Towards Certified Separate Compilation for Concurrent Programs

Extended Version

Hanru Jiang[*]
University of Science and Technology of China, China
hanru219@mail.ustc.edu.cn

Hongjin Liang[*‡]
Nanjing University, China
hongjin@nju.edu.cn

Siyang Xiao Junpeng Zha
University of Science and Technology of China, China
{yutio888,jpzha}@mail.ustc.edu.cn

Xinyu Feng[†‡]
Nanjing University, China
xyfeng@nju.edu.cn

Abstract

Certified separate compilation is important for establishing end-to-end guarantees for certified systems consisting of multiple program modules. There has been much work building certified compilers for sequential programs. In this paper, we propose a language-independent framework consisting of the key semantics components and lemmas that bridge the verification gap between the compilers for sequential programs and those for (race-free) concurrent programs, so that the existing verification work for the former can be reused. One of the key contributions of the framework is a novel footprint-preserving compositional simulation as the compilation correctness criterion. The framework also provides a new mechanism to support confined benign races which are usually found in efficient implementations of synchronization primitives.

With our framework, we develop CASCompCert, which extends CompCert for certified separate compilation of race-free concurrent Clight programs. It also allows linking of concurrent Clight modules with x86-TSO implementations of synchronization primitives containing benign races. All our work has been implemented in the Coq proof assistant.

CCS Concepts • **Theory of computation → Program verification**; **Abstraction**; • **Software and its engineering → Correctness**; **Semantics**; *Concurrent programming languages*; *Compilers.*

Keywords Concurrency, Certified Compilers, Data-Race-Freedom, Simulations

ACM Reference Format:
Hanru Jiang, Hongjin Liang, Siyang Xiao, Junpeng Zha, and Xinyu Feng. 2019. Towards Certified Separate Compilation for Concurrent Programs: Extended Version. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 59 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

[*]The first two authors contributed equally and are listed alphabetically.
[†]Corresponding author.
[‡]Also with State Key Laboratory for Novel Software Technology.

Conference'17, July 2017, Washington, DC, USA
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Separate compilation is important for real-world systems, which usually consist of multiple program modules that need to be compiled independently. Correct compilation then needs to guarantee that the target modules can work together and preserve the semantics of the source program as a whole. It requires not only that individual modules be compiled correctly, but also that the expected interactions between modules be preserved at the target.

CompCert [16], the most well-known certified realistic compiler, establishes the semantics preservation property for compilation of *sequential* Clight programs, but with no explicit support of separate compilation. To support general separate compilation, Stewart et al. [29] develop Compositional CompCert, which allows the modules to call each other's external functions. Like CompCert, Compositional CompCert only supports sequential programs too.

Stewart et al. [29] do argue that Compositional CompCert may be applied for data-race-free (DRF) concurrent programs, since they behave the same in the standard interleaving semantics as in certain non-preemptive semantics where switches between threads occur only at certain designated program points. The sequential compilation is sound as long as the switch points are viewed as external function calls so that optimizations do not go beyond them.

Although the argument is plausible, there are still significant challenges to actually implement it. We need proper formulation of the non-preemptive semantics and the notion of DRF. Then we need to indeed prove that DRF programs have the same behaviors (including termination) in the interleaving semantics as in the non-preemptive semantics, and verify that the compilation preserves DRF. More importantly, the formulation and the proofs should be done in a compositional and language-independent way, to allow separate compilation where the modules can be implemented in different languages. Reusing the proofs of CompCert is challenging too, as memory allocation for multiple threads may require a different memory model from that of CompCert, and the non-deterministic semantics also makes it difficult to reuse the downward simulation in CompCert.

In addition, the requirement of DRF could be sometimes overly restrictive. Although Boehm [3] points out there are essentially *no* "benign" races in source programs, and languages like C/C++ essentially give no semantics to racy programs [4], it is still meaningful to allow benign races in machine language code for better performance (e.g., the spin locks in Linux). Also machine languages commonly allow relaxed behaviors. Can we support realistic target code with potential benign races in relaxed machine models?

There has been work on verified compilation for concurrent programs [20, 27], but with only limited support of compositionality. We explain the major challenges in detail in Sec. 2, and discuss more related work in Sec. 8.

In this paper, we propose a language-independent framework consisting of the key semantics components and verification steps that bridge the gap between compilation for sequential programs and for DRF concurrent programs. We apply our framework to verify the correctness of CompCert for separate compilation of DRF programs to x86 assembly. We also extend the framework to allow *confined benign races* in x86-TSO (confined in that the racy code must execute in a separate region of memory and have race-free abstraction), so that optimized implementations of synchronization primitives like spin-locks in Linux can be supported. Our work is based on previous work on certified compilation, but makes the following new contributions:

- We design a compositional *footprint-preserving simulation* as the correctness formulation of separate compilation for sequential modules (Sec. 4). It considers module interactions at both external function calls and synchronization points, thus is compositional with respect to both module linking and non-preemptive concurrency. It also requires the footprints (i.e., the set of memory locations accessed during the execution) of the source and target modules to be related. This way we effectively reduce the proof of the compiler's preservation of DRF, a whole program property, to the proof of *local* footprint preservation.
- We work with an *abstract* programming language (Sec. 3), which is not tied to any specific synchronization constructs such as locks but uses abstract labels to model how such constructs interact with other modules. It also abstracts away the concrete primitives that access memory. We introduce the notion of *well-defined languages* to enforce a set of constraints over the state transitions and the related footprints, which actually give an extensional interpretation of footprints. These constraints are satisfied by various real languages such as Clight and x86 assembly. With the abstract language, we study the equivalence between preemptive and non-preemptive semantics (Sec. 3.3), the equivalence between DRF and NPDRF (the notion of race-freedom defined in the non-preemptive setting, shown in Sec. 5), and the properties of our new simulation. As a result, the lemmas in our proof framework are re-usable when instantiating to real languages.
- We prove a strengthened DRF-guarantee theorem for the x86-TSO model (Sec. 7.3). It allows an x86-TSO program to call a (x86-TSO) module with benign races. If one can replace the racy module with a more abstract version so that the resulting program is DRF in the sequentially consistent (SC) semantics, then the original racy x86-TSO program would behave the same with the more abstract program in the SC semantics. This way we allow the target programs to have confined benign races in the relaxed x86-TSO model. We use this approach to support efficient x86-TSO implementations of locks.
- Putting all these together, our framework (see Fig. 2 and Fig. 3) successfully builds certified separate compilation for concurrent programs *from sequential compilation*. It highlights the importance of DRF preservation for correct compilation. It also shows a possible way to adapt the existing work of CompCert and Compositional CompCert to interleaving concurrency.
- As an instantiation of our language-independent compilation verification framework, we develop the certified compiler CASCompCert[1] (Sec. 7). We instantiate the source and target languages as Clight and x86 assembly. We also provide an efficient x86-TSO implementation of locks as a synchronization library. CASCompCert reuses the compilation passes of CompCert-3.0.1 [6] (including all the translation passes and four optimization passes). We prove that they satisfy our new compilation correctness criterion, where we reuse a considerable amount of the original CompCert proofs, with minor adjustment for footprint-preservation. The proofs for each pass take less than one person week on average.

Supplementary materials for this paper, including the Coq development and a technical report (TR), are publicly available at https://plax-lab.github.io/publications/ccc/.

2 Informal Development

Below we first give an overview of the main ideas in CompCert [16, 17] and Compositional CompCert [29] as starting points for our work. Then we explain the challenges and our ideas in reusing them to build certified separate compilation for concurrent programs.

[1]It is short for an extended **CompCert** with the support of **C**oncurrency, **A**bstraction and **S**eparate compilation.

2.1 CompCert

Figure 1(a) shows the key proof structure of CompCert. The compilation *Comp* is correct, if for every source program S, the compiled code C preserves the semantics of S. That is,

$$\text{Correct}(Comp) \overset{\text{def}}{=} \forall S, C.\ Comp(S) = C \implies S \approx C\ .$$

The semantics preservation $S \approx C$ requires S and C have the same sets of observable event traces:

$$S \approx C \text{ iff } \forall \mathcal{B}.\ Etr(S, \mathcal{B}) \iff Etr(C, \mathcal{B})\ .$$

Here we write $Etr(S, \mathcal{B})$ to mean that an execution of S produces the observable event trace $\mathcal{B}$, and likewise for C.

To verify $S \approx C$, CompCert relies on the determinism of the target language (written as $\text{Det}(C)$ in Fig. 1(a)) and proves only the downward direction $S \sqsubseteq C$, i.e., S refines C:

$$S \sqsubseteq C \text{ iff } \forall \mathcal{B}.\ Etr(S, \mathcal{B}) \implies Etr(C, \mathcal{B})\ .$$

The determinism $\text{Det}(C)$ ensures that C admits only one event trace, so we can derive the upward refinement $S \sqsupseteq C$ from $S \sqsubseteq C$. The latter is then proved by constructing a (downward) simulation relation $S \precsim C$, depicted in Fig. 1(c). However, the simulation $\precsim$ relates whole programs only and it does not take into account the interactions with other modules which may update the shared resource. So it is not compositional and does *not* support separate compilation.

2.2 Compositional CompCert

Compositional CompCert supports separate compilation by re-defining the simulation relation for modules. Figure 1(b) shows its proof structure. Suppose we make separate compilation $Comp_1, \dots, Comp_n$. Each $Comp_i$ transforms a source module S_i to a target module C_i. The overall compilation is correct if, when linked together, the target modules $C_1 \circ \dots \circ C_n$ preserve the semantics of the source modules $S_1 \circ \dots \circ S_n$ (here we write $\circ$ as the module-linking operator). For example, the following program consists of two modules. The function `f` in module `S1` calls the external function `g`. The external module `S2` may access the variable `b` in `S1`.

```
// Module S1
extern void g(int *x);        // Module S2
int f(){                      int g(int *x){
  int a = 0, b = 0;             *x = 3;
  g(&b);                      }
  return a + b; }
```
(2.1)

Suppose the two modules `S1` and `S2` are independently compiled to the target modules `C1` and `C2`. The correctness of the overall compilation requires $(\texttt{S1} \circ \texttt{S2}) \approx (\texttt{C1} \circ \texttt{C2})$.

With the determinism of the target modules, we only need to prove the downward refinement $(\texttt{S1} \circ \texttt{S2}) \sqsubseteq (\texttt{C1} \circ \texttt{C2})$, which is reduced to proving $(\texttt{S1} \circ \texttt{S2}) \precsim (\texttt{C1} \circ \texttt{C2})$, just as in CompCert. Ideally we hope to know $(\texttt{S1} \circ \texttt{S2}) \precsim (\texttt{C1} \circ \texttt{C2})$ from $\texttt{S1} \precsim \texttt{C1}$ and $\texttt{S2} \precsim \texttt{C2}$, and ensure the latter two by correctness of $Comp_1, \dots, Comp_n$. However, the CompCert simulation $\precsim$ is not compositional.

To achieve compositionality, Compositional CompCert defines the simulation relation $\precsim'$ shown in Fig 1(d). It is

$$\begin{tikzcd}
S \approx C \\
S \sqsubseteq C \arrow[u, Rightarrow, "\text{Det}(C)"'] \\
S \precsim C \arrow[u, Rightarrow]
\end{tikzcd}$$

(a) CompCert

$$\begin{tikzcd}
S_1 \circ \dots \circ S_n \approx C_1 \circ \dots \circ C_n \\
S_1 \circ \dots \circ S_n \sqsubseteq C_1 \circ \dots \circ C_n \arrow[u, Rightarrow, "\text{Det}(C_1 \circ \dots \circ C_n)"'] \\
S_1 \circ \dots \circ S_n \precsim C_1 \circ \dots \circ C_n \arrow[u, Rightarrow] \\
\forall i.\ \text{R}, \text{G} \vdash S_i \precsim' C_i \arrow[u, Rightarrow]
\end{tikzcd}$$

(b) Compositional CompCert

$$\begin{tikzcd}
S \arrow[r] \arrow[d, dash, "\precsim"'] & S' \arrow[d, dashed, dash, "\precsim"] \\
C \arrow[r, dashed, "+"] & C'
\end{tikzcd}$$

(c) $S \precsim C$

$$\begin{tikzcd}
S_1 \arrow[r] \arrow[d, dash, "\precsim'"'] & S_2 \arrow[r, thick] \arrow[d, dashed, dash, "\precsim'"'] \arrow[dl, phantom, "G"] & S_3 \arrow[r] \arrow[d, dashed, dash, "\precsim'"'] \arrow[dl, phantom, "R"] & S_4 \arrow[d, dashed, dash, "\precsim'"'] \arrow[dl, phantom, "G"] \\
C_1 \arrow[r, dashed, "+"] & C_2 \arrow[r, thick] & C_3 \arrow[r, dashed, "+"] & C_4
\end{tikzcd}$$

(d) $R, G \vdash S \precsim' C$

Figure 1. Proof structures of certified compilation

parameterized with the interactions between modules, formulated as the rely/guarantee conditions R and G [14]. The rely condition R of a module specifies the general callee behaviors happen at the *external function calls* of the current module (shown as the thick arrows in Fig. 1(d)). The guarantee G specifies the possible transitions made by the module itself (the thin arrows). The simulation is compositional as long as the R and G of linked modules are compatible.

Compositional CompCert proves that the CompCert compilation passes satisfy the new simulation $\precsim'$. The intuition is that the compiler optimizations do not go beyond external calls (unless only local variables get involved). That is, for the example (2.1), the compiler cannot do optimizations based on the (wrong) assumption that `b` is 0 when `f` returns.

Since the R steps happen only at the external calls, it cannot be applied to concurrent programs, where module/thread interactions may occur at any program point. However, if we consider race-free concurrent programs only, where threads are properly synchronized, we may consider the interleaving at certain synchronization points only. It has been a folklore theorem that DRF programs in interleaving semantics behave the same as in non-preemptive semantics. For instance, the following program (2.2) uses a lock to synchronize the accesses of the shared variables, and it is race-free. Its behaviors are the same as those when the threads yield controls at `lock()` and `unlock()` only. That is, we can view lines `1-2` and lines `4-5` in either thread as sequential code that will not be interfered by the other. The interactions occur only at the boundaries of the critical regions.

```
1  r1 = 1;         ||  r2 = 2;
2  r1 = r1 + 1;    ||  r2 = r2 + 1;
3  lock();         ||  lock();
4    x = 1;        ||    x = 2;
5    y = x + 1;    ||    y = x + 1;
6  unlock();       ||  unlock();
```
(2.2)

Intuitively, we can use Compositional CompCert to compile the program, where the code segment between two consecutive switch points is compiled as sequential code.

By viewing the switch points as special external function calls, the simulation $\precsim'$ can be applied to relate the non-preemptive executions of the source and target modules.

2.3 Challenges and Our Approaches

Although the idea of applying Compositional CompCert to compile DRF programs is plausible, we have to address several key challenges to really implement it.

Q: How to give language-independent formulations of DRF and non-preemptive semantics? The interaction semantics introduced in Compositional CompCert describes modules' interactions without referring to the concrete languages used to implement the modules. This allows composition of modules implemented in different languages. We would like to follow the semantics framework, but how do we define DRF and non-preemptive semantics if we do not even know the concrete synchronization constructs and the commands that access memory?

A: We extend the module-local semantics in Compositional CompCert so that each local step of a module reports its footprints, i.e. the memory locations it accesses. Instead of relying on the concrete memory-access commands to define what valid footprints are, we introduce the notion of *well-defined languages* (in Sec. 3) to specify the requirements over the state transitions and the related footprints. For instance, we require the behavior of each step is affected by the read set only, and each step does *not* touch the memory outside of the write set. When we instantiate the framework with real languages, we prove they satisfy these requirements.

Besides, we also allow module-local steps to generate messages `EntAtom` and `ExtAtom` to indicate the boundary of the atomic operations inside the module. The concrete commands that generate these messages are unspecified, which can be different in different modules.

Q: What memory model to use in the proofs? The choice of memory models could greatly affect the complexity of proofs. For instance, using the same memory model as CompCert allows us to reuse CompCert proofs, but it also causes many problems. The CompCert memory model records the next available block number `nextblock` for memory allocation. Using the model under a concurrent setting may ask all threads to share one `nextblock`. Then allocation in one thread would affect the subsequent allocations in other threads. This breaks the property that re-ordering non-conflicting operations from different threads would *not* affect the final states, which is a key lemma we rely on to prove the equivalence between preemptive and non-preemptive semantics for DRF programs. In addition, sharing the `nextblock` by all threads also means we have to keep track of the ownership of each allocated block when we reason about footprints.

A: We decide to use a different memory model. We reserve separate address spaces F for memory allocation in different threads (see Sec. 3). Therefore allocation of one thread would not affect behaviors of others. This greatly simplifies the semantics and the proofs, but also makes it almost impossible to reuse CompCert proofs (see Sec. 7.2). We address this problem by establishing some semantics equivalence between our memory model and the CompCert memory model (shown in Sec. 7.2).

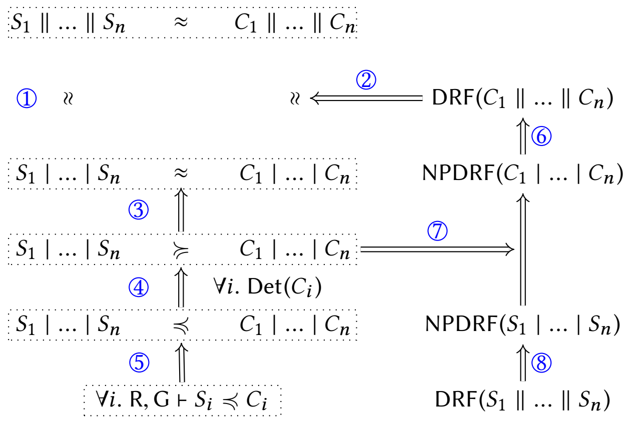
Q: How to compositionally prove DRF-preservation? The simulation $\precsim'$ in Compositional CompCert cannot ensure DRF-preservation. As we have explained, DRF is a whole-program property, and so is DRF-preservation. To support separate compilation, we need to reduce DRF-preservation to some thread-local property.

A: We propose a new compositional simulation $\preccurlyeq$ (see Sec. 4). Based on the simulation in Fig. 1(d), we additionally require *footprint consistency* saying that the target C should have the same or smaller footprints than the source S during related transitions. For instance, when compiling lines `4-5` of the left thread in (2.2), the target is only allowed to read `x` and write to `x` and `y`. Note that we check footprint consistency at switch points only. This way we allow compiler optimizations as long as they do not go beyond the switch points. For lines `4-5` of the left thread in (2.2), the target could be `y=2;x=1` where the writes to `x` and `y` are swapped.

Q: Can we flip refinement/simulation in spite of non-determinism? As we explained, the last steps of CompCert and Compositional CompCert in Fig. 1 derive semantics equivalence $\approx$ (or the upward refinement $\sqsupseteq$) from the downward refinement $\sqsubseteq$ using determinism of target programs. Actually the simulations $\precsim$ and $\precsim'$ can also be flipped if the target programs are deterministic. It is unclear if the refinement or simulation can still be flipped in concurrent settings where programs have non-deterministic behaviors. The problem is that the target program can be more fine-grained and have more non-deterministic interleavings than the source.

A: With non-preemptive semantics, the non-determinism occurs only at certain switch points. Then, in our simulation, we require the source and target to switch at the same time and to the same thread. As a result, although the switching step is still non-deterministic, there exists a one-to-one correspondence between the switching steps of the source and of the target. Thus such non-determinism will not affect the flip of our simulation.

Q: How to support benign races and relaxed machine models? It is difficult to write useful DRF programs within *sequential* languages (e.g., CompCert Clight) because they do not provide synchronization primitives (e.g., locks). Nevertheless, we can implement synchronization primitives as external modules so that the threads written in the sequential languages can call functions (e.g., lock-acquire and lock-release) in the external modules. In practice, the efficient implementation π_o of the synchronization primitives may be written directly in assembly languages, and may introduce


$S_1 \parallel \dots \parallel S_n \approx C_1 \parallel \dots \parallel C_n$
①
②
$\mathsf{DRF}(C_1 \parallel \dots \parallel C_n)$
⑥
$S_1 \mid \dots \mid S_n \approx C_1 \mid \dots \mid C_n$
$\mathsf{NPDRF}(C_1 \mid \dots \mid C_n)$
③
$S_1 \mid \dots \mid S_n \succcurlyeq C_1 \mid \dots \mid C_n$
⑦
④
$\forall i.\ \mathsf{Det}(C_i)$
$S_1 \mid \dots \mid S_n \preccurlyeq C_1 \mid \dots \mid C_n$
$\mathsf{NPDRF}(S_1 \mid \dots \mid S_n)$
⑤
⑧
$\forall i.\ \mathsf{R}, \mathsf{G} \vdash S_i \preccurlyeq C_i$
$\mathsf{DRF}(S_1 \parallel \dots \parallel S_n)$
Figure 2. Our basic framework

benign races when used by multiple threads simultaneously (e.g., see Fig. 10 for such an efficient implementation of spin locks). We may view that there is a separate compiler transforming the abstract primitives γ_o to their implementations π_o. But how do we prove the compilation correctness in the case when the target code may contain races?

In addition, our previous discussions all assumed that the source and target programs have sequentially consistent (SC) behaviors. But real-world target machines commonly use relaxed semantics. Although most relaxed models have DRF guarantees saying that DRF programs have SC behaviors in the relaxed semantics, there are no such guarantees for racy programs. It is unclear whether the compilation is still correct when the target code (which may have benign races due to the use of efficient implementations of synchronization primitives) uses relaxed semantics.

***A**:* Although the target assembly program using π_o may contain races, we do know that the assembly program using γ_o is DRF if the source program using γ_o is DRF and the compilation is DRF-preserving. We propose a compositional simulation $\pi_o \preccurlyeq^{\mathrm{o}} \gamma_o$, which ensures a strengthened DRF-guarantee theorem for the x86-TSO model. It says, the x86-TSO program using π_o behaves the same as the program using γ_o in the SC semantics if the latter is DRF.

2.4 Frameworks and Key Semantics Components

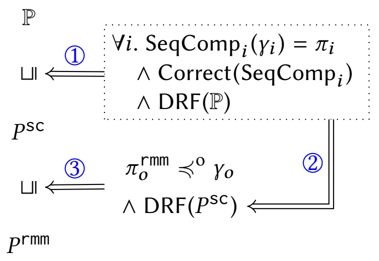
Figure 2 shows the semantics components and proof steps in our basic framework for DRF and SC target code. The goal of our compilation correctness proof is to show the semantics preservation (i.e., $S_1 \| \dots \| S_n \approx C_1 \| \dots \| C_n$ at the top of the figure). This follows the correctness of separate compilation of each module, formulated as $\mathsf{R}, \mathsf{G} \vdash S_i \preccurlyeq C_i$ (the bottom left), which is our new footprint-preserving module-local simulation. We do the proofs in the following steps. Double arrows in the figure are logical implications.

First, we restrict the compilation to source preemptive code that is race-free (i.e., $\mathsf{DRF}(S_1 \parallel \dots \parallel S_n)$ at the right bottom of the figure). Then from the equivalence between preemptive and non-preemptive semantics, we derive ①, the equivalence between $S_1 \parallel \dots \parallel S_n$ and $S_1 \mid \dots \mid S_n$, the latter representing non-preemptive execution of the threads. Similarly, *if we have* $\mathsf{DRF}(C_1 \parallel \dots \parallel C_n)$ (at the top right), we can derive ②. With ① and ②, we can derive the semantics preservation $\approx$ between preemptive programs from $\approx$ between their non-preemptive counterparts.

Second, DRF of the target programs is obtained through the path ⑥,⑦ and ⑧. We define a notion of DRF for non-preemptive programs (called NPDRF), making it equivalent to DRF, from which we derive ⑥ and ⑧. To know $\mathsf{NPDRF}(C_1 \mid \dots \mid C_n)$ from $\mathsf{NPDRF}(S_1 \mid \dots \mid S_n)$, we need a DRF-preserving simulation $\preccurlyeq$ between non-preemptive programs (see ⑦).

