_n) \rightarrow \tau']$. \square