Third, by composing our footprint-preserving local simulation, the DRF-preserving simulation $\preccurlyeq$ for non-preemptive whole programs can be derived (step ⑤). Given the downward whole-program simulation, we flip it to get an *upward* one (step ④), with the extra premise that the local execution in each target module is deterministic. Using the simulation in both directions we derive the equivalence (step ③).

$$\mathbb{P} \overset{\text{def}}{=} \textbf{let } \{\gamma_1, \dots, \gamma_m, \gamma_o\} \textbf{ in } \mathsf{f}_1 \parallel \dots \parallel \mathsf{f}_n$$
$$P^{\mathsf{sc}} \overset{\text{def}}{=} \textbf{let } \{\pi_1^{\mathsf{sc}}, \dots, \pi_m^{\mathsf{sc}}, \gamma_o\} \textbf{ in } \mathsf{f}_1 \parallel \dots \parallel \mathsf{f}_n$$
$$P^{\mathsf{rmm}} \overset{\text{def}}{=} \textbf{let } \{\pi_1^{\mathsf{rmm}}, \dots, \pi_m^{\mathsf{rmm}}, \pi_o^{\mathsf{rmm}}\} \textbf{ in } \mathsf{f}_1 \parallel \dots \parallel \mathsf{f}_n$$


$\mathbb{P}$
①
$\sqsupseteq$
$\forall i.\ \mathsf{SeqComp}_i(\gamma_i) = \pi_i$
$\wedge\ \mathsf{Correct}(\mathsf{SeqComp}_i)$
$\wedge\ \mathsf{DRF}(\mathbb{P})$
P^{sc}
②
③
$\sqsupseteq$
$\pi_o^{\mathsf{rmm}} \preccurlyeq^{\mathrm{o}} \gamma_o$
$\wedge\ \mathsf{DRF}(P^{\mathsf{sc}})$
P^{rmm}
Figure 3. The extended framework

The extended framework. Figure 3 shows our ideas for adapting the results of Fig. 2 to relaxed target models and to allow the target programs to contain confined benign races. The goal of the compilation correctness proof is still to show the refinement $\mathbb{P} \sqsupseteq P^{\mathsf{rmm}}$. Here the source $\mathbb{P}$ contains a library module γ_o of the abstract synchronization primitives, as well as normal client modules $\gamma_1, \dots, \gamma_m$. Threads in $\mathbb{P}$ can call functions in these modules. In the target code P^{rmm}, each client module γ_i is compiled to π_i using a sequential compiler $\mathsf{SeqComp}_i$. The library module γ_o is implemented as π_o, which may contain benign races. The superscript $\mathtt{rmm}$ indicates the use of relaxed memory models.

To prove the refinement, we first apply the results of Fig. 2 and prove $\mathbb{P} \sqsupseteq P^{\mathsf{sc}}$ assuming $\mathsf{DRF}(\mathbb{P})$ and correctness of each $\mathsf{SeqComp}_i$ (step ① in Fig. 3). Here the superscript $\mathtt{sc}$ means the use of SC assembly semantics. Each client module π_i in P^{sc} has the same code as in P^{rmm} but uses the SC semantics. Note that P^{sc} still uses the abstract library γ_o rather than its racy implementation π_o. We can view that at this step the library is compiled by an identity transformation.

(Entry)	$\mathtt{f}$	$\in$	String
(Prog)	$P, \mathbb{P}$	$::=$	$\mathbf{let}\ \Pi\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n$
(GEnv)	ge	$\in$	$\mathit{Addr} \rightharpoonup_{\mathsf{fin}} \mathit{Val}$
(MdSet)	Π, Γ	$::=$	$\{(tl_1, ge_1, \pi_1), \ldots, (tl_m, ge_m, \pi_m)\}$
(Lang)	tl, sl	$::=$	$(\mathit{Module}, \mathit{Core}, \mathsf{InitCore}, \longmapsto)$
(Module)	π, γ	$::=$	$\ldots$
(Core)	$\kappa, \Bbbk$	$::=$	$\ldots$
$\mathsf{InitCore}$	$\in$	$\mathit{Module} \to \mathit{Entry} \rightharpoonup \mathit{Core}$	
$\longmapsto$	$\in$	$\mathit{FList} \times (\mathit{Core} \times \mathit{State}) \to \mathcal{P}((\mathit{Msg} \times \mathit{FtPrt}) \times ((\mathit{Core} \times \mathit{State}) \cup \mathbf{abort}))$	
(ThrdID)	$\mathtt{t}$	$\in$	$\mathbb{N}$
(Addr)	l	$::=$	$\ldots$
(Val)	v	$::=$	$l \mid \ldots$
(FList)	$F, \mathbb{F}$	$\in$	$\mathcal{P}^{\omega}(\mathit{Addr})$
(State)	σ, Σ	$\in$	$\mathit{Addr} \rightharpoonup_{\mathsf{fin}} \mathit{Val}$
(FtPrt)	δ, Δ	$::=$	(rs, ws) where $rs, ws \in \mathcal{P}(\mathit{Addr})$
(Msg)	ι	$::=$	$\tau \mid e \mid \mathtt{ret} \mid \mathsf{EntAtom} \mid \mathsf{ExtAtom}$
(Event)	e	$::=$	$\ldots$
(Config)	ϕ, Φ	$::=$	$(\kappa, \sigma) \mid \mathbf{abort}$

Figure 4. The abstract concurrent language

Figure 2 also shows DRF preservation, so we know $\mathsf{DRF}(P^{\mathsf{sc}})$ given $\mathsf{DRF}(\mathbb{P})$ (step ② in Fig. 3). The last step ③ is our strengthened DRF-guarantee theorem. We require a compositional simulation relation $\pi_o^{\mathsf{rmm}} \preccurlyeq^{\mathsf{o}} \gamma_o$ holds, saying that π_o in the relaxed semantics indeed implements γ_o.

The approach not only supports racy implementations of synchronization primitives, but also applies in more general cases when π_o is a racy implementation of a general concurrent object such as a stack or a queue. For instance, π_o could be the Treiber stack implementation [30], and then γ_o could be an atomic abstract stack. Then $\pi_o^{\mathsf{rmm}} \preccurlyeq^{\mathsf{o}} \gamma_o$ can be viewed as a correctness criterion of the object implementation in a relaxed model. It ensures a kind of "contextual refinement", i.e., any client threads using π_o (in relaxed semantics) generate no more observable behaviors than using γ_o instead (in SC semantics), as long as the clients using γ_o is DRF.

Although we expect the approach is general enough to work for different relaxed machine models, so far we have only proved the result for `x86-TSO`, as shown in Sec. 7.3.

Note that the notations used here are simplified ones to give a semi-formal overview of the key ideas. We may use different notations in the formal development below.

3 The Language and Semantics

3.1 The Abstract Language

Figure 4 shows the syntax and the state model of an abstract language for preemptive concurrent programming. A program P consists of n threads, each with an entry $\mathtt{f}$ that labels a code segment in a module in Π. In high-level languages such as Clight, $\mathtt{f}$ is usually the name of a function in a module. A module declaration in Π is a triple consisting of the language declaration tl, the global environment ge, and the

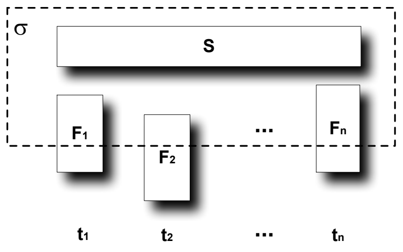


Figure 5. The state model

code π. Here ge contains the global variables declared in the module. It is a finite partial mapping from a global variable's address to its initial value.

The abstract module language tl is defined as a tuple $(\mathit{Module}, \mathit{Core}, \mathsf{InitCore}, \longmapsto)$, whose components can be instantiated for different concrete languages. *Module* describes the syntax of programs. As in Compositional CompCert [29], *Core* is the set of internal "core" states κ, such as control continuations or register files. The function $\mathsf{InitCore}$ returns an initial "core" state κ whenever a thread is created or an external function call is made. In this paper we mainly focus on threads as different modules, and omit the external calls to simplify the presentation. *We do support external calls in our Coq implementation in the same way as in Compositional CompCert.* The labelled transition "$\longmapsto$" models the local execution of a module, which we explain below. We use $\mathcal{P}(S)$ to represent the powerset of the set S.

Module-local semantics. The local execution step inside a module is in the form of $F \vdash (\kappa, \sigma) \overset{\iota}{\underset{\delta}{\longmapsto}} (\kappa', \sigma')$. The *global* memory state σ is a finite partial mapping from memory addresses to values.[2] Each step is also labeled with a message ι and a footprint δ.

Each module also has a *free list* F, an *infinite* set of memory addresses. It models the preserved space for allocating local stack frames. Initially we require $F \cap \mathsf{dom}(\sigma) = \emptyset$, and the only memory accessible by the module is the statically allocated global variables declared in the ge of all modules, represented as the *shared* part S in Fig. 5. The local execution of a module may allocate memory from its F,[3] which enlarges the state σ. So "$F - \mathsf{dom}(\sigma)$" is the set of free addresses, depicted in Fig. 5 as the part outside of the boundary of σ. The memory allocated from F is exclusively owned by this module. F for different modules must be *disjoint* (see the Load rule in Fig. 7).

The messages ι contain information about the module-local steps. Here we only consider externally observable

[2]In our Coq implementation, we use the more concrete CompCert's block-based memory model, where memory addresses l are instantiated as pairs of block IDs and offsets. This allows us to reuse the CompCert code.

[3]The readers should not confuse the allocation from F with dynamic heap memory allocation like `malloc`. The former is for allocation of stack frames only. For the latter, we assume there is a designated module implementing `malloc`, whose free memory blocks are pointed to by a global variable in its ge. Therefore, these memory blocks are in S in Fig. 5, but not in F.

events e (such as outputs), termination of threads (ret), and the beginning and the end of *atomic blocks* (EntAtom and ExtAtom). Any other steps are silent, labeled with a special symbol τ. The label τ is often omitted for cleaner presentation. Atomic blocks are the language constructs to ensure sequential execution of code blocks inside them. They can be implemented differently in different module languages. The messages define the protocols of communications with the global whole-program semantics (presented below).

The footprint δ is a pair (rs, ws) of the read set and write set of memory locations in this step.[4] We write emp for the footprint where both sets are empty.

Note we use two sets of symbols to distinguish the source and the target level notations. Symbols such as $\mathbb{P}$, $\mathbb{F}$, $\Bbbk$, Σ, Γ and γ are used for the source. Their counterparts, P, F, κ, σ, Π and π, are for the target.

Well-defined languages. Although the abstract module language tl can be instantiated with different real languages, the instantiation needs to satisfy certain basic requirements. We formulate these requirements in Def. 1. It gives us an extensional interpretation of footprints.

Definition 1 (Well-Defined Languages). $\mathsf{wd}(tl)$ iff , for any execution step $F \vdash (\kappa, \sigma) \overset{\iota}{\underset{\delta}{\longmapsto}} (\kappa', \sigma')$ in this language, all of the following hold (some auxiliary definitions are in Fig. 6):

(1) $\mathsf{forward}(\sigma, \sigma')$;
(2) $\mathsf{LEffect}(\sigma, \sigma', \delta, F)$;
(3) For any σ_1, if $\mathsf{LEqPre}(\sigma, \sigma_1, \delta, F)$, then there exists σ_1' such that $F \vdash (\kappa, \sigma_1) \overset{\iota}{\underset{\delta}{\longmapsto}} (\kappa', \sigma_1')$ and $\mathsf{LEqPost}(\sigma', \sigma_1', \delta, F)$.
(4) Let $\delta_0 = \bigcup\{\delta \mid \exists \kappa', \sigma'.\, F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}} (\kappa', \sigma')\}$. For any σ_1, if $\mathsf{LEqPre}(\sigma, \sigma_1, \delta_0, F)$, then for any $\kappa_1', \sigma_1', \iota_1, \delta_1$, $F \vdash (\kappa, \sigma_1) \overset{\iota_1}{\underset{\delta_1}{\longmapsto}} (\kappa_1', \sigma_1') \implies \exists \sigma'.\, F \vdash (\kappa, \sigma) \overset{\iota_1}{\underset{\delta_1}{\longmapsto}} (\kappa_1', \sigma')$.

It requires that a step may enlarge the memory domain but cannot reduce it (Item (1)), and the additional memory should be allocated from F and included in the write set (Item (2)). Item (2) also requires that the memory out of the write set should keep unchanged. Item (3) says that the memory updates and allocation only depend on the memory content in the read set, the availability of the memory cells in the write set, and the set of memory locations already allocated from F. Item (4) requires that the non-determinism of each step is not affected by memory contents outside of all the possible read sets in δ_0. Here we do not relate σ_1' and σ', which can be derived from Item (3).

We have proved in Coq that some real languages satisfy wd, including Clight, Cminor, and x86 assembly [13].

$$
\begin{array}{rcl}
\sigma \overset{rs}{=\!=} \sigma' & \text{iff} & \forall l \in rs.\ l \notin (\mathsf{dom}(\sigma) \cup \mathsf{dom}(\sigma')) \vee \\
& & l \in (\mathsf{dom}(\sigma) \cap \mathsf{dom}(\sigma')) \wedge \sigma(l) = \sigma'(l) \\
\delta \subseteq \delta' & \text{iff} & (\delta.rs \subseteq \delta'.rs) \wedge (\delta.ws \subseteq \delta'.ws) \\
\delta \cup \delta' & \overset{\text{def}}{=} & (\delta.rs \cup \delta'.rs,\ \delta.ws \cup \delta'.ws)
\end{array}
$$

$$\mathsf{forward}(\sigma, \sigma') \text{ iff } (\mathsf{dom}(\sigma) \subseteq \mathsf{dom}(\sigma'))$$

$$
\begin{array}{l}
\mathsf{LEqPre}(\sigma_1, \sigma_2, \delta, F) \quad \text{iff} \\
\quad \sigma_1 \overset{\delta.rs}{=\!=\!=} \sigma_2 \wedge (\mathsf{dom}(\sigma_1) \cap \delta.ws) = (\mathsf{dom}(\sigma_2) \cap \delta.ws) \\
\qquad \wedge (\mathsf{dom}(\sigma_1) \cap F) = (\mathsf{dom}(\sigma_2) \cap F) \\
\mathsf{LEqPost}(\sigma_1, \sigma_2, \delta, F) \quad \text{iff} \\
\quad \sigma_1 \overset{\delta.ws}{=\!=\!=} \sigma_2 \wedge\ (\mathsf{dom}(\sigma_1) \cap F) = (\mathsf{dom}(\sigma_2) \cap F) \\
\mathsf{LEffect}(\sigma_1, \sigma_2, \delta, F) \quad \text{iff} \\
\quad \sigma_1 \overset{\mathsf{dom}(\sigma_1) - \delta.ws}{=\!=\!=\!=\!=\!=} \sigma_2 \wedge (\mathsf{dom}(\sigma_2) - \mathsf{dom}(\sigma_1)) \subseteq (\delta.ws \cap F)
\end{array}
$$

Figure 6. Auxiliary definitions about states and footprints

3.2 The Global Preemptive Semantics

Figure 7 shows the global states and selected global semantics rules to model the interaction between modules. The world W consists of the thread pool T, the ID $\mathtt{t}$ of the current thread, a bit d indicating whether the current thread is in an atomic block or not, and the memory state σ. The thread pool T maps a thread ID to a triple recording the module language tl, the free list F, and the current core state κ.[5]

The Load rule in Fig. 7 shows the initialization of the world from the program. The memory σ is initialized as $\mathsf{GE}(\Pi)$, the union of ge from all the modules. The union is defined only if all the ge's are compatible. The rule also requires that σ contain no wild pointers. This requirement is formalized as $\mathsf{closed}(\sigma)$ in Fig. 7, which says the addresses stored in σ must be also in $\mathsf{dom}(\sigma)$.

Global transitions of W are also labeled with footprints δ and messages o. Each global step executes the module locally and processes the message of the local transition. The EntAt and ExtAt rules correspond to the entry and exit of atomic blocks, respectively. The flag d is flipped in the two steps. Since context-switch can be done only when d is 0, as required by the Switch rule below, we know a thread in its atomic block cannot be preempted. The Switch rule shows that context-switch can occur at any program point outside of atomic blocks ($d = 0$).

Below we write $F \vdash \phi \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{*} \phi'$ for zero or multiple silent steps, where δ is the accumulation of the footprint of each step. $F \vdash \phi \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{+} \phi'$ represents at least one step. Similar notations are used for global steps. Also $W \Rightarrow^{*} W'$ is for zero or multiple steps that either are silent or produce sw events.

Event-trace refinement and equivalence. An externally observable event trace $\mathcal{B}$ is a finite or infinite sequence of external events e, and may end with a termination marker

[4] In our Coq code, a footprint contains two additional fields *cmps* and *frees* for operations that observe and modify memory permissions. They allow footprints to capture memory operations more precisely, but they are orthogonal to our main ideas for supporting concurrency and hence merged into *rs* and *ws* in the paper to simplify the presentation.

[5] As we mentioned, our Coq implementation supports external function calls, so T actually maps a thread ID to a *stack* of triples (tl, F, κ). The Coq code also contains additional semantics rules for external calls. The formalization reuses the definitions in Compositional CompCert.

$$
\begin{array}{rrcl@{\qquad}rrcl}
(\mathit{World}) & W, \mathbb{W} & ::= & (T, \mathtt{t}, d, \sigma) & (\mathit{AtomBit}) & d & ::= & 0 \mid 1 \\
(\mathit{NPWorld}) & \widehat{W}, \widehat{\mathbb{W}} & ::= & (T, \mathtt{t}, \mathbb{d}, \sigma) & (\mathit{GMsg}) & o & ::= & \tau \mid e \mid \mathsf{sw} \\
(\mathit{AtomBits}) & \mathbb{d} & ::= & \{\mathtt{t}_1 \leadsto d_1, \ldots, \mathtt{t}_n \leadsto d_n\} \\
(\mathit{ThrdPool}) & T, \mathbb{T} & ::= & \{\mathtt{t}_1 \leadsto (tl_1, F_1, \kappa_1), \ldots, \mathtt{t}_n \leadsto (tl_1, F_n, \kappa_n)\}
\end{array}
$$

$$
\mathsf{GE}(\{(tl_1, ge_1, \pi_1), \ldots, (tl_m, ge_m, \pi_m)\}) \overset{\text{def}}{=}
\begin{cases}
\bigcup_{i=1}^{m} ge_i, & \text{if } \forall i, j.\ ge_i \xlongequal{\mathsf{dom}(ge_i) \cap \mathsf{dom}(ge_j)} ge_j \\
\text{undefined}, & \text{otherwise}
\end{cases}
$$

$$
\begin{array}{lll}
\mathsf{closed}(S, \sigma) & \text{iff} & \forall l, l'.\ l \in S \wedge l' = \sigma(l) \implies l' \in S \\
\mathsf{closed}(\sigma) & \text{iff} & \mathsf{closed}(\mathsf{dom}(\sigma), \sigma)
\end{array}
$$

$$
\frac{\begin{array}{l}
\text{for all } i \text{ and } j \text{ in } \{1, \ldots, n\}, \text{ and } i \neq j: \\
\qquad F_i \cap F_j = \emptyset \quad \mathsf{dom}(\sigma) \cap F_i = \emptyset \\
\qquad tl_i.\mathsf{InitCore}(\pi_i, \mathtt{f}_i) = \kappa_i, \ \text{ where } (tl_i, ge_i, \pi_i) \in \Pi \\
T = \{1 \leadsto (tl_1, F_1, \kappa_1), \ldots, n \leadsto (tl_n, F_n, \kappa_n)\} \\
\mathtt{t} \in \mathsf{dom}(T) \qquad \sigma = \mathsf{GE}(\Pi) \qquad \mathsf{closed}(\sigma)
\end{array}}{\textbf{let } \Pi \textbf{ in } \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n \overset{load}{\Longrightarrow} (T, \mathtt{t}, 0, \sigma)}\ \textsf{Load}
$$

$$
\frac{T(\mathtt{t}) = (tl, F, \kappa) \quad F \vdash (\kappa, \sigma) \underset{\delta}{\overset{\tau}{\longmapsto}} (\kappa', \sigma')}{(T, \mathtt{t}, d, \sigma) \underset{\delta}{\overset{\tau}{\Rightarrow}} (T\{\mathtt{t} \leadsto (tl, F, \kappa')\}, \mathtt{t}, d, \sigma')}\ \tau\text{-step}
$$

$$
\frac{T(\mathtt{t}) = (tl, F, \kappa) \quad F \vdash (\kappa, \sigma) \xmapsto[\mathsf{emp}]{\mathsf{EntAtom}} (\kappa', \sigma)}{(T, \mathtt{t}, 0, \sigma) \xRightarrow[\mathsf{emp}]{\tau} (T\{\mathtt{t} \leadsto (tl, F, \kappa')\}, \mathtt{t}, 1, \sigma)}\ \textsf{EntAt}
$$

$$
\frac{T(\mathtt{t}) = (tl, F, \kappa) \quad F \vdash (\kappa, \sigma) \xmapsto[\mathsf{emp}]{\mathsf{ExtAtom}} (\kappa', \sigma)}{(T, \mathtt{t}, 1, \sigma) \xRightarrow[\mathsf{emp}]{\tau} (T\{\mathtt{t} \leadsto (tl, F, \kappa')\}, \mathtt{t}, 0, \sigma)}\ \textsf{ExtAt}
$$

$$
\frac{\mathtt{t}' \in \mathsf{dom}(T)}{(T, \mathtt{t}, 0, \sigma) \xRightarrow[\mathsf{emp}]{\mathsf{sw}} (T, \mathtt{t}', 0, \sigma)}\ \textsf{Switch}
$$

$$
\frac{\begin{array}{c}
T(\mathtt{t}) = (tl, F, \kappa) \quad \mathbb{d}(\mathtt{t}) = 0 \quad \mathtt{t}' \in \mathsf{dom}(T) \\
F \vdash (\kappa, \sigma) \xmapsto[\mathsf{emp}]{\mathsf{EntAtom}} (\kappa', \sigma) \quad T' = T\{\mathtt{t} \leadsto (tl, F, \kappa')\}
\end{array}}{(T, \mathtt{t}, \mathbb{d}, \sigma) :\xRightarrow[\mathsf{emp}]{\mathsf{sw}} (T', \mathtt{t}', \mathbb{d}\{\mathtt{t} \leadsto 1\}, \sigma)}\ \textsf{EntAt}_{\mathsf{np}}
$$

$$
\frac{\begin{array}{c}
T(\mathtt{t}) = (tl, F, \kappa) \quad \mathbb{d}(\mathtt{t}) = 1 \quad \mathtt{t}' \in \mathsf{dom}(T) \\
F \vdash (\kappa, \sigma) \xmapsto[\mathsf{emp}]{\mathsf{ExtAtom}} (\kappa', \sigma) \quad T' = T\{\mathtt{t} \leadsto (tl, F, \kappa')\}
\end{array}}{(T, \mathtt{t}, \mathbb{d}, \sigma) :\xRightarrow[\mathsf{emp}]{\mathsf{sw}} (T', \mathtt{t}', \mathbb{d}\{\mathtt{t} \leadsto 0\}, \sigma)}\ \textsf{ExtAt}_{\mathsf{np}}
$$

Figure 7. Preemptive and non-preemptive global semantics

done or an abortion marker **abort**. Following the definition in CompCert, we use $P \sqsubseteq \mathbb{P}$ to represent the event-trace refinement, and $P \approx \mathbb{P}$ for equivalence.

3.3 The Non-Preemptive Semantics

A key step in our framework is to reduce the semantics preservation under the preemptive semantics to the semantics preservation in non-preemptive semantics. We write **let** Π **in** $\mathtt{f}_1 | \ldots | \mathtt{f}_n$ or $\hat{P}$ for the program with non-preemptive semantics, to distinguish it from the preemptive concurrency. The global world $\widehat{W}$ is defined similarly as the preemptive world W, except that $\widehat{W}$ keeps an atomic bit map $\mathbb{d}$ recording whether each thread's next step is inside an atomic block. We need to record the atomic bits of all threads because the context switch may occur when a thread just enters an atomic block.

The last two rules in Fig. 7 define the non-preemptive global steps $\widehat{W} :\underset{\delta}{\overset{o}{\Rightarrow}} \widehat{W}'$. More rules are shown in Appendix A. There is no rule like Switch of the preemptive semantics, since context-switch occurs only at synchronization points. $\mathsf{EntAt}_{\mathsf{np}}$ and $\mathsf{ExtAt}_{\mathsf{np}}$ execute one step of the current thread $\mathtt{t}$, and then non-deterministically switch to a thread $\mathtt{t}'$. The corresponding global steps produce the sw events.

4 The Footprint-Preserving Simulation

In this section, we define a *module-local* simulation as the correctness obligation of each module's compilation, which is compositional and preserves footprints, allowing us to derive a whole-program simulation that preserves DRF.

Footprint consistency. As in CompCert, the simulation requires that the source and the target generate the same external events. In addition, it also requires that the target has the same or smaller footprints than the source, which is important to ensure DRF-preservation. Recall that the memory accessible by a thread $\mathtt{t}_i$ consists of two parts, the *shared* memory $\mathbb{S}$ and the local memory allocated from $\mathbb{F}_i$, as shown in Fig. 5. DRF informally requires that the threads never have conflicting accesses to the memory in $\mathbb{S}$ at the same time.

We introduce the triple μ below to record the key information about the shared memory at the source and the target.

$$\mu \overset{\text{def}}{=} (\mathbb{S}, S, f), \text{ where } \mathbb{S}, S \in \mathcal{P}(\mathit{Addr}) \text{ and } f \in \mathit{Addr} \rightharpoonup \mathit{Addr}.$$

Here $\mathbb{S}$ and S specify the shared memory locations at the source and the target respectively. The partial mapping f maps locations at the source level to those at the target. We require μ to be well-formed, defined as $\mathsf{wf}(\mu)$ in Fig. 8.

Then, given footprints Δ and δ, we define their consistency with respect to μ as $\mathsf{FPmatch}(\mu, \Delta, \delta)$ in Fig. 8. It says the *shared* locations in δ must be contained in Δ, modulo the mapping $\mu.f$. We only consider the shared locations in $\mu.S$ because accesses of local memory would *not* cause races. The shared locations in $\delta.rs$ of the target module are allowed to come from $\Delta.ws$ of the source, since transforming a write to a read would not introduce more races.

Rely/guarantee conditions. We use rely and guarantee conditions to specify interaction between modules. They need to enforce the view of accessibility of shared and local memory in Fig. 5. More specifically, a module expects others to keep its local memory (in $\mathbb{F}$) intact. In addition, although other modules may update the shared memory $\mathbb{S}$, they must preserve certain properties of $\mathbb{S}$. One such property is $\mathsf{closed}(\mathbb{S}, \Sigma)$

$$
\begin{array}{l}
f\{\mathbb{S}\} \overset{\text{def}}{=} \{l' \mid \exists l.\ l \in \mathbb{S} \wedge f(l) = l'\} \\
f|_{\mathbb{S}} \overset{\text{def}}{=} \{(l, f(l)) \mid l \in (\mathbb{S} \cap \mathsf{dom}(f))\} \\
\mathsf{wf}(\mu) \ \text{ iff } \ \mathsf{injective}(\mu.f) \wedge \mathsf{dom}(\mu.f) = \mu.\mathbb{S} \wedge \mu.f\{\mu.\mathbb{S}\} = \mu.S \\
\mathsf{FPmatch}(\mu, \Delta, \delta) \ \text{ iff } \ (\delta.rs \cap \mu.S \subseteq \mu.f\{\Delta.rs \cup \Delta.ws\}) \\
\qquad\qquad\qquad\qquad \wedge (\delta.ws \cap \mu.S \subseteq \mu.f\{\Delta.ws\}) \\
\widehat{f}(v) \overset{\text{def}}{=} \begin{cases} v, & \text{if } v \notin Addr \\ f(v), & \text{if } v \in Addr \wedge v \in \mathsf{dom}(f) \\ \text{undefined}, & \text{otherwise} \end{cases} \\
\mathsf{Inv}(f, \Sigma, \sigma) \ \text{ iff } \ \forall l, l'.\ l \in \mathsf{dom}(\Sigma) \wedge f(l) = l' \\
\qquad\qquad\qquad \Longrightarrow l' \in \mathsf{dom}(\sigma) \wedge \widehat{f}(\Sigma(l)) = \sigma(l') \\
\mathsf{HG}(\Delta, \Sigma, \mathbb{F}, \mathbb{S}) \ \text{ iff } \ \Delta \subseteq (\mathbb{F} \cup \mathbb{S}) \wedge \mathsf{closed}(\mathbb{S}, \Sigma) \\
\mathsf{LG}(\mu, (\delta, \sigma, F), (\Delta, \Sigma)) \ \text{ iff } \ \delta \subseteq (F \cup \mu.S) \wedge \mathsf{closed}(\mu.S, \sigma) \\
\qquad\qquad\qquad \wedge \mathsf{FPmatch}(\mu, \Delta, \delta) \wedge \mathsf{Inv}(\mu.f, \Sigma, \sigma) \\
\mathsf{R}(\Sigma, \Sigma', \mathbb{F}, \mathbb{S}) \ \text{ iff } \ (\Sigma \overset{\mathbb{F}}{=\!=} \Sigma') \wedge \mathsf{closed}(\mathbb{S}, \Sigma') \wedge \mathsf{forward}(\Sigma, \Sigma') \\
\mathsf{Rely}(\mu, (\Sigma, \Sigma', \mathbb{F}), (\sigma, \sigma', F)) \ \text{ iff } \ \mathsf{R}(\Sigma, \Sigma', \mathbb{F}, \mu.\mathbb{S}) \wedge \mathsf{R}(\sigma, \sigma', F, \mu.S) \\
\qquad\qquad\qquad \wedge \mathsf{Inv}(\mu.f, \Sigma', \sigma') \\
\lfloor\varphi\rfloor(ge) \overset{\text{def}}{=} \begin{cases} \{(\varphi(l), \widehat{\varphi}(v)) \mid (l, v) \in ge\}, \\ \qquad \text{if } (\mathsf{dom}(ge) \cup (\mathsf{range}(ge) \cap Addr)) \subseteq \mathsf{dom}(\varphi) \\ \text{undefined}, \qquad \text{otherwise} \end{cases} \\
\mathsf{initM}(\varphi, ge, \Sigma, \sigma) \ \text{ iff } \ ge \subseteq \Sigma \wedge \mathsf{closed}(\Sigma) \\
\qquad\qquad\qquad \wedge \mathsf{dom}(\sigma) = \varphi\{\mathsf{dom}(\Sigma)\} \wedge \mathsf{Inv}(\varphi, \Sigma, \sigma)
\end{array}
$$

Figure 8. Footprint matching and rely/guarantee conditions

(defined in Fig. 7), which ensures $\mathbb{S}$ cannot contain memory pointers pointing to local memory cells in any $\mathbb{F}_i$. Otherwise a thread $\mathtt{t}_j$ can update the memory in $\mathbb{F}_i$ by tracing these pointers. [6] Also the invariant Inv should be preserved, which relates the contents of the corresponding memory locations in Σ and σ. It expresses the same thing as memory injection in CompCert [6]. We encode these requirements in the rely condition Rely in Fig. 8, and define the guarantee conditions HG and LG correspondingly.

The simulation. Below we define $(sl, ge, \gamma) \preccurlyeq_\varphi (tl, ge', \pi)$ to relate the *non-preemptive* executions of the source module (sl, ge, γ) and the target one (tl, ge', π). The *injective function* φ maps source addresses to the target ones.

Definition 2 (Module-Local Downward Simulation)**.**
$(sl, ge, \gamma) \preccurlyeq_\varphi (tl, ge', \pi)$ iff

1. $\lfloor\varphi\rfloor(ge) = ge'$; and
2. for all $\mathtt{f}, \Bbbk, \Sigma, \sigma, \mathbb{F}, F$, and $\mu = (\mathsf{dom}(\Sigma), \mathsf{dom}(\sigma), \varphi|_{\mathsf{dom}(\Sigma)})$, if $sl.\mathsf{InitCore}(\gamma, \mathtt{f}) = \Bbbk$, $\mathbb{F} \cap \mathsf{dom}(\Sigma) = F \cap \mathsf{dom}(\sigma) = \emptyset$, and $\mathsf{initM}(\varphi, ge, \Sigma, \sigma)$, then there exist $i \in \mathtt{index}$ and κ such that $tl.\mathsf{InitCore}(\pi, \mathtt{f}) = \kappa$, and
$$(\mathbb{F}, (\Bbbk, \Sigma), \mathtt{emp}) \preccurlyeq^i_\mu (F, (\kappa, \sigma), \mathtt{emp}),$$
where $(\mathbb{F}, (\Bbbk, \Sigma), \Delta) \preccurlyeq^i_\mu (F, (\kappa, \sigma), \delta)$ is defined in Def. 3.

It says that, starting from some core states $\Bbbk$ and κ, with any states (Σ and σ) and free lists ($\mathbb{F}$ and F) satisfying some initial constraints, we have the simulation $(\mathbb{F}, (\Bbbk, \Sigma), \mathtt{emp}) \preccurlyeq^i_\mu (F, (\kappa, \sigma), \mathtt{emp})$ defined in Def. 3. Here ge and ge' must be related through $\lfloor\varphi\rfloor$ (see Fig. 8). The initial states Σ and σ also need to be related with ge and φ through initM.

Definition 3. $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0) \preccurlyeq^i_\mu (F, (\kappa, \sigma), \delta_0)$ is the largest relation such that, whenever $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0) \preccurlyeq^i_\mu (F, (\kappa, \sigma), \delta_0)$, then the following are true:

1. for all $\Bbbk', \Sigma'$ and Δ, if $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\tau}{\underset{\Delta}{\longmapsto}} (\Bbbk', \Sigma')$ and $(\Delta_0 \cup \Delta) \subseteq (\mathbb{F} \cup \mu.\mathbb{S})$, then one of the following holds:
   a. $\exists j < i.\ (\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta) \preccurlyeq^j_\mu (F, (\kappa, \sigma), \delta_0)$, or
   b. there exist κ', σ', δ and j such that:
      i. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{+} (\kappa', \sigma')$;
      ii. $(\delta_0 \cup \delta) \subseteq (F \cup \mu.S)$ and $\mathsf{FPmatch}(\mu, \Delta_0 \cup \Delta, \delta_0 \cup \delta)$; and
      iii. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta) \preccurlyeq^j_\mu (F, (\kappa', \sigma'), \delta_0 \cup \delta)$.
2. for all $\Bbbk'$ and ι, if $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\iota}{\underset{\mathtt{emp}}{\longmapsto}} (\Bbbk', \Sigma)$, $\iota \neq \tau$, and $\mathsf{HG}(\Delta_0, \Sigma, \mathbb{F}, \mu.\mathbb{S})$, there exist κ', δ, σ' and κ'' such that:
   a. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{*} (\kappa', \sigma')$, and $F \vdash (\kappa', \sigma') \overset{\iota}{\underset{\mathtt{emp}}{\longmapsto}} (\kappa'', \sigma')$, and
   b. $\mathsf{LG}(\mu, (\delta_0 \cup \delta, \sigma', F), (\Delta_0, \Sigma))$, and
   c. for all σ'' and Σ', if $\mathsf{Rely}(\mu, (\Sigma, \Sigma', \mathbb{F}), (\sigma', \sigma'', F))$, then there exists j such that
   $$(\mathbb{F}, (\Bbbk', \Sigma'), \mathtt{emp}) \preccurlyeq^j_\mu (F, (\kappa'', \sigma''), \mathtt{emp}).$$

The simulation $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0) \preccurlyeq^i_\mu (F, (\kappa, \sigma), \delta_0)$ carries Δ_0 and δ_0, the footprints accumulated at the source and the target, respectively. The definition follows the diagram in Fig. 1(d). For every τ-step in the source (case 1), if the newly generated footprints and the accumulated Δ_0 are *in scope* (i.e. every location must either be from the freelist space $\mathbb{F}$ of current thread, or from the shared memory $\mu.\mathbb{S}$), then the step corresponds to zero or multiple τ-steps in the target, and the simulation holds over the resulting states with the accumulated footprints and a new index j. We carry the well-founded index to ensure the simulation preserves termination.

If the source step corresponds to at least one target steps (case 1-b), the footprints at the target must also be in scope and be consistent with the source level footprints. The accumulation of footprints allows us to establish FPmatch for compiler optimizations that reorder the instructions.

At the switch points when the source generates a non-silent message ι (case 2), if the footprints and states satisfy the high-level guarantee HG, the target must be able to generate the same ι, and the accumulated footprints and the state satisfy the low-level guarantee LG. We also need to consider the interaction with other modules or threads. For any environment steps satisfying Rely, the simulation must hold over the new states, with some index j and *empty* footprints — Since the effects of the current thread have been

[6] We disallow cross-module escape of pointers pointing to stack-allocated variables. (We still allow the escape within a module and the transfer of dynamically allocated heap data structures.) Our TR [13] presents the framework with the support of stack pointer escape. Adding the support looks relatively orthogonal to our main ideas for supporting concurrency, and we can follow the approach of Compositional CompCert (the part for stack pointer escape).

made visible to the environments at the switch point, we can clear the accumulated footprints. Rely and the guarantees HG and LG are defined in Fig. 8, as explained before.

Note that each case in Def. 3 has prerequisites about the source level footprints (e.g. the footprints are in scope or satisfy HG). We need to prove that these requirements indeed hold at the source level, to make the simulation meaningful instead of being vacuously true. We formalize these requirements separately as ReachClose in Def. 4. It is a simplified version of the *reach-close* concept by Stewart et al. [29] (simplified because we disallow the escape of local stack pointers into the shared memory). The compilation correctness assumes that all the source modules satisfy ReachClose (see Lem. 6 and Def. 11 below).

Definition 4 (Reach Closed Module). $\mathsf{ReachClose}(sl, ge, \gamma)$ iff , for all $\mathsf{f}, \Bbbk, \Sigma, \mathbb{F}$ and $\mathbb{S}$, if $sl.\mathsf{InitCore}(\gamma, \mathsf{f}) = \Bbbk$, $\mathbb{S} = \mathsf{dom}(\Sigma)$, $ge \subseteq \Sigma$, $\mathbb{F} \cap \mathbb{S} = \emptyset$, and $\mathsf{closed}(\mathbb{S}, \Sigma)$, then $\mathsf{RC}(\mathbb{F}, \mathbb{S}, (\Bbbk, \Sigma))$.

Here RC is defined as the largest relation such that, whenever $\mathsf{RC}(\mathbb{F}, \mathbb{S}, (\Bbbk, \Sigma))$, then for all Σ' such that $\mathsf{R}(\Sigma, \Sigma', \mathbb{F}, \mathbb{S})$, and for all $\Bbbk', \Sigma', \Sigma'', \iota$ and Δ such that $\mathbb{F} \vdash (\Bbbk, \Sigma') \overset{\iota}{\underset{\Delta}{\longmapsto}} (\Bbbk', \Sigma'')$, we have $\mathsf{HG}(\Delta, \Sigma'', \mathbb{F}, \mathbb{S})$, and $\mathsf{RC}(\mathbb{F}, \mathbb{S}, (\Bbbk', \Sigma''))$.

The relation $\mathsf{RC}(\mathbb{F}, \mathbb{S}, (\Bbbk, \Sigma))$ essentially says during every step of the execution of $(\Bbbk, \Sigma)$, HG always holds over the resulting footprints Δ and states, even with possible interference from the environment, as long as the environment steps satisfy the rely condition R defined in Fig. 8.

Our simulation is transitive. One can decompose the whole compiler correctness proofs into proofs for individual compilation passes.

Lemma 5 (Transitivity). $\forall sl, sl', tl, \gamma, \gamma', \pi.$ if $(sl, ge, \gamma) \preccurlyeq_{\varphi} (sl', ge', \gamma')$ and $(sl', ge', \gamma') \preccurlyeq_{\varphi'} (tl, ge'', \pi)$, then $(sl, ge, \gamma) \preccurlyeq_{\varphi' \circ \varphi} (tl, ge'', \pi)$.

The proof of Lem. 5 relies on the auxiliary lemmas similar to "memory interpolations" in Compostional CompCert.

Lemma 6 (Compositionality, ⑤ in Fig. 2). For any $\mathsf{f}_1, \ldots, \mathsf{f}_n, \varphi, \Gamma = \{(sl_1, ge_1, \gamma_1), \ldots, (sl_m, ge_m, \gamma_m)\}$, and $\Pi = \{(tl_1, ge'_1, \pi_1), \ldots, (tl_m, ge'_m, \pi_m)\}$, if

$$\forall i \in \{1, \ldots, m\}.\ \mathsf{wd}(sl_i) \land \mathsf{wd}(tl_i) \land \mathsf{ReachClose}(sl_i, ge_i, \gamma_i) \land (sl_i, ge_i, \gamma_i) \preccurlyeq_{\varphi} (tl_i, ge'_i, \pi_i)\ ,$$

then $\quad$ **let** Γ **in** $\mathsf{f}_1 \mid \ldots \mid \mathsf{f}_n \preccurlyeq$ **let** Π **in** $\mathsf{f}_1 \mid \ldots \mid \mathsf{f}_n$.

Lemma 6 shows the compositionality of our simulation. The whole program *downward* simulation $\hat{\mathbb{P}} \preccurlyeq \hat{P}$ relates the non-preemptive execution of the whole source program $\mathbb{P}$ and target program P. The definition is given in Appendix B.

With determinism of the target module language (written as $\mathsf{det}(tl)$), we can flip $\hat{\mathbb{P}} \preccurlyeq \hat{P}$ to derive the upward simulation $\hat{P} \leqslant \hat{\mathbb{P}}$. We give the definition of $\mathsf{det}(tl)$ and the Flip Lemma (④ in Fig. 2) in Appendix B. Lemma 7 shows the non-preemptive global simulation ensures the refinement.

Lemma 7 (Soundness, ③ in Fig. 2). If $\hat{P} \leqslant \hat{\mathbb{P}}$, then $\hat{P} \sqsubseteq \hat{\mathbb{P}}$.

$$\frac{P \overset{load}{\Longrightarrow} W \qquad W \Rightarrow^{*} W' \qquad W' \Longmapsto \mathsf{Race}}{P \Longmapsto \mathsf{Race}}$$

$$\frac{\begin{array}{c}\mathtt{predict}(W, \mathtt{t}_1, (\delta_1, d_1)) \qquad \mathtt{predict}(W, \mathtt{t}_2, (\delta_2, d_2)) \\ \mathtt{t}_1 \neq \mathtt{t}_2 \qquad (\delta_1, d_1) \frown (\delta_2, d_2)\end{array}}{W \Longmapsto \mathsf{Race}}\ \textsf{Race}$$

$$\frac{W = (T, \_, 0, \sigma) \quad T(\mathtt{t}) = (F, \kappa) \quad F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}} (\kappa', \sigma')}{\mathtt{predict}(W, \mathtt{t}, (\delta, 0))}\ \textsf{Predict-0}$$

$$\frac{\begin{array}{c}W = (T, \_, 0, \sigma) \qquad T(\mathtt{t}) = (F, \kappa) \\ F \vdash (\kappa, \sigma) \overset{\mathsf{EntAtom}}{\underset{\mathsf{emp}}{\longmapsto}} (\kappa', \sigma) \quad F \vdash (\kappa', \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{*} (\kappa'', \sigma'')\end{array}}{\mathtt{predict}(W, \mathtt{t}, (\delta, 1))}\ \textsf{Predict-1}$$

Figure 9. Data races in preemptive semantics

5 Data-Race-Freedom

Below we first define the conflict of footprints.

$$\begin{array}{lll} \delta_1 \frown \delta_2 & \text{iff} & (\delta_1.ws \cap \delta_2 \neq \emptyset) \lor (\delta_2.ws \cap \delta_1 \neq \emptyset) \\ (\delta_1, d_1) \frown (\delta_2, d_2) & \text{iff} & (\delta_1 \frown \delta_2) \land (d_1 = 0 \lor d_2 = 0) \end{array}$$

Recall that, when used as a set, δ represents $\delta.rs \cup \delta.ws$. Since we do *not* treat accesses of the same memory location inside atomic blocks as a race, we instrument a footprint δ with the atomic bit d to record whether the footprint is generated inside an atomic block ($d = 1$) or not ($d = 0$). Two instrumented footprints (δ_1, d_1) and (δ_2, d_2) are conflicting if δ_1 and δ_2 are conflicting and at least one of d_1 and d_2 is 0.

We define data races in Fig. 9 for preemptive semantics. In the Race rule, W steps to Race if there are conflicting footprints of two threads predicted from the current configuration through $\mathtt{predict}(W, \mathtt{t}, (\delta, d))$. Then we define $\mathsf{DRF}(P)$:

$$\mathsf{DRF}(P) \ \text{ iff } \ \neg(P \Longmapsto \mathsf{Race})$$

$\mathsf{NPDRF}(\hat{P})$ is defined similarly. We can prove NPDRF is equivalent to DRF (⑥ and ⑧ in Fig. 2). The following Lem. 8 shows the simulation preserves NPDRF. Given the equivalence, we know it also preserves DRF.

Lemma 8 (NPDRF Preservation, ⑦ in Fig. 2). For any $\hat{\mathbb{P}}, \hat{P}$, if $\hat{P} \leqslant \hat{\mathbb{P}}$, and $\mathsf{NPDRF}(\hat{\mathbb{P}})$, then $\mathsf{NPDRF}(\hat{P})$.

Lemma 9 (Semantics Equivalence, ① and ② in Fig. 2). For any $\Pi, \mathsf{f}_1, \ldots, \mathsf{f}_m$, if $\mathsf{DRF}($**let** Π **in** $\mathsf{f}_1 \parallel \ldots \parallel \mathsf{f}_m)$, then **let** Π **in** $\mathsf{f}_1 \mid \ldots \mid \mathsf{f}_m \approx$ **let** Π **in** $\mathsf{f}_1 \parallel \ldots \parallel \mathsf{f}_m$.

6 The Final Theorem

Putting all the previous results together, we are able to prove our final theorem in the basic framework (Fig. 2). We first model a sequential compiler SeqComp as follows:

$$\mathsf{SeqComp} ::= (\mathsf{CodeT}, \varphi)\ , \ \text{ where } \mathsf{CodeT} \in \mathit{Module} \rightharpoonup \mathit{Module}$$

As the key proof obligation, we need to verify that each SeqComp is Correct. The correctness is defined based on our footprint-preserving module-local simulation. Recall that $\lfloor\varphi\rfloor$ is defined in Fig. 8.

Definition 10 (Sequential Compiler Correctness). $\mathsf{Correct}(\mathsf{SeqComp}, sl, tl)$ iff

$$\forall \gamma, \pi, ge, ge'.\ \mathsf{SeqComp.CodeT}(\gamma) = \pi \wedge \lfloor \mathsf{SeqComp}.\varphi \rfloor(ge) = ge' \implies (sl, ge, \gamma) \preccurlyeq_{\mathsf{SeqComp}.\varphi} (tl, ge', \pi)\ .$$

The desired correctness GCorrect (Def. 11) of *concurrent* program compilation is the semantics preservation of whole programs, i.e., every target concurrent program is a refinement of the source. Here all the SeqComp must agree on the transformation φ of global environments (see item 1 below).

Definition 11 (Concurrent Compiler Correctness). $\mathsf{GCorrect}((\mathsf{SeqComp}_1, sl_1, tl_1), \ldots, (\mathsf{SeqComp}_m, sl_m, tl_m))$ iff for any $\varphi, \mathtt{f}_1, \ldots, \mathtt{f}_n, \Gamma = \{(sl_1, ge_1, \gamma_1), \ldots, (sl_m, ge_m, \gamma_m)\}$, and $\Pi = \{(tl_1, ge'_1, \pi_1), \ldots, (tl_m, ge'_m, \pi_m)\}$, if

1. $\forall i \in \{1, \ldots, m\}.\ (\mathsf{SeqComp}_i.\mathsf{CodeT}(\gamma_i) = \pi_i) \wedge \mathsf{injective}(\varphi) \wedge (\mathsf{SeqComp}_i.\varphi = \varphi) \wedge \lfloor\varphi\rfloor(ge_i) = ge'_i$,
2. $\mathsf{Safe}(\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n)$, and $\mathsf{DRF}(\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n)$,
3. $\forall i \in \{1, \ldots, m\}.\ \mathsf{ReachClose}(sl_i, ge_i, \gamma_i)$,

then $\quad \mathbf{let}\ \Pi\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n \sqsubseteq \mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n$.

Our final theorem is then formulated as Thm. 12. It says if a set of sequential compilers are certified to satisfy our correctness obligation Correct, the source and target languages sl_i and tl_i are well-defined, and the target languages are deterministic, then the sequential compilers as a whole is GCorrect for compiling concurrent programs. The proof simply applies the lemmas that correspond to ①-⑧ in Fig. 2.

Theorem 12 (Final Theorem). For any $\mathsf{SeqComp}_1, \ldots, \mathsf{SeqComp}_m, sl_1, \ldots, sl_m, tl_1, \ldots, tl_m$ such that for any $i \in \{1, \ldots, m\}$ we have $\mathsf{wd}(sl_i)$, $\mathsf{wd}(tl_i)$, $\mathsf{det}(tl_i)$, and $\mathsf{Correct}(\mathsf{SeqComp}_i, sl_i, tl_i)$, then

$$\mathsf{GCorrect}((\mathsf{SeqComp}_1, sl_1, tl_1), \ldots, (\mathsf{SeqComp}_m, sl_m, tl_m)).$$

7 The CASCompCert Compiler

We apply our framework to develop CASCompCert. It uses CompCert-3.0.1 [6] for compilation of multi-threaded Clight programs to x86-SC, i.e. x86 with SC semantics. We further extend the framework (as shown in Fig. 3) to support x86-TSO [28] as the target language. The source program we compile consists of multiple sequential Clight threads. Inter-thread synchronization can be achieved through *external calls* to an external module. Since the external synchronization module can be shared by the threads, below we also refer to it as an *object* and the Clight threads as *clients*.

As a tiny example, Fig. 10(a) shows a spin-lock implementation in a simple imperative language which we call CImp. $\langle C \rangle$ is an atomic block, which cannot be interrupted by other threads. The EntAtom and ExtAtom events are generated at the beginning and the end of the atomic block respectively. The command `assert(B)` aborts if B is false.

```
lock(){  r := 0; while(r==0){ <r:=[L]; [L]:=0;> }  }
unlock(){  < r := [L]; assert(r == 0); [L] := 1; > }
```

(a) lock specification γ_{lock}

```
lock:     movl      $L,       %ecx
          movl      $0,       %edx
l_acq:    movl      $1,       %eax
     lock cmpxchgl  %edx,     (%ecx)
          je        enter
spin:     movl      (%ecx),   %ebx
          cmp       $0,       %ebx
          je        spin
          jmp       l_acq
enter:    retl
unlock:   movl      $L,       %eax
          movl      $1,       (%eax)
          retl
```

(b) lock implementation π_{lock}

```
void inc(){
  int32_t tmp;
  lock();
    tmp = x;
    x ++;
  unlock();
  print(tmp);
}
```

(c) a Clight client γ_{C}

Figure 10. Lock as an external module of Clight programs

This source implementation of locks serves as an abstract specification, which is *manually* translated to x86-TSO, as shown in Fig. 10(b). The implementation is similar to the Linux spin-lock implementation (a.k.a. the TTAS lock [11]). To acquire the lock, the `l_acq` block reads and resets the lock bit to 0 atomically by the `lock`-prefixed `cmpxchgl`, which ensures mutual exclusion. The `spin` loop reads the lock bit until it appears to be available (i.e., not 0). Note that the load and store operations in the `spin` and `unlock` blocks are *not* `lock`-prefixed. This optimization introduces benign races. With the external lock module, we can implement a DRF counter `inc` in Clight, as shown in Fig. 10(c). An example whole program $\mathbb{P}$ is $\mathbf{let}\ \{\gamma_{\text{C}}, \gamma_{\texttt{lock}}\}\ \mathbf{in}\ \texttt{inc()} \parallel \texttt{inc()}$.

As shown in Fig. 3, we compile the code in two steps. First, we use CompCert to compile the Clight client to x86-SC, but leave object code (e.g., $\gamma_{\texttt{lock}}$ in Fig. 10) untouched. Second, we transform the resulting x86-SC client code to x86-TSO. Syntactically this is an identity transformation, but the semantics changes. Then we manually transform the source object code to x86-TSO code. We can prove the resulting whole program in x86-TSO preserves the source semantics as long as the x86-TSO object code refines its source.

7.1 Language Instantiations

We need to first instantiate our abstract languages of Fig. 4 with Clight, x86-SC and the intermediate languages introduced in CompCert. We also instantiate it with the simple language CImp for the source object code.

The Clight language. The *Module* in Fig. 4 is instantiated with the same Clight syntax as in CompCert. The core state κ is a pair of a local state c and an index N indicating the position of the next block in the freelist F to be allocated, as shown below. InitCore initializes N to 0. Local transitions of Clight are instrumented with footprints.

$$
\begin{array}{rlll@{\qquad}rlll}
(\mathit{FList}) & F & ::= & b_1 :: b_2 :: \ldots & (\mathit{Block}) & b & \in & \mathbb{N}^+ \\
(\mathit{BIndex}) & N & \in & \mathbb{N} & (\mathit{Core}) & \kappa & ::= & (c, N) \\
(\mathit{Mem}) & \sigma & \in & \mathit{Block} \rightharpoonup_{\text{fin}} (\mathbb{N} \rightharpoonup \mathtt{val}) & & & &
\end{array}
$$

The source language CImp for objects. The instantiation of the abstract language with CImp lets the atomic blocks generate the EntAtom and ExtAtom events. To allow benign races in the target x86-TSO code, we need to ensure partition of the client data and the object data. This can be enforced through the permissions in the CompCert memory model. The client programs can only access memory locations whose permission is not None. Therefore we set the permission of the object data (the memory at the location L in our example in Fig. 10) to None. Also, we require the CImp program can *only* access memory locations with None permission. It aborts if trying to access memory locations whose permissions are *not* None .

7.2 Adapting CompCert

Given the program **let** $\{\gamma_1, \ldots, \gamma_l, \gamma_o\}$ **in** $\mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n$ consisting of Clight modules γ_i and the module γ_o (we omit *sl* and *ge* in the modules to simplify the presentation), the compilation Comp is defined as

$$\mathtt{Comp}(\mathbf{let}\ \{\gamma_1, \ldots, \gamma_l, \gamma_o\}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n) \overset{\text{def}}{=}$$
$$\mathbf{let}\ \{\mathtt{CompCert}(\gamma_1), \ldots, \mathtt{CompCert}(\gamma_l), \mathtt{IdTrans}(\gamma_o)\}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n$$

where CompCert is the adapted compilation consisting of the original CompCert passes, and IdTrans is the identity translation which returns the object module unchanged.

Lemma 13. Correct(CompCert, Clight, x86-SC).

We have proved Lem. 13, i.e. the original CompCert-3.0.1 passes satisfy our Correct in Def. 10. The verified compilation passes (shown in Fig. 11) include all the translation passes and four optimization passes (Tailcall, Renumber, Tunneling and CleanupLabels).[7] Proving other optimization passes would be similar and is left as future work.

We also prove the well-definedness of Clight, x86-SC, and CImp, the determinism of x86-SC and CImp, and the correctness of IdTrans for CImp. Together with our framework's final theorem (Thm. 12), we derive the following result:

Theorem 14 (Correctness with x86-SC backend and obj.). GCorrect((CompCert, Clight, x86-SC), (IdTrans, CImp, CImp)).

To prove Lem. 13, we try to *reuse as much the original CompCert correctness proofs as possible.* We address the following two main challenges in reusing CompCert proofs.

Converting memory layout. Many CompCert lemmas rely on the specific definition of the CompCert memory model, which is different from ours. In CompCert, memory allocations in an execution get *consecutive* natural numbers as block numbers. This fact is used extensively in CompCert's fundamental libraries and its compilation correctness proofs.

[7]The compiler option -g for insertion of debugging information is disabled.

But it does not hold in our model, where each thread has its own freelist F (an infinite sequence of block numbers). Since the F of different threads must be disjoint, we *cannot* make each F an infinite sequence of consecutive natural numbers to directly simulate CompCert.

Our solution is to define a bijection between memories under the two models. As a result, the behaviors of a thread under our model are equivalent to its behaviors under CompCert model, and our module-local simulation can be derived from a simulation based on the CompCert model. This way we reuse most CompCert libraries and compilation proofs without modification.

Footprint preservation. CompCert does not model footprints. Fortunately many of its definitions and lemmas can be slightly modified to support footprint preservation. For instance, Fig. 12 shows a key lemma in the proof of the Selection pass, sel_expr_correct, with our newly-added code highlighted. It says the selected expression must evaluate to a value refined by the Cminor expression. We simply extend the lemma by requiring *the selected expression has smaller footprint while evaluating on related memory.*

7.3 x86-TSO as the Target Language

Now we show how to generate x86-TSO code while preserving the behaviors of the source program. The theorem below shows our final goal. To avoid clutter, below we use *sl* and *tl* to represent $sl_{\mathtt{Clight}}$ and $tl_{\mathtt{x86\text{-}TSO}}$ respectively.

Theorem 15 (Correctness with x86-TSO backend and obj.). For any $\mathtt{f}_1 \ldots \mathtt{f}_n$, $\Gamma = \{(sl, ge_1, \gamma_1), \ldots, (sl, ge_m, \gamma_m), (sl_{\mathtt{CImp}}, ge_o, \gamma_o)\}$, and $\Pi = \{(tl, ge'_1, \pi_1), \ldots, (tl, ge'_m, \pi_m), (tl, ge_o, \pi_o)\}$, if

1. $\forall i \in \{1, \ldots, m\}.\ (\mathtt{CompCert.CodeT}(\gamma_i) = \pi_i) \land \mathsf{injective}(\varphi) \land (\mathtt{CompCert}.\varphi = \varphi) \land \lfloor\varphi\rfloor(ge_i) = ge'_i$,
2. $\mathsf{Safe}(\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n)$ and $\mathsf{DRF}(\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n)$,
3. $\forall i \in \{1, \ldots, m\}.\ \mathsf{ReachClose}(sl, ge_i, \gamma_i)$, and $\mathsf{ReachClose}(sl_{\mathtt{CImp}}, ge_o, \gamma_o)$,
4. $(tl, ge_o, \pi_o) \preccurlyeq^{\mathrm{o}} (sl_{\mathtt{CImp}}, ge_o, \gamma_o)$,

then $\mathbf{let}\ \Pi\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n \sqsubseteq' \mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n$.

Here the premises 1-3 are similar to those required in Def. 11. In addition, the premise 4 requires that the x86-TSO code π_o of the object be simulated by γ_o. The simulation $(tl, ge_o, \pi_o) \preccurlyeq^{\mathrm{o}} (sl_{\mathtt{CImp}}, ge_o, \gamma_o)$ is an extension of Liang and Feng [19] with the support of TSO semantics for the low-level code. Due to space limit, we omit the definition here.

The refinement relation $\sqsubseteq'$ is a weaker version of $\sqsubseteq$ (see Sec. 3.2). It does not preserve termination (the formal definition omitted here). This is because our simulation $\preccurlyeq^{\mathrm{o}}$ for the object code does not preserve termination for now, which we leave as future work.

Theorem 15 can be derived from Thm. 14 (for the compilation from Clight to x86-SC), and from Lem. 16 below, saying the x86-TSO code refines the x86-SC client code and the

$$\begin{tikzcd}
\mathit{Clight} \arrow[r, Rightarrow, "\texttt{Cshmgen}"] & \mathit{C\#minor} \arrow[r, Rightarrow, "\texttt{Cminorgen}"] & \mathit{Cminor} \arrow[r, Rightarrow, "\texttt{Selection}"] & \mathit{CminorSel} \arrow[r, Rightarrow, "\texttt{RTLgen}"] & \mathit{RTL} \arrow[r, Rightarrow, "\texttt{Tailcall, Renumber}"] & \mathit{RTL} \arrow[d, Rightarrow, "\texttt{Allocation}"] \\
\mathit{x86\text{-}SC\ assembly} & \mathit{Mach} \arrow[l, Rightarrow, "\texttt{Asmgen}"'] & \mathit{Linear} \arrow[l, Rightarrow, "\texttt{Stacking}"'] & \mathit{Linear} \arrow[l, Rightarrow, "\texttt{CleanupLabels}"'] & \mathit{LTL} \arrow[l, Rightarrow, "\texttt{Linearize}"'] & \mathit{LTL} \arrow[l, Rightarrow, "\texttt{Tunneling}"']
\end{tikzcd}$$

Figure 11. Proved CompCert compilation passes

```
Lemma sel_expr_correct:
 forall sp e m a v fp, Cminor.eval_expr sge sp e m a v ->
 Cminor.eval_expr_fp sge sp e m a fp ->
 forall e' le m', env_lessdef e e' -> Mem.extends m m' ->
 exists v', exists fp', FP.subset fp' fp /\
 eval_expr_fp tge sp e' m' le (sel_expr a) fp' /\
 eval_expr tge sp e' m' le (sel_expr a) v' /\ Val.lessdef v v'.
```

Figure 12. Coq code example

Compilation passes and framework	Spec		Proof	
	CompCert	Ours	CompCert	Ours
Cshmgen	515	1021	1071	1503
Cminorgen	753	1556	1152	1251
Selection	336	500	647	783
RTLgen	428	543	821	862
Tailcall	173	328	275	405
Renumber	86	245	117	358
Allocation	704	785	1410	1700
Tunneling	131	339	166	475
Linearize	236	371	349	733
CleanupLabels	126	387	161	388
Stacking	730	1038	1108	2135
Asmgen	208	338	571	1128
Compositionality (Lem. 6)		580		2249
DRF preservation (Lem. 8)		358		1142
Semantics equiv. (Lem. 9)		1540		4718
Lifting		813		1795

Figure 13. Lines of code (using coqwc) in Coq

source object code (we use $tl_{\mathtt{sc}}$ and $tl_{\mathtt{tso}}$ as shorter notations for $tl_{\mathtt{x86\text{-}SC}}$ and $tl_{\mathtt{x86\text{-}TSO}}$ respectively).

Lemma 16 (Restore SC semantics for DRF x86 programs)**.** Let $\Pi_{\mathtt{sc}} = \{(tl_{\mathtt{sc}}, ge_1, \pi_1), \ldots, (tl_{\mathtt{sc}}, ge_m, \pi_m), (sl_{\mathtt{CImp}}, ge_o, \gamma_o)\}$, and $\Pi_{\mathtt{tso}} = \{(tl_{\mathtt{tso}}, ge_1, \pi_1), \ldots, (tl_{\mathtt{tso}}, ge_m, \pi_m), (tl_{\mathtt{tso}}, ge_o, \pi_o)\}$. For any $f_1 \ldots f_n$, if

1. $\mathrm{Safe}(\textbf{let } \Pi_{\mathtt{sc}} \textbf{ in } \mathtt{f}_1 \| \ldots \| \mathtt{f}_n)$ and $\mathrm{DRF}(\textbf{let } \Pi_{\mathtt{sc}} \textbf{ in } \mathtt{f}_1 \| \ldots \| \mathtt{f}_n)$,
2. $(tl_{\mathtt{tso}}, ge_o, \pi_o) \preccurlyeq^{o} (sl_{\mathtt{CImp}}, ge_o, \gamma_o)$,

then $\textbf{let } \Pi_{\mathtt{tso}} \textbf{ in } \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n \sqsubseteq' \textbf{let } \Pi_{\mathtt{sc}} \textbf{ in } \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n$.

As explained before, Lem. 16 can be viewed as a strengthened DRF-guarantee theorem for x86-TSO in that, if we let γ_o contain only skip and $ge_o = \emptyset$, Lem. 16 implies the DRF-guarantee of x86-TSO.

7.4 Proof Efforts in Coq

In Coq we have mechanized the framework (Fig. 2) and the extended framework (Fig. 3) and proved all the related lemmas. We have verified all the CompCert passes in Fig. 11.

Statistics of our Coq implementation and proofs are depicted in Fig. 13. Adapting the compilation correctness proofs from CompCert is relatively lightweight. For most passes our proofs are within 300 lines of code more than the original CompCert proofs. The `Stacking` pass introduces more additional proofs, mostly caused by arguments marshalling for supporting cross-language linking. In our experience, adapting CompCert's original compilation proofs to our settings takes less than one person week per translation pass (except for `Stacking`). For simpler passes such as `Tailcall`, `Linearize`, `Allocation`, and `RTLgen`, it takes less than one person day per pass.

By contrast, implementing our framework is more challenging, which took us about 1 person year. In particular, proving the equivalence between non-preemptive and preemptive semantics for DRF programs took us more time than expected, although it seems to be a well-known folklore theorem. The co-inductive proofs there involve a large number of non-trivial cases of reordering threads' executions.

8 Related Work and Conclusion

Compiler verification. Various work extends CompCert [16] to support separate compilation or concurrency. We have discussed Compositional CompCert [2, 29] in Sec. 1 and 2. SepCompCert [15] extends CompCert with the support of syntactical linking. Their approach requires all the compilation units be compiled by CompCert. They do not support cross-language linking or concurrency as we do.

CompCertTSO [27] compiles ClightTSO programs to the x86-TSO machine. It does not support cross-language linking, and its proof for the two CompCert passes `Stacking` and `Cminorgen` are not compositional. By contrast, we have verified these two passes using our compositional simulation. For the other compositional passes, CompCertTSO relies on a thread-local simulation, which is stronger than ours. It requires that the source and the target always generate the same memory events (excepts for those local variables that can be stored in registers). As a result, some optimizations (such as constant propagation and CSE) in CompCertTSO have to be more restrictive.

As an extension of CompCertTSO, Jagannathan et al. [12] allow the compiler to inject racy code such as the efficient spin lock in Fig. 10. They propose a refinement calculus on the racy code to ensure the compilation correctness. Their work looks similar to our extended framework in Fig. 3, but since they use TSO semantics for both the source and target programs, they do not need to handle the gap between the SC and TSO semantics, so they do not need the source to be DRF as in our work.

Podkopaev et al. [24] prove correctness of the compilation from the promising semantics (which is a high-level operational relaxed model) to the operational ARMv8-POP machine. They develop whole-program simulations to deal with the complicated relaxed behaviors. Later on they verify compilations from the promising semantics to declarative hardware models such as POWER, ARMv7 and ARMv8 [25]. The simulations are tied to the promising semantics. They do not aim for reusing existing sequential compilations but handle a lot of complicated issues in relaxed models.

As part of their CCAL framework, Gu et al. [10] develop thread-safe CompCertX (TSCompCertX), which supports separate compilation of concurrent C programs, and the linking of C programs with assembly modules. However, the compositionality of TSCompCertX is tied to the specific settings of the CCAL model. In particular, it relies on the abstraction of concurrent objects to derive the partition of private and shared memory, and uses the auxiliary push and pull instructions to ensure race freedom. Also TSCompCertX supports only race-free programs in the sequentially consistent memory model. Our work does not need source-level specifications for race-free programs. Besides, we support confined benign races in x86-TSO (a feature not supported in TSCompCertX), where the racy objects are required to have race-free abstraction.

Vellvm [35, 36] proves correctness of several optimization passes for *sequential* LLVM programs. Wang et al. [32] verify a separate compiler from Cito to Bedrock, which relies on axiomatic specifications for cross-language external calls. It is unclear how to adapt their work to concurrency. Perconti and Ahmed [23] verify separate compilation by embedding languages in a combined language. They do not support concurrency either. Ševčík [26] studies safety of a class of optimizations in concurrent settings using an abstract trace semantics. It is unclear if his approach can be applied to verify general compilation. Lochbihler [20] verifies a compiler for concurrent Java programs. His simulation has similar restrictions as CompCertTSO.

Non-preemptive semantics and data-race-freedom. Non-preemptive (or cooperative) semantics has been developed in various settings for various purposes (e.g., [1, 5, 18, 31, 34]). Both Ferreira et al. [9] and Xiao et al.[33] study the relationships between non-preemptive semantics and DRF, but they do not give any mechanized proofs of termination-preserving semantics equivalence as in our work. DRFx [21] proposes a concept called Region-Conflict-Freedom, which looks similar to our NPDRF, but there is no formal operational formulation as we do. Owens [22] proposes Triangular-Race-Freedom (TRF) and proves that TRF programs behaves the same in `x86-SC` and `x86-TSO`. TRF is weaker than DRF and can be satisfied by the efficient spin lock code in Fig. 10.

Conclusion and future work. We present a framework for building certified compilation of concurrent programs from sequential compilation. We develop CASCompCert, which reuses CompCert to compile DRF programs, with the support of confined benign races in manually written assembly (`x86-TSO`). We believe our work is a promising start for certified separate compilation of *general* concurrent programs. The latter goal requires longer-term work and has more problems to address, as discussed below.

First, our work is limited in the support of general concurrent languages. For instance, we have not yet considered thread spawn, though we do not see any particular challenges. The spawn step in the operational semantics needs to assign a new F to each newly created thread. In simulations spawns should be handled in a similar way as context switches. Besides, our extended framework currently does not support multiple objects because it lacks a mechanism to ensure the partition between objects' data. To address the problem, we may follow the ideas in LRG [8] and CAP [7] to set the logical boundaries between objects. Also it is worthwhile to support relaxed concurrency, such as C11-style memory models which have relaxed atomics.

Second, it is also interesting to explore other target machine models, such as PowerPC and ARM which have LL/SC instructions rather than lock-prefixed atomic instructions on x86.

Third, we would like to verify more optimization passes, including those relying on concurrency features. For instance, our work may be modified to support roach-motel reorderings, by distinguishing `EntAtom` and `ExtAtom` in the local simulation and recording the footprints that are moved across `EntAtom` or `ExtAtom`. We also would like to finish the proofs for the remaining CompCert optimization passes (which we do not expect any major technical challenges) and add the support of stack pointer escape following the on-paper proofs in our TR [13].

Finally, we are also curious about extensional (language-independent) characterizations of footprints. Our formulation of wd (Def. 1) is one way to characterize footprints but it is not restrictive enough to rule out all benign races. This in practice is harmless since the current wd already allows us to prove our final theorem of the framework (Thm. 12), and we use properly defined concrete languages to prove correctness of specific compilers (Thm. 14). Nevertheless, it is interesting to explore better formulations of wd.

Acknowledgments

We thank our shepherd John Reppy, Zhong Shao and anonymous referees for their suggestions and comments on earlier versions of this paper. We especially thank Weijie Fan and Jianwei Song for their helps on the Coq proofs of the optimization passes `Tunneling` and `CleanupLabels`. This work is supported in part by grants from National Natural Science Foundation of China (NSFC) under Grant No. 61632005, and from Huawei Innovation Research Program (HIRP).

$$\frac{\begin{array}{c} T(\mathrm{t}) = (tl, F, \kappa) \quad F \vdash (\kappa, \sigma) \xmapsto[\mathsf{emp}]{e} (\kappa', \sigma) \\ T' = T\{\mathrm{t} \rightsquigarrow (tl, F, \kappa')\} \end{array}}{(T, \mathrm{t}, 0, \sigma) \xRightarrow[\mathsf{emp}]{e} (T', \mathrm{t}, 0, \sigma)} \ \text{Print}$$

$$\frac{T(\mathrm{t}) = (tl, F, \kappa) \quad \mathrm{t}' \in \mathrm{dom}(T \backslash \mathrm{t}) \quad F \vdash (\kappa, \sigma) \xmapsto[\mathsf{emp}]{\mathsf{ret}} (\kappa', \sigma)}{(T, \mathrm{t}, 0, \sigma) \xRightarrow[\mathsf{emp}]{\mathsf{sw}} (T \backslash \mathrm{t}, \mathrm{t}', 0, \sigma)} \ \text{Term}$$

$$\frac{T(\mathrm{t}) = (tl, F, \kappa) \quad \mathrm{dom}(T) = \{\mathrm{t}\} \quad F \vdash (\kappa, \sigma) \xmapsto[\mathsf{emp}]{\mathsf{ret}} (\kappa', \sigma)}{(T, \mathrm{t}, 0, \sigma) \xRightarrow[\mathsf{emp}]{\tau} \textbf{done}} \ \text{Done}$$

$$\frac{T(\mathrm{t}) = (tl, F, \kappa) \qquad F \vdash (\kappa, \sigma) \xmapsto[\delta]{} \textbf{abort}}{(T, \mathrm{t}, 0, \sigma) \overset{\tau}{\underset{\delta}{\Rightarrow}} \textbf{abort}} \ \text{Abort}$$

Figure 14. More Rules of the Preemptive Global Semantics

References

[1] Martin Abadi and Gordon Plotkin. 2009. A Model of Cooperative Threads. In *POPL*. 29–40.

[2] Lennart Beringer, Gordon Stewart, Robert Dockins, and Andrew W. Appel. 2014. Verified Compilation for Shared-Memory C. In *ESOP*. 107–127.

[3] Hans-Juergen Boehm. 2011. How to Miscompile Programs with "Benign" Data Races. In *HotPar*. 3–3.

[4] Hans-Juergen Boehm and Sarita V. Adve. 2008. Foundations of the C++ concurrency memory model. In *PLDI*. 68–78.

[5] Gérard Boudol. 2007. Fair Cooperative Multithreading. In *CONCUR*. 272–286.

[6] CompCert Developers. 2017. CompCert-3.0.1. http://compcert.inria.fr/release/compcert-3.0.1.tgz

[7] Thomas Dinsdale-Young, Mike Dodds, Philippa Gardner, Matthew J. Parkinson, and Viktor Vafeiadis. 2010. Concurrent Abstract Predicates. In *ECOOP*. 504–528.

[8] Xinyu Feng. 2009. Local rely-guarantee reasoning. In *POPL*. 315–327.

[9] Rodrigo Ferreira, Xinyu Feng, and Zhong Shao. 2010. Parameterized Memory Models and Concurrent Separation Logic. In *ESOP*. 267–286.

[10] Ronghui Gu, Zhong Shao, Jieung Kim, Xiongnan (Newman) Wu, Jérémie Koenig, Vilhelm Sjöberg, Hao Chen, David Costanzo, and Tahina Ramananandro. 2018. Certified Concurrent Abstraction Layers. In *PLDI*. 646–661.

[11] Maurice Herlihy and Nir Shavit. 2008. *The Art of Multiprocessor Programming*. Morgan Kaufmann.

[12] Suresh Jagannathan, Vincent Laporte, Gustavo Petri, David Pichardie, and Jan Vitek. 2014. Atomicity Refinement for Verified Compilation. *ACM Trans. Program. Lang. Syst.* 36, 2 (2014), 6:1–6:30.

[13] Hanru Jiang, Hongjin Liang, Siyang Xiao, Junpeng Zha, and Xinyu Feng. 2019. Towards Certified Separate Compilation for Concurrent Programs (Technical Report and Coq Implementations). https://plax-lab.github.io/publications/ccc/

[14] Cliff B. Jones. 1983. Tentative Steps Toward a Development Method for Interfering Programs. *ACM Trans. Program. Lang. Syst.* 5, 4 (1983), 596–619.

[15] Jeehoon Kang, Yoonseung Kim, Chung-Kil Hur, Derek Dreyer, and Viktor Vafeiadis. 2016. Lightweight Verification of Separate Compilation. In *POPL*. 178–190.

[16] Xavier Leroy. 2009. Formal verification of a realistic compiler. *Commun. ACM* 52, 7 (2009), 107–115.

[17] Xavier Leroy. 2009. A Formally Verified Compiler Back-end. *J. Autom. Reason.* 43 (December 2009), 363–446. Issue 4.

[18] Peng Li and Steve Zdancewic. 2007. Combining Events and Threads for Scalable Network Services Implementation and Evaluation of Monadic, Application-level Concurrency Primitives. In *PLDI*. 189–199.

[19] Hongjin Liang and Xinyu Feng. 2013. Modular Verification of Linearizability with Non-Fixed Linearization Points. In *PLDI*. 459–470.

[20] Andreas Lochbihler. 2010. Verifying a Compiler for Java Threads. In *ESOP*. 427–447.

[21] Daniel Marino, Abhayendra Singh, Todd Millstein, Madanlal Musuvathi, and Satish Narayanasamy. 2010. DRFX: A Simple and Efficient Memory Model for Concurrent Programming Languages. In *PLDI*. 351–362.

[22] Scott Owens. 2010. Reasoning about the Implementation of Concurrency Abstractions on x86-TSO. In *ECOOP*. 478–503.

[23] James T. Perconti and Amal Ahmed. 2014. Verifying an Open Compiler Using Multi-language Semantics. In *ESOP*. 128–148.

[24] Anton Podkopaev, Ori Lahav, and Viktor Vafeiadis. 2017. Promising Compilation to ARMv8 POP. In *ECOOP*. 22:1–22:28.

[25] Anton Podkopaev, Ori Lahav, and Viktor Vafeiadis. 2019. Bridging the gap between programming languages and hardware weak memory models. *PACMPL* 3, POPL (2019), 69:1–69:31.

[26] Jaroslav Ševčík. 2011. Safe optimisations for shared-memory concurrent programs. In *PLDI*. 306–316.

[27] Jaroslav Ševčík, Viktor Vafeiadis, Francesco Zappa Nardelli, Suresh Jagannathan, and Peter Sewell. 2013. CompCertTSO: A Verified Compiler for Relaxed-Memory Concurrency. *J. ACM* 60, 3 (2013), 22.

[28] Peter Sewell, Susmit Sarkar, Scott Owens, Francesco Zappa Nardelli, and Magnus O. Myreen. 2010. x86-TSO: a rigorous and usable programmer's model for x86 multiprocessors. *Commun. ACM* 53, 7 (2010), 89–97.

[29] Gordon Stewart, Lennart Beringer, Santiago Cuellar, and Andrew W. Appel. 2015. Compositional CompCert. In *POPL*. 275–287.

[30] R. K. Treiber. 1986. *System programming: coping with parallelism*. Technical Report RJ 5118. IBM Almaden Research Center.

[31] Jérôme Vouillon. 2008. Lwt: A Cooperative Thread Library. In *ML*. 3–12.

[32] Peng Wang, Santiago Cuellar, and Adam Chlipala. 2014. Compiler Verification Meets Cross-language Linking via Data Abstraction. In *OOPSLA*. 675–690.

[33] Siyang Xiao, Hanru Jiang, Hongjin Liang, and Xinyu Feng. 2018. Non-Preemptive Semantics for Data-Race-Free Programs. In *ICTAC*. 513–531.

[34] Jaeheon Yi, Caitlin Sadowski, and Cormac Flanagan. 2011. Cooperative Reasoning for Preemptive Execution. In *PPoPP*. 147–156.

[35] Jianzhou Zhao, Santosh Nagarakatte, Milo M.K. Martin, and Steve Zdancewic. 2012. Formalizing the LLVM Intermediate Representation for Verified Program Transformations. In *POPL*. 427–440.

[36] Jianzhou Zhao, Santosh Nagarakatte, Milo M.K. Martin, and Steve Zdancewic. 2013. Formal Verification of SSA-based Optimizations for LLVM. In *PLDI*. 175–186.

A More about the Language Settings

Global preemptive semantics. Fig. 14 gives the omitted rules of the global preemptive semantics.

The Print rule shows the step generating an external event. It requires that the flag d must be 0, i.e., external events can be generated only outside of atomic blocks. Also the step generating an external event does *not* access memory, so its footprint is empty (emp).

When the current thread terminates, we either remove it from the thread pool and then switch to another thread if there is one (see the Term rule), or terminate the whole

$$\frac{\begin{array}{l}\text{for all } i \text{ and } j \text{ in } \{1,\dots,n\}, \text{ and } i \neq j:\\ \quad F_i \cap F_j = \emptyset \quad \mathrm{dom}(\sigma)\cap F_i = \emptyset\\ \quad tl_i.\mathsf{InitCore}(\pi_i, \mathsf{f}_i) = \kappa_i, \text{ where } (tl_i, ge_i, \pi_i) \in \Pi\\ \quad T = \{1 \rightsquigarrow (tl_1, F_1, \kappa_1), \dots, n \rightsquigarrow (tl_n, F_n, \kappa_n)\}\\ \quad \sigma = \mathsf{GE}(\Pi) \quad \mathsf{closed}(\sigma)\\ \quad \mathsf{t} \in \mathrm{dom}(T) \quad \mathbb{d} = \{\mathsf{t}_1 \rightsquigarrow 0, \dots, \mathsf{t}_n \rightsquigarrow 0\}\end{array}}{\mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{f}_1 \mid \dots \mid \mathsf{f}_n :\xRightarrow{load} (T, \mathsf{t}, \mathbb{d}, \sigma)}\ \mathsf{Load}_{\mathsf{np}}$$

$$\frac{\begin{array}{c}T(\mathsf{t}) = (tl, F, \kappa)\\ F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}} (\kappa', \sigma') \qquad T' = T\{\mathsf{t} \rightsquigarrow (tl, F, \kappa')\}\end{array}}{(T, \mathsf{t}, \mathbb{d}, \sigma) :\overset{\tau}{\underset{\delta}{\Rightarrow}} (T', \mathsf{t}, \mathbb{d}, \sigma')}\ \tau\text{-}\mathsf{step}_{\mathsf{np}}$$

$$\frac{\begin{array}{c}T(\mathsf{t}) = (tl, F, \kappa) \qquad \mathbb{d}(\mathsf{t}) = 0 \qquad \mathsf{t}' \in \mathrm{dom}(T)\\ F \vdash (\kappa, \sigma) \overset{e}{\underset{\mathsf{emp}}{\longmapsto}} (\kappa', \sigma) \qquad T' = T\{\mathsf{t} \rightsquigarrow (tl, F, \kappa')\}\end{array}}{(T, \mathsf{t}, \mathbb{d}, \sigma) :\overset{e}{\underset{\mathsf{emp}}{\Longrightarrow}} (T', \mathsf{t}', \mathbb{d}, \sigma)}\ \mathsf{Print}_{\mathsf{np}}$$

$$\frac{\begin{array}{c}T(\mathsf{t}) = (tl, F, \kappa) \qquad \mathbb{d}(\mathsf{t}) = 0\\ F \vdash (\kappa, \sigma) \overset{\mathsf{ret}}{\underset{\mathsf{emp}}{\longmapsto}} (\kappa', \sigma) \qquad \mathsf{t}' \in \mathrm{dom}(T\backslash\mathsf{t})\end{array}}{(T, \mathsf{t}, \mathbb{d}, \sigma) :\overset{\mathsf{sw}}{\underset{\mathsf{emp}}{\Longrightarrow}} (T\backslash\mathsf{t}, \mathsf{t}', \mathbb{d}\backslash\mathsf{t}, \sigma)}\ \mathsf{Term}_{\mathsf{np}}$$

$$\frac{\begin{array}{c}T(\mathsf{t}) = \{\mathsf{t} \rightsquigarrow (tl, F, \kappa)\} \qquad \mathbb{d} = \{\mathsf{t} \rightsquigarrow 0\}\\ F \vdash (\kappa, \sigma) \overset{\mathsf{ret}}{\underset{\mathsf{emp}}{\longmapsto}} (\kappa', \sigma)\end{array}}{(T, \mathsf{t}, \mathbb{d}, \sigma) :\overset{\tau}{\underset{\mathsf{emp}}{\Longrightarrow}} \mathbf{done}}\ \mathsf{Done}_{\mathsf{np}}$$

$$\frac{T(\mathsf{t}) = (tl, F, \kappa) \qquad F \vdash (\kappa, \sigma) \underset{\delta}{\longmapsto} \mathbf{abort}}{(T, \mathsf{t}, \mathbb{d}, \sigma) :\overset{\tau}{\underset{\delta}{\Rightarrow}} \mathbf{abort}}\ \mathsf{Abort}_{\mathsf{np}}$$

Figure 15. More Rules of the Non-Preemptive Global Semantics

program if not, as shown in the Done rule. The Term rule generates the sw message to record the context switch.

The Abort rule says the whole program aborts if a local module aborts.

We write $F \vdash \phi \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{*}\phi'$ for zero or multiple silent steps, where δ is the accumulation of the footprint of each step. $F \vdash \phi \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{+}\phi'$ represents at least one step. Similar notations are used for global steps. We also write $W \Rightarrow^{+} W'$ for multiple steps that either are silent or produce sw events. It must contain at least one silent step. $W \overset{e}{\Rightarrow}{}^{+} W'$ represents multiple steps with exactly one e event produced (while other steps either are silent or produce sw events).

Event-trace refinement and equivalence. Below we give the omitted definitions of event-trace refinement and equivalence.

$$PEtr(P, \mathcal{B}) \ \text{ iff } \ \exists W.\, (P \xRightarrow{load} W) \wedge Etr(W, \mathcal{B})$$

$$\frac{W \Rightarrow^{+} \mathbf{abort}}{Etr(W, \mathbf{abort})} \qquad \frac{W \overset{e}{\Rightarrow}{}^{+} W' \qquad Etr(W', \mathcal{B})}{Etr(W, e :: \mathcal{B})}$$

$$\frac{W \Rightarrow^{+} \mathbf{done}}{Etr(W, \mathbf{done})} \qquad \frac{W \Rightarrow^{+} W' \qquad Etr(W', \epsilon)}{Etr(W, \epsilon)}$$

The co-inductive definition of $Etr(W, \mathcal{B})$ says that $\mathcal{B}$ can be produced by executing W. Note we distinguish the traces of non-terminating (diverging) executions from those of terminating ones. If the execution of W diverges, its observable event trace $\mathcal{B}$ is either of infinitely length, or finite but does not end with **done** or **abort** (called *silent divergence*, see the right-most rule above).

Then we define the refinement $P \sqsubseteq \mathbb{P}$ and the equivalence $P \approx \mathbb{P}$ below. They ensure that if P has a diverging execution, so does $\mathbb{P}$. Thus the refinement and the equivalence relations preserve termination.

Definition 17 (Event-Trace Refinement and Equivalence)**.**
$P \sqsubseteq \mathbb{P}$ iff $\forall \mathcal{B}.\ PEtr(P, \mathcal{B}) \implies PEtr(\mathbb{P}, \mathcal{B})$.
$P \approx \mathbb{P}$ iff $\forall \mathcal{B}.\ PEtr(P, \mathcal{B}) \iff PEtr(\mathbb{P}, \mathcal{B})$.

Safety. Below we use the event traces to define $\mathsf{Safe}(\mathbb{W})$ and $\mathsf{Safe}(\mathbb{P})$.

Definition 18 (Safety)**.** $\mathsf{Safe}(\mathbb{W})$ iff $\neg\exists tr.\ Etr(\mathbb{W}, tr :: \mathbf{abort})$.

$\mathsf{Safe}(\mathbb{P})$ iff $(\exists \mathbb{W}.\ \mathbb{P} \xRightarrow{load} \mathbb{W})$ and $\forall \mathbb{W}.\, (\mathbb{P} \xRightarrow{load} \mathbb{W}) \implies \mathsf{Safe}(\mathbb{W})$.

Non-preemptive semantics. Fig. 15 gives the omitted non-preemptive semantics rules. Like $\mathsf{EntAt}_{\mathsf{np}}$ and $\mathsf{ExtAt}_{\mathsf{np}}$, the rules $\mathsf{Print}_{\mathsf{np}}$ and $\mathsf{Term}_{\mathsf{np}}$ execute one step of the current thread t, and then non-deterministically switch to a thread t' (which could just be t). The corresponding global steps produce the sw events (or the external event e in the ($\mathsf{Print}_{\mathsf{np}}$) rule). Other rules are very similar to their counterparts in the preemptive semantics.

B More about the Simulation

Proof of Lemma 6. Proof of Lemma 6 relies on the following non-interference lemma.

Lemma 19 (Non-Interference)**.** For any $\Bbbk, \Bbbk'$ of some well-defined language sl, and any κ, κ' of some well-defined language tl, if $\mathbb{F}_1 \vdash (\Bbbk, \Sigma_0) \overset{\iota}{\underset{\Delta_0}{\longmapsto}}{}^{+}(\Bbbk', \Sigma)$, and $F_1 \vdash (\kappa, \sigma_0) \overset{\iota}{\underset{\delta_0}{\longmapsto}}{}^{+}(\kappa', \sigma)$, then for any $\mu, \mathbb{F}_2$ and F_2 such that $(\mu.\mathbb{S} \cup \mathbb{F}_1) \cap \mathbb{F}_2 = \emptyset$, and $(\mu.S \cup F_1) \cap F_2 = \emptyset$, we have

$$\begin{array}{l}\mathsf{HG}(\Delta_0, \Sigma, \mathbb{F}_1, \mu.\mathbb{S}) \wedge \mathsf{LG}(\mu, (\delta_0, \sigma, F_1), (\Delta_0, \Sigma))\\ \quad \implies \mathsf{Rely}(\mu, (\Sigma_0, \Sigma, \mathbb{F}_2), (\sigma_0, \sigma, F_2))\ .\end{array}$$

Here we use $\mathbb{F} \vdash (\Bbbk, \Sigma_0) \overset{\iota}{\underset{\Delta_0}{\longmapsto}}{}^{+}(\Bbbk', \Sigma)$ to say:

either $\mathbb{F} \vdash (\Bbbk, \Sigma_0) \overset{\tau}{\underset{\Delta_0}{\longmapsto}}{}^{+}(\Bbbk', \Sigma)$,

or $\mathbb{F} \vdash (\Bbbk, \Sigma_0) \overset{\tau}{\underset{\Delta_0}{\longmapsto}}{}^{*}(\Bbbk'', \Sigma)$ and $\mathbb{F} \vdash (\Bbbk'', \Sigma) \overset{\iota}{\underset{\mathsf{emp}}{\longmapsto}} (\Bbbk', \Sigma)$ for some $\Bbbk''$, and $\iota \neq \tau$.

The non-interference lemma says, after the execution of $(\Bbbk, \Sigma_0)$ and (κ, σ_0) until some interaction point with resulting states Σ and σ, we are able to establish the Rely condition over the initial and resulting states with respect to μ and some freelists $\mathbb{F}_2$ and F_2, if the freelists $\mathbb{F}_2$ and F_2 contains no shared location and are disjoint with the freelists $\mathbb{F}_1$ and F_1 respectively, and the execution guarantees HG and LG. Here LG is established by our module-local simulation if HG is guaranteed, and HG is established by RC by the following lemma, which says we can establish HG for multiple steps if the module and corresponding state satisfies RC.

Lemma 20 (RC-guarantee)**.** If $\mathsf{RC}(\mathbb{S}, \mathbb{F}, (\Bbbk, \Sigma_0))$ and $\mathbb{F} \vdash (\Bbbk, \Sigma_0) \underset{\Delta_0}{\overset{\iota}{\longmapsto}}{}^{+} (\Bbbk', \Sigma)$ then $\mathsf{HG}(\Delta_0, \Sigma, \mathbb{F}, \mathbb{S})$.

The non-preemptive global downward simulation. The definition of the whole program simulation is similar to the module local simulation, except the case for environment interference (Rely steps), which is unnecessary for whole program simulation. Every source step should correspond to multiple target steps. As in module local simulation, we always require the target footprints matches those of the source, defined as FPmatch. Also the footprint should always be in scope. As a special requirement for the whole program simulation, we always require the source and the target do lock-step context switch and they always switch to the same thread.

Below we use $\mathsf{curF}(\widehat{W})$ for the freelist of the current thread, i.e., $\mathsf{curF}((\mathbb{T}, \mathtt{t}, \_, \_)) \overset{\text{def}}{=} \mathbb{T}(\mathtt{t}).\mathbb{F}$.

Definition 21 (Whole-Program Downward Simulation)**.**
Let $\hat{\mathbb{P}} = \textbf{let}\ \Gamma\ \textbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m$, $\hat{P} = \textbf{let}\ \Pi\ \textbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m$,
$\Gamma = \{(sl_1, ge_1, \gamma_1), \ldots, (sl_m, ge_m, \gamma_m)\}$ and
$\Pi = \{(tl_1, ge'_1, \pi_1), \ldots, (tl_m, ge'_m, \pi_m)\}$.
We say $\hat{\mathbb{P}} \preccurlyeq \hat{P}$ iff

- $\exists \varphi.\ \big(\forall 1 \le i \le m.\ \lfloor\varphi\rfloor ge_i = ge'_i\big)$, and
- for any $\widehat{\mathbb{W}}$, if $\hat{\mathbb{P}} \overset{load}{:\Longrightarrow} \widehat{\mathbb{W}}$, then there exists $\widehat{W}$, $i \in \mathtt{index}$, μ , such that $\hat{P} \overset{load}{:\Longrightarrow} \widehat{W}$ and $(\widehat{\mathbb{W}}, \mathsf{emp}) \preccurlyeq^i_\mu (\widehat{W}, \mathsf{emp})$.

Here we define $(\widehat{\mathbb{W}}, \Delta_0) \preccurlyeq^i_\mu (\widehat{W}, \delta_0)$ as the largest relation such that whenever $(\widehat{\mathbb{W}}, \Delta_0) \preccurlyeq^i_\mu (\widehat{W}, \delta_0)$, then the following are true:

1. $\widehat{\mathbb{W}}.\mathtt{t} = \widehat{W}.\mathtt{t}$, and $\widehat{\mathbb{W}}.\mathbb{d} = \widehat{W}.\mathbb{d}$;
2. $\forall \widehat{\mathbb{W}}', \Delta.$ if $\widehat{\mathbb{W}} \underset{\Delta}{\overset{\tau}{:\Rightarrow}} \widehat{\mathbb{W}}'$ then one of the following holds:
   a. $\exists j.\ j < i$, and $(\widehat{\mathbb{W}}', \Delta_0 \cup \Delta) \preccurlyeq^j_\mu (\widehat{W}, \delta_0)$; or
   b. $\exists \widehat{W}', \delta, j.$
      i. $\widehat{W} \underset{\delta}{\overset{\tau}{:\Rightarrow}}{}^{+} \widehat{W}'$, and
      ii. $(\delta_0 \cup \delta) \subseteq (\mathsf{curF}(\widehat{W}) \cup \mu.S)$, and $\mathsf{FPmatch}(\mu, \Delta_0 \cup \Delta, \delta_0 \cup \delta)$; and
      iii. $(\widehat{\mathbb{W}}', \Delta_0 \cup \Delta) \preccurlyeq^j_\mu (\widehat{W}', \delta_0 \cup \delta)$;
3. $\forall \mathbb{T}', \mathbb{d}', \Sigma', o, \mathtt{t}'.$ if $o \ne \tau$ and $\widehat{\mathbb{W}} \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma')$, then
   $\exists \widehat{W}', \delta, T', \sigma', j.$ for any $\mathtt{t}'' \in \mathrm{dom}(\mathbb{T}')$, we have
   a. $\widehat{W} \underset{\delta}{\overset{\tau}{:\Rightarrow}}{}^{*} \widehat{W}'$, and $\widehat{W}' \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (T', \mathtt{t}'', \mathbb{d}', \sigma')$, and
   b. $(\delta_0 \cup \delta) \subseteq \mathsf{curF}(\widehat{W}) \cup \mu.S$, and $\mathsf{FPmatch}(\mu, \Delta_0, \delta_0 \cup \delta)$; and
   c. $((\mathbb{T}', \mathtt{t}'', \mathbb{d}', \Sigma'), \mathsf{emp}) \preccurlyeq^j_\mu ((T', \mathtt{t}'', \mathbb{d}', \sigma'), \mathsf{emp})$;
4. if $\widehat{\mathbb{W}} \underset{\mathsf{emp}}{\overset{\tau}{:\Longrightarrow}} \textbf{done}$, then $\exists \widehat{W}', \delta.$
   a. $\widehat{W} \underset{\delta}{\overset{\tau}{:\Rightarrow}}{}^{*} \widehat{W}'$, and $\widehat{W}' \underset{\mathsf{emp}}{\overset{\tau}{:\Longrightarrow}} \textbf{done}$, and
   b. $(\delta_0 \cup \delta) \subseteq \mathsf{curF}(\widehat{W}) \cup \mu.S$, and $\mathsf{FPmatch}(\mu, \Delta_0, \delta_0 \cup \delta)$.

The condition on context switches (lock step, and to the same thread) in our whole-program simulations is not so strong as it sounds. It can be naturally derived from composing our module-local simulations. In our module-local simulation, we already have lock-step EntAtom/ExtAtom steps (see case 2 in Def. 3), which will be switch points in whole programs. Besides, since the local simulation is supposed to relate the source and target of the same thread, it seems natural to require the source and target to always switch to the same thread in the whole-program simulation.

The switching-to-the-same-thread condition makes it easy to flip the whole-program simulation (i.e. derive the whole-program upward simulation from the downward one). The upward and downward simulations require us to map each target thread to some source thread and also map each source thread to a target one. Since the number of target threads is the same as the number of source threads, we would need to require the mapping between the source and target threads to be a bijection. The easiest way to ensure the bijection is to let the thread identifiers to be the same.

The non-preemptive global upward simulation. We define the whole program upward simulation as $\hat{P} \leqslant \hat{\mathbb{P}}$ in Def. 22. It is similar to the downward simulation $\hat{\mathbb{P}} \preccurlyeq \hat{P}$, with the positions of source and target swapped. Note that we do *not* flip the FPmatch condition, since we always require footprint of target program being a refinement of the footprint of the source program, in order to prove DRF of the target program. Correspondingly, the address mapping φ is *not* flipped either.

Definition 22 (Whole-Program Upward Simulation)**.**
Let $\hat{\mathbb{P}} = \textbf{let}\ \Gamma\ \textbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m$, $\hat{P} = \textbf{let}\ \Pi\ \textbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m$,
$\Gamma = \{(sl_1, ge_1, \gamma_1), \ldots, (sl_m, ge_m, \gamma_m)\}$ and
$\Pi = \{(tl_1, ge'_1, \pi_1), \ldots, (tl_m, ge'_m, \pi_m)\}$.
We say $\hat{P} \leqslant \hat{\mathbb{P}}$ iff

- $\exists \varphi.\ \big(\forall 1 \le i \le m.\ \lfloor\varphi\rfloor ge_i = ge'_i\big)$, and
- for any $\widehat{\mathbb{W}}$, if $\hat{P} \overset{load}{:\Longrightarrow} \widehat{W}$, then there exists $\widehat{\mathbb{W}}$, $i \in \mathtt{index}$, μ, such that $\hat{\mathbb{P}} \overset{load}{:\Longrightarrow} \widehat{\mathbb{W}}$ and $(\widehat{W}, \mathsf{emp}) \leqslant^i_\mu (\widehat{\mathbb{W}}, \mathsf{emp})$.

Here we define $(\widehat{W}, \delta_0) \leqslant^i_\mu (\widehat{\mathbb{W}}, \Delta_0)$ as the largest relation such that whenever $(\widehat{W}, \delta_0) \leqslant^i_\mu (\widehat{\mathbb{W}}, \Delta_0)$, then the following are true:

$$\frac{\hat{P} \overset{load}{:\Longrightarrow} \widehat{W} \qquad \widehat{W} \Rightarrow^{*} \widehat{W}' \qquad \widehat{W}' :\Longmapsto \mathsf{Race}}{\hat{P} :\Longmapsto \mathsf{Race}}$$

$$\frac{\hat{P} \overset{load}{:\Longrightarrow} \widehat{W} \quad \mathsf{NPpred}(\widehat{W}, \mathtt{t}_1, (\delta_1, d_1)) \quad \mathtt{t}_1 \neq \mathtt{t}_2 \quad \mathsf{NPpred}(\widehat{W}, \mathtt{t}_2, (\delta_2, d_2)) \qquad (\delta_1, d_1) \frown (\delta_2, d_2)}{\hat{P} :\Longmapsto \mathsf{Race}}$$

$$\frac{\widehat{W} \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} \widehat{W}' \quad o \neq \tau \quad \mathtt{t}_1 \neq \mathtt{t}_2 \quad (\delta_1, d_1) \frown (\delta_2, d_2) \quad \mathsf{NPpred}(\widehat{W}', \mathtt{t}_1, (\delta_1, d_1)) \quad \mathsf{NPpred}(\widehat{W}', \mathtt{t}_2, (\delta_2, d_2))}{\widehat{W} :\Longmapsto \mathsf{Race}} \ \mathsf{Race_{np}}$$

$$\frac{\widehat{W} = (T, \_, \mathbb{d}, \sigma) \qquad T(\mathtt{t}) = (F, \kappa) \qquad F \vdash (\kappa, \sigma) \underset{\delta}{\overset{\tau}{\longmapsto}}{}^{*} (\kappa', \sigma') \qquad \mathbb{d}(\mathtt{t}) = d}{\mathsf{NPpred}(\widehat{W}, \mathtt{t}, (\delta, d))} \ \mathsf{Predict_{np}}$$

Figure 16. Data Race in Non-Preemptive Semantics

1. $\widehat{\mathbb{W}}.\mathtt{t} = \widehat{W}.\mathtt{t}$, and $\widehat{\mathbb{W}}.\mathbb{d} = \widehat{W}.\mathbb{d}$;
2. $\forall \widehat{W}', \delta$. if $\widehat{W} \underset{\delta}{\overset{\tau}{:\Rightarrow}} \widehat{W}'$, then one of the following holds:
   a. $\exists j.\ j < i$, and $(\widehat{W}', \delta_0 \cup \delta) \preccurlyeq_{\mu}^{j} (\widehat{\mathbb{W}}, \Delta_0)$;
   b. $\exists \widehat{\mathbb{W}}', \Delta, j$.
      i. $\widehat{\mathbb{W}} \underset{\Delta}{\overset{\tau}{:\Rightarrow}}{}^{+} \widehat{\mathbb{W}}'$, and
      ii. $(\delta_0 \cup \delta) \subseteq \mathsf{curF}(\widehat{W}) \cup \mu.S$, and $\mathsf{FPmatch}(\mu, \Delta_0 \cup \Delta, \delta_0 \cup \delta)$; and
      iii. $(\widehat{W}', \delta_0 \cup \delta) \preccurlyeq_{\mu}^{j} (\widehat{\mathbb{W}}', \Delta_0 \cup \Delta)$;
3. $\forall T', \mathbb{d}', \sigma', o, \mathtt{t}'$. if $o \neq \tau$ and $\widehat{W} \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (T', \mathtt{t}', \mathbb{d}', \sigma')$, then
   $\exists \widehat{\mathbb{W}}', \Delta, \mathbb{T}', \Sigma', j$. for any $\mathtt{t}'' \in \mathrm{dom}(T')$, we have
   a. $\widehat{\mathbb{W}} \underset{\Delta}{\overset{\tau}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\widehat{\mathbb{W}}' \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (\mathbb{T}', \mathtt{t}'', \mathbb{d}', \Sigma')$, and
   b. $\delta_0 \subseteq \mathsf{curF}(\widehat{W}) \cup \mu.S$, and $\mathsf{FPmatch}(\mu, \Delta_0 \cup \Delta, \delta_0)$; and
   c. $((T', \mathtt{t}'', \mathbb{d}', \sigma'), \mathsf{emp}) \preccurlyeq_{\mu}^{j} ((\mathbb{T}', \mathtt{t}'', \mathbb{d}', \Sigma'), \mathsf{emp})$;
4. if $\widehat{W} \underset{\mathsf{emp}}{\overset{\tau}{:\Longrightarrow}}$ **done**, then $\exists \widehat{\mathbb{W}}', \Delta$.
   a. $\widehat{\mathbb{W}} \underset{\Delta}{\overset{\tau}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\widehat{\mathbb{W}}' \underset{\mathsf{emp}}{\overset{\tau}{:\Longrightarrow}}$ **done**, and
   b. $\delta_0 \subseteq \mathsf{curF}(\widehat{W}) \cup \mu.S$, and $\mathsf{FPmatch}(\mu, \Delta_0 \cup \Delta, \delta_0)$.

Determinism of the target language, and flip of the global simulation.

Definition 23 (Deterministic Languages). $\det(tl)$ iff, for all configuration ϕ in tl (see the definition of ϕ in Fig. 4), and for all F, $F \vdash \phi \underset{\delta_1}{\overset{\iota_1}{\longmapsto}} \phi_1 \wedge F \vdash \phi \underset{\delta_2}{\overset{\iota_2}{\longmapsto}} \phi_2 \implies \phi_1 = \phi_2 \wedge \iota_1 = \iota_2 \wedge \delta_1 = \delta_2$.

Lemma 24 (Flip, ④ in Fig. 2)**.**
For any $\mathtt{f}_1, \ldots, \mathtt{f}_n, ge, \varphi, \Gamma = \{(sl_1, ge_1, \gamma_1), \ldots, (sl_m, ge_m, \gamma_m)\}$, $\Pi = \{(tl_1, ge_1', \pi_1), \ldots, (tl_m, ge_m', \pi_m)\}$, if $\forall i.\ \det(tl_i)$, and

$$\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m \preccurlyeq_{\varphi} \mathbf{let}\ \Pi\ \mathbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m\,,$$
$$\text{then } \mathbf{let}\ \Pi\ \mathbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m \leqslant_{\varphi} \mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m\,.$$

C More about DRF and NPDRF

Data races in the non-preemptive semantics ($\hat{P} :\Longmapsto \mathsf{Race}$) are defined in Fig. 16. Similar to the definition for the preemptive semantics, a non-preemptive program configuration $\widehat{W}$ steps to Race ($\widehat{W} :\Longmapsto \mathsf{Race}$) if two threads are predicted having conflicting footprints, as shown in rule $\mathsf{Race_{np}}$. Predictions are made only at the switch points of non-preemptive semantics ($\widehat{W} \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} \widehat{W}'$ where $o \neq \tau$), i.e., program points at atomic block boundaries, after an observable event, or after thread termination. The prediction rule $\mathsf{Predict_{np}}$ is a unified version of its counterpart Predict-0 and Predict-1 in Fig. 9, where $\mathbb{d}$ indicates whether the predicted steps are inside an atomic block. Similar to the Predict-1 rule, we do *not* insist on predicting the footprint generated by the execution reaching the next switch point, because the code segments between switch points could be nonterminating. In addition to at the switch points, we also need to be able to perform a prediction at the initial state as well (the second rule in Fig. 16), because the first executing thread is non-deterministically picked at the beginning, which has similar effect as thread switching.

A program $\hat{P}$ is NPDRF if it never steps to Race :

$$\mathsf{NPDRF}(\hat{P}) \ \text{ iff } \ \neg(\hat{P} :\Longmapsto \mathsf{Race})$$

Note that defining NPDRF is not for studying the absence of data races in the non-preemptive semantics (which is probably not very interesting since the execution is non-preemptive anyway). Rather, it is intended to serve as an equivalent notion of DRF but formulated in the non-preemptive semantics. The following lemma shows the equivalence.

Lemma 25 (Equivalence between DRF and NPDRF, ⑥ and ⑧ in Fig. 2)**.** For any $\mathtt{f}_1, \ldots, \mathtt{f}_m$,$\Pi = \{(tl_1, \pi_1), \ldots, (tl_m, \pi_m)\}$ such that $\forall i.\ \mathsf{wd}(tl_i)$, and $P = \mathbf{let}\ \Pi\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_m$,

$$\mathsf{DRF}(P) \iff \mathsf{NPDRF}(P)\,.$$

D More about the Extended Framework

D.1 x86-TSO semantics

Our x86-TSO semantics follows the x86-TSO model presented in [22, 28] and CompCertTSO [27].

To make the refinement proof easy, we choose to present the x86-TSO global semantics in a similar way as our preemptive global semantics. The machine states and the global transition rules are shown in Fig. 17 and 18 respectively. Here

- All modules are x86-TSO modules.
- The whole porgram configuration $\mathbb{W}$ is instrumented with write buffers B. Each thread $\mathtt{t}$ has its own write buffer $\mathrm{B}(\mathtt{t})$.
- We do not need the atomic bit and EntAtom, ExtAtom events in the x86-TSO model.

The Unbuffer rule in the global semantics in Fig. 18 says that the machine non-deterministically apply the writes in a thread's buffer to the real memory.

The x86 instruction sets and the local semantics are shown in Fig. 19. For normal writes, a thread always writes to its

$$
\begin{array}{rclllrcll}
(\mathit{Prog}) & P & ::= & \textbf{let}\ \Pi\ \textbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n & & (\mathit{Entry}) & \mathtt{f} & \in & \mathit{String} \\
(\mathit{MdSet}) & \Pi & ::= & \{(ge_1, \pi_1), \ldots, (ge_m, \pi_m)\} & & (\mathit{Module}) & \pi & ::= & \ldots \\
(\mathit{Lang}) & tl_{\mathtt{x86\text{-}TSO}} & ::= & (\mathit{Module},\ \mathit{Core},\ \mathsf{InitCore},\ \longmapsto_{\mathtt{tso}}) & & (\mathit{Core}) & \kappa & ::= & (\pi, \ldots) \\
 & ge & \in & \mathit{Addr} \rightharpoonup_{\mathsf{fin}} \mathit{Val} & & \mathsf{InitCore} & \in & \multicolumn{2}{l}{\mathit{Module} \to \mathit{Entry} \rightharpoonup \mathit{list}(\mathit{Val}) \rightharpoonup \mathit{Core}}
\end{array}
$$

$$
\longmapsto_{\mathtt{tso}} \ \in\ \mathit{FList} \times (\mathit{Core} \times \mathit{State} \times \mathit{Buf}) \to \{\mathcal{P}((\mathit{Msg} \times \mathit{FtPrt}) \times (\mathit{Core} \times \mathit{State} \times \mathit{Buf})), \textbf{abort}\}
$$

$$
\begin{array}{rcllrcll}
(\mathit{ThrdID}) & \mathtt{t} & \in & \mathbb{N} & (\mathit{Addr}) & l & ::= & \ldots \\
(\mathit{State}) & \sigma & \in & \mathit{Addr} \rightharpoonup_{\mathsf{fin}} \mathit{Val} & (\mathit{Val}) & v & ::= & l \mid \ldots \\
(\mathit{Buf}) & \mathrm{b} & ::= & \epsilon \mid (l, v) :: \mathrm{b} & (\mathit{FList}) & F, \mathbb{F} & \in & \mathcal{P}^{\omega}(\mathit{Addr}) \\
(\mathit{Msg}) & \iota & ::= & \tau \mid e \mid \mathsf{call}(\mathtt{f}, \vec{v}) \mid \mathtt{ret}(v) & (\mathit{Event}) & e & ::= & \textbf{print}(v) \\
(\mathit{Config}) & \phi, \Phi & ::= & (\kappa, \sigma) \mid \textbf{abort} & & & &
\end{array}
$$

$$
\begin{array}{rclllrcl}
(\mathit{World}) & W_{\mathtt{tso}} & ::= & (\Pi, \mathcal{F}, T, \mathtt{t}, \mathrm{B}, \sigma) & & & & \\
(\mathit{ThrdPool}) & T, \mathbb{T} & ::= & \{\mathtt{t}_1 \leadsto K_1, \ldots, \mathtt{t}_n \leadsto K_n\} & & & & \\
(\mathit{RtMdStk}) & K, \mathbb{K} & ::= & \epsilon \mid (F, \kappa) :: K & & & & \\
(\mathit{Buffers}) & \mathrm{B} & ::= & \{\mathtt{t}_1 \leadsto \mathrm{b}_1, \ldots, \mathtt{t}_n \leadsto \mathrm{b}_n\} & & & & \\
(\mathit{FSpace}) & \mathcal{F}, \mathbb{FS} & ::= & F :: \mathcal{F} \quad \text{(co-inductive)} & & & & \\
(\mathit{GMsg}) & o & ::= & \tau \mid e \mid \mathtt{sw} \mid \mathtt{ub} & (\mathit{ASet}) & S, \mathbb{S} & \in & \mathcal{P}(\mathit{Addr})
\end{array}
$$

Figure 17. x86TSO machine

buffer rather than the real memory. For `lock`-prefixed instructions, the thread requires the buffer is empty and it updates the memory directly (see the LOCKDEC rule).

$$\frac{\begin{array}{l}\mathcal{F}_0 = F_1 :: \ldots :: F_n :: \mathcal{F} \qquad \mathcal{F} = F_{n+1} :: F_{n+2} :: \ldots \\ \text{for all } i \neq j: \quad F_i \cap F_j = \emptyset \quad \mathrm{dom}(\sigma) \cap F_i = \emptyset \\ \text{for all } i \text{ in } \{1, \ldots, n\}: \quad tl_{\texttt{x86-TSO}}.\mathsf{InitCore}(\pi_i, \mathsf{f}_i) = \kappa_i, \quad \text{where } (tl_{\texttt{x86-TSO}}, ge_i, \pi_i) \in \Pi \\ T = \{1 \rightsquigarrow (tl_{\texttt{x86-TSO}}, F_1, \kappa_1) :: \epsilon, \ldots, n \rightsquigarrow (tl_{\texttt{x86-TSO}}, F_n, \kappa_n) :: \epsilon\} \\ \mathsf{t} \in \{1, \ldots, n\} \quad \sigma = \mathrm{GE}(\Pi) \quad \mathrm{closed}(\sigma)\end{array}}{\textbf{let } \Pi \textbf{ in } \mathsf{f}_1 \parallel \ldots \parallel \mathsf{f}_n \overset{load}{\Longrightarrow}_{\mathsf{tso}} \Pi, \mathcal{F}, T, \mathsf{t}, \{\mathsf{t}_1 \rightsquigarrow \epsilon, \ldots \mathsf{t}_n \rightsquigarrow \epsilon\}} \text{ Load}$$

$$\frac{\begin{array}{c}T(\mathsf{t}) = (F, \kappa) :: K \quad \mathrm{B}(\mathsf{t}) = \mathrm{b} \\ F \vdash (\kappa, (\mathrm{b}, \sigma)) \xrightarrow[\delta\ \ \mathsf{tso}]{\iota} (\kappa', (\mathrm{b}', \sigma')) \\ T' = T\{\mathsf{t} \rightsquigarrow (F, \kappa') :: K\} \quad \iota = \tau\end{array}}{(\Pi, \mathcal{F}, T, \mathsf{t}, \mathrm{B}, \sigma)) \overset{\iota}{\underset{\delta\ \mathsf{tso}}{\Rightarrow}} (\Pi, \mathcal{F}, T', \mathsf{t}, \mathrm{B}\{\mathsf{t} \rightsquigarrow \mathrm{b}'\}\sigma'))} \ \tau\text{-step}$$

$$\frac{\begin{array}{c}T(\mathsf{t}) = (F, \kappa) :: K \quad \mathrm{B}(\mathsf{t}) = \epsilon \\ F \vdash (\kappa, (\epsilon, \sigma)) \xrightarrow[\mathsf{emp}\ \mathsf{tso}]{e} (\kappa', (\epsilon, \sigma)) \\ T' = T\{\mathsf{t} \rightsquigarrow (F, \kappa') :: K\}\end{array}}{(\Pi, \mathcal{F}, T, \mathsf{t}, \mathrm{B}, \sigma)) \overset{e}{\underset{\mathsf{emp}\ \mathsf{tso}}{\Longrightarrow}} (\Pi, \mathcal{F}, T', \mathsf{t}, \mathrm{B}, \sigma))} \text{ Print}$$

$$\frac{\begin{array}{c}T(\mathsf{t}) = (F, \kappa) :: K \qquad \mathrm{B}(\mathsf{t}) = \mathrm{b} \qquad F \vdash (\kappa, (\mathrm{b}, \sigma)) \xrightarrow[\mathsf{emp}\ \ \mathsf{tso}]{\mathsf{call}(\mathsf{f}, \vec{v})} (\kappa', (\mathrm{b}, \sigma)) \\ \mathcal{F} = F_1 :: \mathcal{F}' \qquad \mathsf{initRtMd}(\Pi, \mathsf{f}, \vec{v}, \kappa_1) \qquad T' = \{\mathsf{t} \rightsquigarrow (F_1, \kappa_1) :: (F, \kappa') :: K\}\end{array}}{(\Pi, \mathcal{F}, T, \mathsf{t}, \mathrm{B}, \sigma) \overset{\tau}{\underset{\mathsf{emp}\ \mathsf{tso}}{\Longrightarrow}} (\Pi, \mathcal{F}', T', \mathsf{t}, \mathrm{B}, \sigma)} \text{ EFCall}$$

$$\frac{\begin{array}{c}T(\mathsf{t}) = (F_1, \kappa_1) :: (F, \kappa) :: K, \qquad F_1 \vdash (\kappa_1, (\mathrm{b}, \sigma)) \xrightarrow[\mathsf{emp}\ \ \mathsf{tso}]{\mathsf{ret}(v)} (\kappa_1', (\mathrm{b}, \sigma)) \\ \mathsf{AftExtn}(\kappa, v) = \kappa' \qquad T' = T\{\mathsf{t} \rightsquigarrow (F, \kappa') :: K\}\end{array}}{(\Pi, \mathcal{F}, T, \mathsf{t}, \mathrm{B}, \sigma) \overset{\tau}{\underset{\mathsf{emp}\ \mathsf{tso}}{\Longrightarrow}} (\Pi, \mathcal{F}, T', \mathsf{t}, \mathrm{B}, \sigma)} \text{ EFRet}$$

$$\frac{\begin{array}{c}T(\mathsf{t}) = (F, \kappa) :: \epsilon \quad \mathrm{B}(\mathsf{t}) = \epsilon \quad \mathsf{t}' \in \mathrm{dom}(T \backslash \mathsf{t}) \\ F \vdash (\kappa, (\epsilon, \sigma)) \xrightarrow[\mathsf{tso}]{\mathsf{ret}(v)} (\kappa', (\epsilon, \sigma))\end{array}}{(\Pi, \mathcal{F}, T, \mathsf{t}, \mathrm{B}, \sigma) \overset{\mathsf{sw}}{\underset{\mathsf{emp}\ \mathsf{tso}}{\Longrightarrow}} (\Pi, \mathcal{F}, T \backslash \mathsf{t}, \mathsf{t}', \mathrm{B} \backslash \mathsf{t}, \sigma)} \text{ Term}$$

$$\frac{\begin{array}{c}T(\mathsf{t}) = (F, \kappa) :: \epsilon \quad \mathrm{B}(\mathsf{t}) = \epsilon \quad \mathrm{dom}(T) = \mathrm{dom}(\mathrm{B}) = \{\mathsf{t}\} \\ F \vdash (\kappa, (\epsilon, \sigma)) \xrightarrow[\mathsf{tso}]{\mathsf{ret}(v)} (\kappa', (\epsilon, \sigma))\end{array}}{(\Pi, \mathcal{F}, T, \mathsf{t}, \mathrm{B}, \sigma) \overset{\tau}{\Rightarrow}_{\mathsf{tso}} \textbf{done}} \text{ Done}$$

$$\frac{\mathsf{t}' \in \mathrm{dom}(T)}{(\Pi, \mathcal{F}, T, \mathsf{t}, \mathrm{B}, \sigma) \overset{\mathsf{sw}}{\underset{\mathsf{emp}\ \mathsf{tso}}{\Longrightarrow}} (\Pi, \mathcal{F}, T, \mathsf{t}', \mathrm{B}, \sigma)} \text{ Switch}$$

$$\frac{\mathrm{B}(\mathsf{t}') = (l, v) :: b \quad \sigma' = \mathtt{apply\_buffer}((l, v), \sigma) \quad \mathrm{B}' = \mathrm{B}\{\mathsf{t}' \rightsquigarrow b\}}{(\Pi, \mathcal{F}, T, \mathsf{t}, \mathrm{B}, \sigma) \overset{\mathsf{ub}}{\underset{\mathsf{emp}\ \mathsf{tso}}{\Longrightarrow}} (\Pi, \mathcal{F}, T, \mathsf{t}, \mathrm{B}', \sigma')} \text{ Unbuffer}$$

$$\frac{T(\mathsf{t}) = (F, \kappa) :: \epsilon, \quad \mathrm{B}(\mathsf{t}) = \mathrm{b} \quad F \vdash (\kappa, (\mathrm{b}, \sigma)) \xrightarrow[\mathsf{tso}]{} \textbf{abort}}{(\Pi, \mathcal{F}, T, \mathsf{t}, \mathrm{B}, \sigma) \overset{\tau}{\Rightarrow}_{\mathsf{tso}} \textbf{abort}} \text{ Abort}$$

Figure 18. x86TSO global semantics

$$\begin{array}{rcll}
(\mathit{Instr}) & c & ::= & \mathsf{mov}\ \mathsf{r}_d, \mathsf{r}_s \mid \mathsf{Iadd}\ \mathsf{r}_d, \mathsf{r}_s \mid \mathtt{call}\ \mathsf{f} \mid \mathtt{ret} \mid \ldots \\
 & & \mid & \mathsf{lock\ xchg}\ l\ \mathsf{r}_s \mid \mathsf{lock\ xadd}\ l\ \mathsf{r}_s \mid \mathsf{lock\ dec}\ l \mid \mathtt{mfence} \\
(\mathit{Core}) & \kappa & ::= & \mathit{Register} \rightarrow \mathit{Val} \\
(\mathit{Register}) & \mathsf{r} & ::= & \mathsf{PC} \mid \mathsf{SP} \mid \ldots
\end{array}$$

$$\frac{\mathtt{find\_instr}(ge, \kappa(\mathsf{PC})) = (\mathsf{mov}\ \mathsf{r}_d,\ [\mathsf{r}_s]) \quad \kappa(\mathsf{r}_s) = l \qquad l \notin \mathrm{dom}(\mathrm{b}) \quad \sigma(l) = v \quad \delta = (\{l\}, \emptyset)}{(ge, F) \vdash (\kappa, (\mathrm{b}, \sigma)) \overset{\tau}{\underset{\delta}{\longmapsto}}_{\mathtt{tso}} (\kappa\{\mathsf{r} \rightsquigarrow v, \mathsf{PC} \rightsquigarrow \mathsf{PC}+1\}, (\mathrm{b}, \sigma))} \ \text{READMEM}$$

$$\frac{\mathtt{find\_instr}(ge, \kappa(\mathsf{PC})) = (\mathsf{mov}\ \mathsf{r}_d,\ [\mathsf{r}_s]) \quad \kappa(\mathsf{r}_s) = l \qquad \mathrm{b} = \mathrm{b}_1 +\!\!+ (l, v) :: \mathrm{b}_0 \quad l \notin \mathrm{dom}(\mathrm{b}_0) \quad \delta = (\{l\}, \emptyset)}{(ge, F) \vdash (\kappa, (\mathrm{b}, \sigma)) \overset{\tau}{\underset{\delta}{\longmapsto}}_{\mathtt{tso}} (\kappa\{\mathsf{r} \rightsquigarrow v, \mathsf{PC} \rightsquigarrow \mathsf{PC}+1\}, (\mathrm{b}, \sigma))} \ \text{READBUF}$$

$$\frac{\mathtt{find\_instr}(ge, \kappa(\mathsf{PC})) = (\mathsf{mov}\ [\mathsf{r}_d],\ \mathsf{r}_s) \quad \kappa(\mathsf{r}_s) = v \quad \kappa(\mathsf{r}_d) = l \qquad \delta = (\emptyset, \{l\}) \quad \mathtt{apply\_buffer}(\mathrm{b} +\!\!+ (l, v), \sigma) = \sigma'}{(ge, F) \vdash (\kappa, (\mathrm{b}, \sigma)) \overset{\tau}{\underset{\delta}{\longmapsto}}_{\mathtt{tso}} (\kappa\{\mathsf{PC} \rightsquigarrow \mathsf{PC}+1\}, (\mathrm{b} +\!\!+ (l, v), \sigma))} \ \text{WRITEBUF}$$

$$\frac{\mathtt{find\_instr}(ge, R(\mathsf{PC})) = \mathsf{lock\ dec}\ l\ \mathsf{r} \quad R(\mathsf{SF}) = \sigma(l) \leq 0 \qquad \kappa' = \kappa\{\mathsf{PC} \rightsquigarrow \mathsf{PC}+1, \mathsf{r} \rightsquigarrow \sigma(l)\} \quad \sigma' = \sigma\{l \rightsquigarrow \sigma(l) - 1\} \quad \delta = (\{l\}, \{l\})}{(ge, F) \vdash (\kappa, (\epsilon, \sigma)) \overset{\tau}{\underset{\delta}{\longmapsto}}_{\mathtt{tso}} (\kappa', (\epsilon, \sigma'))} \ \text{LOCKDEC}$$

$$\frac{\mathtt{find\_instr}(ge, R(\mathsf{PC})) = \mathtt{mfence}}{(ge, F) \vdash (\kappa, (\epsilon, \sigma)) \overset{\tau}{\underset{\mathsf{emp}}{\longmapsto}}_{\mathtt{tso}} (\kappa\{\mathsf{PC} \rightsquigarrow \mathsf{PC}+1\}, (\epsilon, \sigma))} \ \text{BARRIER}$$

$$\frac{\mathtt{find\_instr}(ge, R(\mathsf{PC})) = \mathtt{call}\ \mathbf{print}() \qquad \kappa' = \kappa\{\mathsf{PC} \rightsquigarrow \mathsf{PC}+1\} \quad \sigma(\text{first argument}) = v}{(ge, F) \vdash (\kappa, (\epsilon, \sigma)) \xrightarrow[\mathsf{emp}]{\mathbf{print}(v)}_{\mathtt{tso}} (\kappa', (\epsilon, \sigma))} \ \text{PRINT}$$

$$\begin{array}{c}
\mathtt{apply\_buffer}((l, v) :: \mathrm{b}, \sigma) ::= \mathtt{apply\_buffer}(\mathrm{b}, \sigma\{l \rightsquigarrow v\}) \\
\mathtt{apply\_buffer}(\epsilon, \sigma) ::= \sigma
\end{array}$$

Figure 19. x86TSO thread local semantics

$$\begin{array}{rcl}
\mathtt{objM}(l,\Sigma) & \overset{\text{def}}{=} & l \in \mathrm{dom}(\Sigma) \wedge \Sigma(l) = (\_, \mathsf{None}\ ) \\
\mathtt{objFP}(\delta,\Sigma) & \overset{\text{def}}{=} & \forall l \in (\delta.rs \cup \delta.ws)\,.\,\mathtt{objM}(l,\Sigma) \\
\mathtt{clientFP}(\delta,\Sigma) & \overset{\text{def}}{=} & \forall l \in (\delta.rs \cup \delta.ws)\,.\,\neg\mathtt{objM}(l,\Sigma) \\
\mathsf{conflict}_c(\delta,\mathsf{t},\mathsf{B}) & \overset{\text{def}}{=} & (\delta.rs \cup \delta.ws) \cap \bigcup_{\mathsf{t}'\neq\mathsf{t}} \mathrm{dom}(\mathsf{B}(\mathsf{t}')) \neq \emptyset \\
\mathsf{bufConflict}(\mathsf{B},\Sigma) & \overset{\text{def}}{=} & \exists l, \mathsf{t}_1, \mathsf{t}_2. \\
 & & l \in \mathrm{dom}(\mathsf{B}(\mathsf{t}_1)) \cap \mathrm{dom}(\mathsf{B}(\mathsf{t}_2)) \\
 & & \wedge\neg\mathtt{objM}(l,\Sigma) \\
\mathsf{G} \Rightarrow \mathsf{R} & \text{iff} & \mathsf{G}((\Sigma,\sigma),(\Sigma',\sigma')) \implies \mathsf{R}((\Sigma,\sigma),(\Sigma',\sigma')) \\
\mathsf{I} \ltimes \mathsf{I}'((\Sigma,\sigma),(\Sigma',\sigma')) & \overset{\text{def}}{=} & \mathsf{I}((\Sigma,\sigma)) \wedge \mathsf{I}'((\Sigma',\sigma'))
\end{array}$$

Figure 20. Auxiliary definitions for module local simulations

D.2 Object correctness definition

Definition 26 (Object corrrectness).
$(tl_{\mathtt{tso}}, ge_o, \pi_o) \preccurlyeq^{\mathsf{o}} (sl_{\mathtt{CImp}}, ge_o, \gamma_o)$ iff $\exists \mathsf{R}_\mathsf{o}, \mathsf{G}_\mathsf{o}, \mathsf{I}_\mathsf{o}$. such that

1. for any t, $(sl_{\mathtt{CImp}}, ge_o, \gamma_o) \succcurlyeq^{\mathsf{o}}_{\mathsf{R}_\mathsf{o}^\mathsf{t};\mathsf{G}_\mathsf{o}^\mathsf{t};\mathsf{I}_\mathsf{o}} (tl_{\mathtt{tso}}, ge_o, \pi_o)$, and
2. $\mathtt{RespectPartition}_o(\mathsf{R}_\mathsf{o}, \mathsf{G}_\mathsf{o}, \mathtt{objM})$ and
3. $\mathtt{Sta}(\mathsf{I}_\mathsf{o}, \mathsf{R}_\mathsf{o}^\mathsf{t})$ and
4. $\forall \mathsf{t}_1 \neq \mathsf{t}_2.\, \mathsf{G}_\mathsf{o}^{\mathsf{t}_1} \Rightarrow \mathsf{R}_\mathsf{o}^{\mathsf{t}_2}$.

Where $\mathtt{RespectPartition}_o$ is defined in definition 27.
$(sl_{\mathtt{CImp}}, ge_o, \gamma_o) \succcurlyeq^{\mathsf{o}}_{\mathsf{R}_\mathsf{o}^\mathsf{t};\mathsf{G}_\mathsf{o}^\mathsf{t};\mathsf{I}_\mathsf{o}} (tl_{\mathtt{tso}}, ge_o, \pi_o)$ is an extension of Liang and Feng [19] with the support of TSO semantics for the low-level code, and is defined in definition 28.

Definition 27 (Respect partition).
$\mathtt{RespectPartition}_o(\mathsf{R}_\mathsf{o}, \mathsf{G}_\mathsf{o}, p_\mathsf{o})$ iff

1. $\forall \Sigma, \mathsf{B}, \sigma, \Sigma', \mathsf{B}', \sigma', \mathsf{t}$.
$$\Sigma' \xlongequal{\{l \mid p_\mathsf{o}(l,\Sigma)\}} \Sigma \wedge \sigma' \xlongequal{\{l \mid p_\mathsf{o}(l,\Sigma)\}} \sigma$$
$$\wedge \begin{pmatrix} \mathsf{B}' = \mathsf{B} \\ \vee \exists l, v, \mathsf{t}' \neq \mathsf{t}.\, \neg p_\mathsf{o}(l,\Sigma) \wedge \mathsf{B}' = \mathsf{B}\{\mathsf{t}' \leadsto \mathsf{B}(\mathsf{t}') +\!\!+ (l,v)\} \end{pmatrix}$$
$$\implies \mathsf{R}_\mathsf{o}^\mathsf{t}((\Sigma,(\mathsf{B},\sigma)),(\Sigma',(\mathsf{B}',\sigma)))$$
2. $\forall \Sigma, \Sigma', \mathsf{B}, \sigma, \mathsf{B}', \sigma', \mathsf{t}$.
$$\mathsf{G}_\mathsf{o}^\mathsf{t}((\Sigma,(\mathsf{B},\sigma))(\Sigma',(\mathsf{B}',\sigma'))) \implies$$
$$\begin{pmatrix} \Sigma' \xlongequal{\{l \mid \neg p_\mathsf{o}(l,\Sigma)\}} \Sigma \wedge \sigma' \xlongequal{\{l \mid \neg p_\mathsf{o}(l,\Sigma)\}} \sigma \\ \wedge(\mathsf{B}' = \mathsf{B} \vee \exists l, v.\, p_\mathsf{o}(l,\Sigma) \wedge \mathsf{B}' = \mathsf{B}\{\mathsf{t} \leadsto \mathsf{B}(\mathsf{t}) +\!\!+ (l,v)\}) \end{pmatrix}$$

Here we partition memory into object memory and client memory using assertion p_o. And this definition says object rely/guarantee relations $\mathsf{R}_\mathsf{o}/\mathsf{G}_\mathsf{o}$ respect this partition if and only if the object rely R_o holds for any change of non-object memory, and the object guarantee G_o guarantees non-object moemry unchanged.

We instantiated the partition as $\mathtt{objM}$, defined in Fig. 20, which determines whether a location is in object memory by the location's permission.

$(\!|\cdot|\!)$ lifts object global environment ge_o to memory with None permission:

$$(\!|ge_o|\!)(l) \overset{\text{def}}{=} \begin{cases} (ge_o(l), \mathsf{None}\ ), & \text{if } l \in \mathrm{dom}(ge_o) \\ \text{undefined}, & \text{otherwise} \end{cases}$$

Definition 28 (Object module simulation).
$(sl_{\mathtt{CImp}}, ge_o, \gamma_o) \succcurlyeq^{\mathsf{o}}_{\mathsf{R}_\mathsf{o}^\mathsf{t};\mathsf{G}_\mathsf{o}^\mathsf{t};\mathsf{I}_\mathsf{o}} (tl_{\mathtt{tso}}, ge_o, \pi_o)$ iff

1. $\forall \Sigma, \sigma.\, (\!|ge_o|\!)(l) \subseteq \Sigma \wedge ge_o \subseteq \sigma \implies$
$\mathsf{I}_\mathsf{o}(\Sigma, (\{\mathsf{t}_1 \leadsto \epsilon, \ldots, \mathsf{t}_n \leadsto \epsilon\}, \sigma))$, and
2. $\forall \Sigma, \mathsf{B}, \sigma, \mathsf{f}, \vec{v}, \kappa$. if $\mathsf{InitCore}(\pi, \mathsf{f}, \vec{v}) = \kappa$ and $\mathsf{I}_\mathsf{o}(\Sigma, (\mathsf{B}, \sigma))$ then
$\exists \Bbbk.\ \mathsf{InitCore}(\gamma, \mathsf{f}, \vec{v}) = \Bbbk$ and $(\Bbbk, \Sigma) \succcurlyeq^{\mathsf{o}}_{\mathsf{t};\mathsf{R}_\mathsf{o}^\mathsf{t};\mathsf{G}_\mathsf{o}^\mathsf{t};\mathsf{I}_\mathsf{o}} (\kappa, (\mathsf{B}, \sigma))$.
Here $(\Bbbk, \Sigma) \succcurlyeq^{\mathsf{o}}_{\mathsf{t};\mathsf{R}_\mathsf{o}^\mathsf{t};\mathsf{G}_\mathsf{o}^\mathsf{t};\mathsf{I}_\mathsf{o}} (\kappa, (\mathsf{B}, \sigma))$ is defined as the largest relation such that whenever $(\Bbbk, \Sigma) \succcurlyeq^{\mathsf{o}}_{\mathsf{t};\mathsf{R}_\mathsf{o}^\mathsf{t};\mathsf{G}_\mathsf{o}^\mathsf{t};\mathsf{I}_\mathsf{o}} (\kappa, (\mathsf{B}, \sigma))$
, then all the following hold:
   a. if $(\kappa, (\mathsf{B}(\mathsf{t}), \sigma)) \xmapsto[\delta]{\tau} (\kappa', (\mathsf{b}', \sigma')$ then
   either $(\Bbbk, \Sigma) \longmapsto^* \mathbf{abort}$,
   or $\exists \Bbbk', \Sigma', \Delta$. such that
      i. $(\Bbbk, \Sigma) \xmapsto[\Delta]{\tau}{}^* (\Bbbk', \Sigma')$ or
      $(\Bbbk, \Sigma) \xmapsto{\mathsf{EntAtom}} (\Bbbk_0, \Sigma) \xmapsto[\Delta]{\tau}{}^* (\Bbbk_1, \Sigma') \xmapsto{\mathsf{ExtAtom}} (\Bbbk', \Sigma')$
      , and
      ii. $\mathsf{G}_\mathsf{o}^\mathsf{t}((\Sigma, (\mathsf{B}, \sigma)), (\Sigma', (\mathsf{B}\{\mathsf{t} \leadsto \mathsf{b}'\}, \sigma')))$, and $\mathtt{objFP}(\delta, \Sigma)$
      , and
      iii. $(\Bbbk', \Sigma') \succcurlyeq^{\mathsf{o}}_{\mathsf{t};\mathsf{R}_\mathsf{o}^\mathsf{t};\mathsf{G}_\mathsf{o}^\mathsf{t};\mathsf{I}_\mathsf{o}} (\kappa', (\mathsf{B}\{\mathsf{t} \leadsto \mathsf{b}'\}, \sigma'))$.
   b. if $(\kappa, (\mathsf{B}(\mathsf{t}), \sigma)) \xmapsto[\mathsf{emp}]{\mathsf{ret}(v)} (\kappa', (\mathsf{B}(\mathsf{t}), \sigma)$ then
   either $(\Bbbk, \Sigma) \longmapsto^* \mathbf{abort}$,
   or $\exists \Bbbk'$. such that
      i. $(\Bbbk, \Sigma) \xmapsto[\mathsf{emp}]{\mathsf{ret}(v)}{}^* (\Bbbk', \Sigma)$, and
      ii. $(\Bbbk', \Sigma) \succcurlyeq^{\mathsf{o}}_{\mathsf{t};\mathsf{R}_\mathsf{o}^\mathsf{t};\mathsf{G}_\mathsf{o}^\mathsf{t};\mathsf{I}_\mathsf{o}} (\kappa', (\mathsf{B}, \sigma))$.
   c. if $\mathsf{R}_\mathsf{o}^\mathsf{t}((\Sigma, (\mathsf{B}, \sigma)), (\Sigma', (\mathsf{B}', \sigma'))$ then
   $(\kappa, \Sigma') \succcurlyeq^{\mathsf{o}}_{\mathsf{t};\mathsf{R}_\mathsf{o}^\mathsf{t};\mathsf{G}_\mathsf{o}^\mathsf{t};\mathsf{I}_\mathsf{o}} (\kappa, (\mathsf{B}', \sigma'))$.
   d. $\neg(\kappa, (\mathsf{B}(\mathsf{t}), \sigma)) \xmapsto[\mathsf{emp}]{\mathsf{call}(\mathsf{f}, \vec{v})} (\kappa', (\mathsf{B}(\mathsf{t}), \sigma)$.
   e. if $(\kappa, (\mathsf{B}(\mathsf{t}), \sigma)) \longmapsto \mathbf{abort}$, then $(\kappa, \Sigma) \longmapsto \mathbf{abort}$.

D.3 Proof of Theorem 15

Theorem 29 (Correctness with x86-TSO backend and obj.).
For any $\mathsf{f}_1 \ldots \mathsf{f}_n, \Gamma = \{(sl, ge_1, \gamma_1), \ldots, (sl, ge_m, \gamma_m), (sl_{\mathtt{CImp}}, ge_o, \gamma_o)\}$, and $\Pi = \{(tl, ge_1', \pi_1), \ldots, (tl, ge_m', \pi_m), (tl, ge_o, \pi_o)\}$, if

1. $\forall i \in \{1, \ldots, m\}.\ (\mathtt{CompCert.CodeT}(\gamma_i) = \pi_i) \wedge \mathtt{injective}(\varphi)$
$\wedge\ (\mathtt{CompCert.}\varphi = \varphi) \wedge \lfloor\varphi\rfloor ge_i = ge_i'$,
2. $\mathsf{Safe}(\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathsf{f}_1 \parallel \ldots \parallel \mathsf{f}_n)$,
3. $\mathsf{DRF}(\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathsf{f}_1 \parallel \ldots \parallel \mathsf{f}_n)$,
4. $\forall i \in \{1, \ldots, m\}.\ \mathsf{ReachClose}(sl, ge_i, \gamma_i)$,
and $\mathsf{ReachClose}(sl_{\mathtt{CImp}}, ge_o, \gamma_o)$,
5. $(tl, ge_o, \pi_o) \preccurlyeq^{\mathsf{o}} (sl_{\mathtt{CImp}}, ge_o, \gamma_o)$

then $\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathsf{f}_1 \parallel \ldots \parallel \mathsf{f}_n \sqsupseteq' \mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{f}_1 \parallel \ldots \parallel \mathsf{f}_n$.

Proof. By transitivity of refinement, it suffices to prove

$$\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathsf{f}_1 \parallel \ldots \parallel \mathsf{f}_n \sqsupseteq \mathbf{let}\ \Pi_{\mathsf{sc}}\ \mathbf{in}\ \mathsf{f}_1 \parallel \ldots \parallel \mathsf{f}_n \tag{D.1}$$

and

$$\mathbf{let}\ \Pi_{\mathsf{sc}}\ \mathbf{in}\ \mathsf{f}_1 \parallel \ldots \parallel \mathsf{f}_n \sqsupseteq' \mathbf{let}\ \Pi\ \mathbf{in}\ \mathsf{f}_1 \parallel \ldots \parallel \mathsf{f}_n \tag{D.2}$$

Where
$\Pi_{\mathsf{sc}} = \{(tl_{\mathsf{sc}}, ge_1', \pi_1), \ldots, (tl_{\mathsf{sc}}, ge_m', \pi_m), (sl_{\mathtt{CImp}}, ge_o, \gamma_o)\}$
(D.1) is proved by applying Theorem 19, and premise 1-4.

To prove (D.2), by Lemma 30 below, it suffice to prove $\mathsf{Safe}(\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n)$ and $\mathsf{DRF}(\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n)$. The first is implied by premise 2 and (D.1); the second is implied by premise 3 and DRF preservation results as we discussed in section 5. □

Lemma 30 (Restore SC semantics for DRF x86 programs). For any $\Pi_{\mathsf{sc}} = \{(tl_{\mathsf{sc}}, ge_1, \pi_1), \ldots, (tl_{\mathsf{sc}}, ge_m, \pi_m), (sl_{\mathtt{CImp}}, ge_o, \gamma_o)\}$, $\Pi_{\mathsf{tso}} = \{(tl_{\mathsf{tso}}, ge_1, \pi_1), \ldots, (tl_{\mathsf{tso}}, ge_m, \pi_m), (tl_{\mathsf{tso}}, ge_o, \pi_o)\}$, and for any $f_1, \ldots, f_n$,

1. $\mathsf{Safe}(\mathbf{let}\ \Pi_{\mathsf{sc}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n)$, and
2. $\mathsf{DRF}(\mathbf{let}\ \Pi_{\mathsf{sc}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n)$, and
3. $(tl, ge_o, \pi_o) \preccurlyeq^{\mathrm{o}} (sl_{\mathtt{CImp}}, ge_o, \gamma_o)$

then

$\mathbf{let}\ \Pi_{\mathsf{sc}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n \sqsupseteq' \mathbf{let}\ \Pi_{\mathsf{tso}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n$.

Proof. By lemma 32, it suffices to prove $\mathbf{let}\ \Pi_{\mathsf{tso}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n \leqslant_{id} \mathbf{let}\ \Pi_{\mathsf{sc}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n$, as defined below.

By compositionality theorem 37 and lemma 35, 36, 42, it suffices to prove $\mathsf{DRF}(\mathbf{let}\ \Pi_{\mathsf{sc}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n)$, and $\mathsf{Safe}(\mathbf{let}\ \Pi_{\mathsf{sc}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n)$, which are exactly premises 1 and 2. □

D.3.1 Whole-program simulation between TSO and SC

Definition 31 (Whole program simulation).
For any $f_1, \ldots, f_n, \Pi_{\mathsf{sc}}, \Pi_{\mathsf{tso}}, ge_o$,
$\mathbf{let}\ \Pi_{\mathsf{tso}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n \leqslant_{id} \mathbf{let}\ \Pi_{\mathsf{sc}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n$ iff
$\forall \sigma, T, \mathtt{t}, \mathtt{B}.$ if
$\mathbf{let}\ \Pi_{\mathsf{tso}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n \overset{load}{\Longrightarrow}_{\mathsf{tso}} (\Pi_{\mathsf{tso}}, \mathcal{F}, T, \mathtt{t}, (\mathtt{B}, \sigma))$ then
$\exists \Sigma, \mathbb{T}.\ \mathbf{let}\ \Pi_{\mathsf{sc}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n \overset{load}{\Longrightarrow} (\Pi_{\mathtt{x86\text{-}SC}}, \mathcal{F}, \mathbb{T}, \mathtt{t}, 0, \Sigma))$ and
$(\Pi_{\mathsf{tso}}, \mathcal{F}, T, \mathtt{t}, (\mathtt{B}, \sigma)) \leqslant_{id} (\Pi_{\mathsf{sc}}, \mathcal{F}, \mathbb{T}, \mathtt{t}, 0, \Sigma))$.

Here $W \leqslant_{id} \mathbb{W}$ is defined as the largest relation such that whenever $W \leqslant_{id} \mathbb{W}$, then the following hold:

1. if $W \overset{\tau/\mathsf{ub}}{\Longrightarrow} W'$ then there exists $\mathbb{W}'$ such that $\mathbb{W} \overset{\tau}{\Rightarrow}{}^* \mathbb{W}'$ and $W' \leqslant_{id} \mathbb{W}'$.
2. if $W \overset{\iota}{\Rightarrow} W'$ and $\iota \neq \tau$ and $\iota \neq \mathsf{ub}$ then there exists $\mathbb{W}'$ such that $\mathbb{W} \overset{\iota}{\Rightarrow} \mathbb{W}'$ and $W' \leqslant_{id} \mathbb{W}'$.
3. if $W \Longrightarrow \mathbf{abort}$, then $\mathbb{W} \Longrightarrow^* \mathbf{abort}$.
4. if $W \Longrightarrow \mathbf{done}$, then $\mathbb{W} \Longrightarrow^* \mathbf{done}$.

Lemma 32 (whole program simulation implies ref.).
For any $f_1, \ldots, f_n, \Pi_{\mathsf{sc}}, \Pi_{\mathsf{tso}}$,
$\mathbf{let}\ \Pi_{\mathsf{tso}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n \leqslant_{id} \mathbf{let}\ \Pi_{\mathsf{sc}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n$
$\Longrightarrow\ \mathbf{let}\ \Pi_{\mathsf{sc}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n \sqsupseteq_{id} \mathbf{let}\ \Pi_{\mathsf{tso}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \ldots \parallel \mathtt{f}_n$

D.3.2 Client simulation and composition

Definition 33 (Client module simulation).
$(tl_{\mathsf{sc}}, ge, \pi) \succcurlyeq^{\mathsf{c}}_{\mathsf{R}^{\mathsf{t}}_{\mathsf{c}};\mathsf{G}^{\mathsf{t}}_{\mathsf{c}};\mathsf{I}_{\mathsf{c}}} (tl_{\mathsf{tso}}, ge, \pi)$ iff
$\forall \Sigma, \mathtt{B}, \sigma, \mathtt{f}, \vec{v}, \kappa.$ if $\mathsf{InitCore}(\pi, \mathtt{f}, \vec{v}) = \kappa$ and $\mathsf{I}_{\mathsf{c}}(\Sigma, (\mathtt{B}, \sigma))$ then
$(\kappa, \Sigma) \succcurlyeq^{\mathsf{c}}_{\mathtt{t};\mathsf{R}^{\mathsf{t}}_{\mathsf{c}};\mathsf{G}^{\mathsf{t}}_{\mathsf{c}};\mathsf{I}_{\mathsf{c}}} (\kappa, (\mathtt{B}, \sigma))$.

Here $(\kappa, \Sigma) \succcurlyeq^{\mathsf{c}}_{\mathtt{t};\mathsf{R}^{\mathsf{t}}_{\mathsf{c}};\mathsf{G}^{\mathsf{t}}_{\mathsf{c}};\mathsf{I}_{\mathsf{c}}} (\kappa, (\mathtt{B}, \sigma))$ is the largest relation such that whenever $(\kappa, \Sigma) \succcurlyeq^{\mathsf{c}}_{\mathtt{t};\mathsf{R}^{\mathsf{t}}_{\mathsf{c}};\mathsf{G}^{\mathsf{t}}_{\mathsf{c}};\mathsf{I}_{\mathsf{c}}} (\kappa, (\mathtt{B}, \sigma))$, then the following hold:

1. if $(\kappa, (\mathtt{B}(\mathtt{t}), \sigma)) \underset{\delta}{\overset{\iota}{\longmapsto}} (\kappa', (\mathtt{b}', \sigma'))$ then
   either $(\kappa, \Sigma) \longmapsto \mathbf{abort}$
   or $\exists \kappa'', \Sigma', \Delta.$ such that $(\kappa, \Sigma) \underset{\Delta}{\overset{\iota}{\longmapsto}}{}^{+} (\kappa'', \Sigma')$ and $\delta.rs \cup \delta.ws \subseteq \Delta.rs \cup \Delta.ws$ and $\mathtt{clientFP}(\delta, \Sigma)$ and one of the following (a) and (b) holds:
   a. $\neg\mathsf{conflict}_{\mathsf{c}}(\delta, \mathtt{t}, \mathtt{B})$, and $\kappa'' = \kappa'$, and $\Delta = \delta$, and $\mathsf{G}^{\mathsf{t}}_{\mathsf{c}}((\Sigma, (\mathtt{B}, \sigma)), (\Sigma', (\mathtt{B}\{\mathtt{t} \rightsquigarrow \mathtt{b}'\}, \sigma')))$, and $(\kappa', \Sigma') \succcurlyeq^{\mathsf{c}}_{\mathtt{t};\mathsf{R}^{\mathsf{t}}_{\mathsf{c}};\mathsf{G}^{\mathsf{t}}_{\mathsf{c}};\mathsf{I}_{\mathsf{c}}} (\kappa', (\mathtt{B}\{\mathtt{t} \rightsquigarrow \mathtt{b}'\}, \sigma'))$, or
   b. $\mathsf{conflict}_{\mathsf{c}}(\delta, \mathtt{t}, \mathtt{B}, \mathtt{b}')$.
2. if $(\kappa, (\mathtt{B}(\mathtt{t}), \sigma)) \longmapsto \mathbf{abort}$, then $(\kappa, \Sigma) \longmapsto^* \mathbf{abort}$.
3. if $\mathsf{R}^{\mathsf{t}}_{\mathsf{c}}((\Sigma, (\mathtt{B}, \sigma)), (\Sigma', (\mathtt{B}', \sigma')))$, then $(\kappa, \Sigma') \succcurlyeq^{\mathsf{c}}_{\mathtt{t};\mathsf{R}^{\mathsf{t}}_{\mathsf{c}};\mathsf{G}^{\mathsf{t}}_{\mathsf{c}};\mathsf{I}_{\mathsf{c}}} (\kappa, (\mathtt{B}', \sigma'))$.

Definition 34 ($\mathsf{I}_{\mathsf{c}}, \mathsf{R}_{\mathsf{c}}, \mathsf{G}_{\mathsf{c}}$ definitions).

$$\begin{array}{l}
\mathsf{I}_{\mathsf{c}}(\Sigma, (\mathtt{B}, \sigma)) \overset{\text{def}}{=} \\
\quad \neg\mathsf{bufConflict}(\mathtt{B}, \Sigma) \\
\quad \wedge \left(\begin{array}{l} \forall \sigma', l.\ \mathtt{apply\_buffers}(\mathtt{B}, \sigma, \sigma') \wedge \neg\mathtt{objM}(l, \Sigma) \\ \qquad \Longrightarrow\ \Sigma(l) = \sigma'(l) \end{array} \right) \\
\mathsf{R}^{\mathsf{t}}_{\mathsf{c}} \overset{\text{def}}{=} \mathsf{I}_{\mathsf{c}} \ltimes \mathsf{I}_{\mathsf{c}} \\
\mathsf{G}^{\mathsf{t}}_{\mathsf{c}}((\Sigma, (\mathtt{B}, \sigma)), (\Sigma', (\mathtt{B}', \sigma'))) \overset{\text{def}}{=} \\
\quad \mathsf{I}_{\mathsf{c}}(\Sigma, (\mathtt{B}, \sigma)) \wedge \mathsf{I}_{\mathsf{c}}(\Sigma', (\mathtt{B}', \sigma')) \\
\quad \wedge\ \sigma' = \sigma \wedge \Sigma' \overset{\{l \mid \mathtt{objM}(l, \Sigma)\}}{=\!=\!=\!=\!=} \Sigma \\
\quad \wedge \left(\begin{array}{l} \mathtt{B}' = \mathtt{B}\{\mathtt{t} \rightsquigarrow \mathtt{B}(\mathtt{t}) +\!\!+ (l, v)\} \wedge \neg\mathtt{objM}(l, \Sigma) \\ \vee \\ \mathtt{B}' = \mathtt{B} \end{array} \right)
\end{array}$$

Where apply_buffers is inductively defined as:

$$\frac{\mathtt{B} = \{\mathtt{t}_1 \rightsquigarrow \epsilon, \ldots, \mathtt{t}_n \rightsquigarrow \epsilon\}}{\mathtt{apply\_buffers}(\mathtt{B}, \sigma, \sigma)}$$

$$\frac{\begin{array}{c}\mathtt{B}(\mathtt{t}) = (l, v) :: \mathtt{b} \\ \mathtt{apply\_buffers}(\mathtt{B}\{\mathtt{t} \rightsquigarrow \mathtt{b}\}, \sigma\{l \rightsquigarrow v\}, \sigma')\end{array}}{\mathtt{apply\_buffers}(\mathtt{B}, \sigma, \sigma')}$$

I_{c} says buffered writes in different threads are not conflicted (unless they are object writes, i.e.write to location l which is $\mathtt{objM}(l, \Sigma)$) and client memory are equal between SC and TSO when buffers are applied to TSO memory.

$\mathsf{G}^{\mathsf{t}}_{\mathsf{c}}$ says a thread can only append write (l, v) to buffer when the location l is not object memory location.

Lemma 35 (client-object R/G properties).
For any $\mathsf{R}_{\mathsf{o}}, \mathsf{G}_{\mathsf{o}}, \mathsf{I}_{\mathsf{o}}$, if $\mathtt{RespectPartition}_{\mathsf{o}}(\mathsf{R}_{\mathsf{o}}, \mathsf{G}_{\mathsf{o}}, \mathtt{objM})$ then
$\forall \mathtt{t}_1 \neq \mathtt{t}_2.\ \mathsf{G}^{\mathtt{t}_1}_{\mathsf{c}} \Rightarrow \mathsf{R}^{\mathtt{t}_2}_{\mathsf{o}}$ and $\forall \mathtt{t}_1, \mathtt{t}_2.\ \mathsf{G}^{\mathtt{t}_1}_{\mathsf{o}} \Rightarrow \mathsf{R}^{\mathtt{t}_2}_{\mathsf{c}}$

Proof. Trivial by definition of $\mathtt{RespectPartition}_{\mathsf{o}}$ and $\mathsf{R}_{\mathsf{c}}, \mathsf{G}_{\mathsf{c}}$. □

Lemma 36 (client R/G/I properties).
$\mathtt{Sta}(\mathsf{I}_{\mathsf{c}}, \mathsf{R}^{\mathsf{t}}_{\mathsf{c}}) \wedge \forall \mathtt{t}_1, \mathtt{t}_2.\ \mathsf{G}^{\mathtt{t}_1}_{\mathsf{c}} \Rightarrow \mathsf{R}^{\mathtt{t}_2}_{\mathsf{c}}$

Proof. Trivial by definition. □

Theorem 37 (TSO obj. compositionality)**.**
For any $f_1, \dots, f_n$,
$\Pi_{\mathtt{sc}} = \{(tl_{\mathtt{sc}}, ge_1, \pi_1), \dots, (tl_{\mathtt{sc}}, ge_m, \pi_m), (sl_{\mathtt{CImp}}, ge_o, \gamma_o)\}$,
$\Pi_{\mathtt{tso}} = \{(tl_{\mathtt{tso}}, ge_1, \pi_1), \dots, (tl_{\mathtt{tso}}, ge_m, \pi_m), (tl_{\mathtt{tso}}, ge_o, \pi_o)\}$,

$$\frac{\begin{array}{c}\forall i \in \{1, \dots, m\}.\ (tl_{\mathtt{sc}}, ge_i, \pi_i) \succcurlyeq^{\mathsf{c}}_{\mathsf{R}^{\mathsf{t}}_{\mathsf{c}};\mathsf{G}^{\mathsf{t}}_{\mathsf{c}};\mathsf{I}_{\mathsf{c}}} (tl_{\mathtt{tso}}, ge_i, \pi_i)\\ (tl_{\mathtt{tso}}, ge_o, \pi_o) \preccurlyeq^{\mathsf{o}} (sl_{\mathtt{CImp}}, ge_o, \gamma_o)\\ \mathsf{DRF}(\mathbf{let}\ \Pi_{\mathtt{SC}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \dots \parallel \mathtt{f}_n)\\ \mathsf{Safe}(\mathbf{let}\ \Pi_{\mathtt{SC}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \dots \parallel \mathtt{f}_n)\end{array}}{\mathbf{let}\ \Pi_{\mathtt{tso}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \dots \parallel \mathtt{f}_n \leqslant_{id} \mathbf{let}\ \Pi_{\mathtt{sc}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \dots \parallel \mathtt{f}_n}$$

Proof. The proof is similar to RGSim compositionality proof, except that we need to prove the TSO execution will not have conflicting client footprints, which is the result of theorem 38. □

D.3.3 DRF implies no conflicting client footprint on TSO machine

Theorem 38 (DRF implies no conflicting client footprint)**.**
For any $f_1, \dots, f_n$,
$\Pi_{\mathtt{sc}} = \{(tl_{\mathtt{sc}}, ge_1, \pi_1), \dots, (tl_{\mathtt{sc}}, ge_m, \pi_m), (sl_{\mathtt{CImp}}, ge_o, \gamma_o)\}$,
$\Pi_{\mathtt{tso}} = \{(tl_{\mathtt{tso}}, ge_1, \pi_1), \dots, (tl_{\mathtt{tso}}, ge_m, \pi_m), (tl_{\mathtt{tso}}, ge_o, \pi_o)\}$,
if

1. $\forall i \in \{1, \dots, m\}.\ (tl_{\mathtt{sc}}, ge_i, \pi_i) \succcurlyeq^{\mathsf{c}}_{\mathsf{R}^{\mathsf{t}}_{\mathsf{c}};\mathsf{G}^{\mathsf{t}}_{\mathsf{c}};\mathsf{I}_{\mathsf{c}}} (tl_{\mathtt{tso}}, ge_i, \pi_i)$, and
2. $(tl_{\mathtt{tso}}, ge_o, \pi_o) \preccurlyeq^{\mathsf{o}} (sl_{\mathtt{CImp}}, ge_o, \gamma_o)$, and
3. $\mathsf{Safe}(\mathbf{let}\ \Pi_{\mathtt{SC}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \dots \parallel \mathtt{f}_n)$, and
4. $\mathsf{DRF}(\mathbf{let}\ \Pi_{\mathtt{SC}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \dots \parallel \mathtt{f}_n)$, and

then $\forall \mathbb{W}_0, W_0.$, if $\mathbf{let}\ \Pi_{\mathtt{sc}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \dots \parallel \mathtt{f}_n \overset{load}{\Longrightarrow} \mathbb{W}_0$, and
$\mathbf{let}\ \Pi_{\mathtt{tso}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \dots \parallel \mathtt{f}_n \overset{load}{\Longrightarrow}_{\mathtt{tso}} W_0$, then
$\forall W', W''.\ W \Rightarrow^* W' \underset{\delta}{\overset{\iota}{\Rightarrow}} W''$ implies
$\neg(\mathtt{clientFP}(\delta, \Sigma) \wedge \mathsf{conflict}_c(\delta, W'.\mathtt{t}, W'.\mathrm{B}))$

Proof. We prove by contradiction. Assume $\mathtt{clientFP}(\delta, \Sigma) \wedge \mathsf{conflict}_c(\delta, W'.\mathtt{t}, W'.\mathrm{B})$, we show we are able to construct SC execution with data race, which contradicts with premise 4.

By $\mathtt{clientFP}(\delta, \Sigma) \wedge \mathsf{conflict}_c(\delta, W'.\mathtt{t}, W'.\mathrm{B})$, we know there exists $W_1, \dots, W_N, \delta_0, \dots, \delta_{N-1}, \iota_0, \dots, \iota_{N-1}$ such that
$\forall i \in \{0, \dots, N-2\}$.
$W_i \underset{\delta_i}{\overset{\iota_i}{\Longrightarrow}} W_{i+1} \wedge \neg(\mathtt{clientFP}(\delta_i, \Sigma) \wedge \mathsf{conflict}_c(\delta_i, W_i.\mathtt{t}, W_i.\mathrm{B}))$ and
$W_{N-1} \underset{\delta_{N-1}}{\overset{\iota_{N-1}}{\Longrightarrow}} W_N$ and
$\mathtt{clientFP}(\delta_{N-1}, \Sigma) \wedge \mathsf{conflict}_c(\delta_{N-1}, W_{N-1}.\mathtt{t}, W_{N-1}.\mathrm{B})$.

By $\mathsf{conflict}_c(\delta_{N-1}, W_{N-1}.\mathtt{t}, W_{N-1}.\mathrm{B})$, there exists one buffer item (l, v) in buffer of some thread such that $l \in \delta_{N-1}.rs \cup \delta_{N-1}.ws$. The buffer item (l, v) must be inserted during the execution from W_0 to W_N, i.e., there exists $M < N$ such that $W_M.\mathtt{t} \neq W_{N-1}.\mathtt{t}$ and $\delta_M \frown \delta_{N-1}$ and $\mathtt{clientFP}(\delta_M, \Sigma)$ and $l \in \delta_M.ws$, and (l, v) not unbuffered during the execution from W_M to W_N.

$$\begin{tikzcd} W_0 \arrow[r] & \cdots \arrow[r] & W_M \arrow[r, "\delta_M"'] & W_{M+1} \arrow[r] & \cdots \arrow[r] & W_{N-1} \arrow[r, "\delta_{N-1}"'] & W_N \end{tikzcd}$$

Now we reorder the execution, to postpone the execution of thread $W_M.\mathtt{t}$ after W_M, such that we could construct an execution trace as show below, such that $\forall i \in \{0, \dots, N-2\}.\ \neg(\mathtt{clientFP}(\delta'_i, \Sigma) \wedge \mathsf{conflict}_c(\delta'_i, W'_i.\mathtt{t}, W'_i.\mathrm{B}))$ and $\delta' \frown \delta''$ and $l \in \delta'.ws$ and $\neg\mathsf{conflict}_c(\delta', \mathtt{t}', W'_{N-1}.\mathrm{B})$, where $\mathtt{clientFP}(\delta', \Sigma)$ and $\mathtt{clientFP}(\delta'', \Sigma)$.

$$\begin{tikzcd} & & & W'_N \\ W_0 \arrow[r] & \cdots \arrow[r] & W'_{N-1} \arrow[ur, "\delta'", "\mathtt{t}'"'] \arrow[dr, "\delta''"', "\mathtt{t}''"] & \\ & & & W''_N \end{tikzcd}$$

Since thread local steps would not change TSO-memory (they only insert buffer items to buffer), we can safely reorder steps of different threads except unbuffer steps. To reorder normal (not unbuffer) step with unbuffer steps, we apply lemma 39 to reorder non-unbuffer step with unbuffer of the same thread, and lemma 40 to reorder non-unbuffer step with unbuffer of another thread.

Since (l, v) not unbuffered during the execution from W_0 to W_N, we could always apply lemma 39 to reorder step M with unbuffer steps of the same thread.

To show premise of lemma 40 hold, it suffice to prove $\forall i.\ \neg\mathsf{bufConflict}(W_i.\mathrm{B}, \Sigma)$. Consider the invariant $\mathcal{R}$ between SC and TSO program configurations, as defined in definition 41 below. It is obvious that $\mathcal{R}(\mathbb{W}_0, W_0)$ holds.
By $\forall i \in \{0, \dots, N-1\}.\ \neg(\mathtt{clientFP}(\delta_i, \Sigma) \wedge \mathsf{conflict}_c(\delta_i, W_i.\mathtt{t}, W_i.\mathrm{B}))$ and safety premise $\mathsf{Safe}(\mathbf{let}\ \Pi_{\mathtt{SC}}\ \mathbf{in}\ \mathtt{f}_1 \parallel \dots \parallel \mathtt{f}_n)$, we could do induction on N and apply client or object local simulations to construct SC execution $\Sigma_1, \dots, \Sigma_N$ such that $\forall i \in \{0, \dots, N-1\}.\ \mathcal{R}(\mathbb{W}_i, W_i, \mathrm{R}_\mathrm{o}, \mathrm{G}_\mathrm{o}, \mathrm{I}_\mathrm{o})$, as shown below.

$$\begin{tikzcd} \mathbb{W}_0 \arrow[r] \arrow[d, dash] & \cdots \arrow[r] & \mathbb{W}_M \arrow[r] \arrow[d, dash] & \mathbb{W}_{M+1} \arrow[r] \arrow[d, dash] & \cdots \arrow[r] & \mathbb{W}_{N-1} \arrow[d, dash] & \\ W_0 \arrow[r] & \cdots \arrow[r] & W_M \arrow[r] & W_{M+1} \arrow[r] & \cdots \arrow[r] & W_{N-1} \arrow[r] & W_N \end{tikzcd}$$

By definition of $\mathcal{R}$ and I_c, $\forall i < N.\ \neg\mathsf{bufConflict}(W_i.\mathrm{B}, \Sigma)$. Now we have reordered the execution to $W'_1, \dots, W'_{N-1}$.

We do the above induction again on $W'_1, \dots W'_{N-1}$, and construct SC execution $\mathbb{W}'_1, \dots \mathbb{W}'_{N-1}$ such that $\forall i \in \{0, \dots, N-1\}.\ \mathcal{R}(\mathbb{W}'_i, W'_i, \mathrm{R}_\mathrm{o}, \mathrm{G}_\mathrm{o}, \mathrm{I}_\mathrm{o})$, as shown below.

$$\begin{tikzcd} \mathbb{W}_0 \arrow[r] \arrow[d, dash] & \cdots \arrow[r] & \mathbb{W}'_{N-1} \arrow[d, dash] \\ W_0 \arrow[r] & \cdots \arrow[r] & W'_{N-1} \end{tikzcd}$$

By $\mathcal{R}(\mathbb{W}'_{N-1}, W'_{N-1}, \mathrm{R_o}, \mathrm{G_o}, \mathrm{I_o})$, $\texttt{clientFP}(\delta', \Sigma)$, and $\texttt{clientFP}(\delta', \Sigma)$, we can apply client simulation and there exists $\mathbb{W}'_N, \mathbb{W}''_N, \Delta', \Delta''$ such that $\Delta' = \delta'$ and $\delta''.rs \cup \delta''.ws \subseteq \Delta''.rs \cup \Delta''.ws$ and

$$\begin{tikzcd}
 & \mathbb{W}'_N \\
\mathbb{W}'_{N-1} \arrow[ur, "\Delta'", "\mathtt{t}'"'] \arrow[dr, "\Delta''"', "\mathtt{t}''"] & \\
 & \mathbb{W}''_N
\end{tikzcd}$$

Since $l \in \delta'.ws$ and $l \in \delta''.ws \cup \delta''.ws$, we have $\Delta' \frown \Delta''$, which indicates a racing execution that contradicts with premise 4.

□

Lemma 39 (Reordering unbuffer and write buffer step of the same thread)**.**
$\forall \kappa, \mathrm{b}, \sigma, \delta, l, v, l', v'$.
if $(\kappa, ((l, v) :: \mathrm{b}, \sigma)) \overset{\tau}{\underset{\delta}{\longmapsto}}_{\mathtt{tso}} (\kappa, ((l, v) :: \mathrm{b} \mathbin{++} (l', v'), \sigma))$,
then $(\kappa, (\mathrm{b}, \sigma\{l \rightsquigarrow v\})) \overset{\tau}{\underset{\delta}{\longmapsto}}_{\mathtt{tso}} (\kappa, (\mathrm{b} \mathbin{++} (l', v'), \sigma\{l \rightsquigarrow v\}))$.

Proof. Trivial by x86-TSO semantics. □

Lemma 40 (Reordering unbuffer and write buffer step of different threads)**.**
$\forall \kappa, \mathrm{b}, \sigma, \delta, l, v, l', v'$. if $(\kappa, (\mathrm{b}, \sigma)) \overset{\tau}{\underset{\delta}{\longmapsto}}_{\mathtt{tso}} (\kappa, (\mathrm{b} \mathbin{++} (l', v'), \sigma))$ and $l \neq l'$,
then $(\kappa, (\mathrm{b}, \sigma\{l \rightsquigarrow v\})) \overset{\tau}{\underset{\delta}{\longmapsto}}_{\mathtt{tso}} (\kappa, (\mathrm{b} \mathbin{++} (l', v'), \sigma\{l \rightsquigarrow v\}))$.

Proof. It suffice to guarantee $l \notin \delta.rs \cup \delta.ws$. By x86-TSO semantics, a step with non-empty read and write footprint must have the same read/write set, i.e. $\delta.rs = \delta.ws = \{l'\}$. And we have $l \notin \delta.rs \cup \delta.ws$ by premise. □

Definition 41 (whole program simulation invariant)**.**
$\mathcal{R}((\Gamma, \mathbb{FS}, \mathbb{T}, \mathtt{t}, d, \Sigma), (\Pi, \mathcal{F}, T, \mathtt{t}', \mathrm{B}, \sigma), \mathrm{R_o}, \mathrm{G_o}, \mathrm{I_o})$ iff

1. $\mathtt{t} = \mathtt{t}'$, and $d = 0$, and $\mathbb{FS} = \mathcal{F}$, and $\mathrm{I_c}(\Sigma, (\mathrm{B}, \sigma))$, and $\mathrm{I_o}(\Sigma, (\mathrm{B}, \sigma))$, and
2. $\forall \mathtt{t}_1.\ (\mathbb{T}(\mathtt{t}_1), \Sigma) \approx (T(\mathtt{t}_1), \mathrm{B}, \sigma)$, and
3. $\neg(\Gamma, \mathbb{FS}, \mathbb{T}, \mathtt{t}, d, \Sigma) \Longmapsto \mathsf{Race}$ and $\neg(\Gamma, \mathbb{FS}, \mathbb{T}, \mathtt{t}, d, \Sigma) \Rightarrow^* \mathbf{abort}$.

Here $(\mathbb{T}(\mathtt{t}), \Sigma) \approx (T(\mathtt{t}), \mathrm{B}, \sigma)$ iff
if $T(\mathtt{t}) = (F_1, \kappa_1) :: \ldots :: (F_k, \kappa_k) :: \epsilon$
and $\mathbb{T}(\mathtt{t}) = (\mathbb{F}_1, \mathbb{k}_1) :: \ldots :: (\mathbb{F}'_k, \mathbb{k}'_k) :: \epsilon$,
then

1. $k = k'$, and
2. $F_j = \mathbb{F}_j$ forall $j = 1 \ldots k$, and
3. $(\mathbb{k}_j, \Sigma) \succcurlyeq^{\mathrm{c}}_{\mathtt{t};\mathrm{R^t_c};\mathrm{G^t_c};\mathrm{I_c}} (\kappa_j, (\mathrm{B}, \sigma))$ forall $j > 1$, and
4. $(\mathbb{k}_1, \Sigma) \succcurlyeq^{\mathrm{c}}_{\mathtt{t};\mathrm{R^t_c};\mathrm{G^t_c};\mathrm{I_c}} (\kappa_1, (\mathrm{B}, \sigma))$ or $(\mathbb{k}_1, \Sigma) \succcurlyeq^{\mathrm{o}}_{\mathtt{t};\mathrm{R^t_o};\mathrm{G^t_o};\mathrm{I_o}} (\kappa_1, (\mathrm{B}, \sigma))$.

D.3.4 x86 assembly module client simulation

Lemma 42 (Client module simulation)**.**
For any $ge, \pi, \mathtt{t}$, $(tl_{\mathtt{x86\text{-}SC}}, ge, \pi) \succcurlyeq^{\mathrm{c}}_{\mathrm{R^t_c};\mathrm{G^t_c};\mathrm{I_c}} (tl_{\mathtt{tso}}, ge, \pi)$

Proof. Since client code are identical, and CompCert memory model guaranteed x86-SC semantics to abort when accessing memory location with None permission (i.e., $\texttt{objM}$), it suffice to prove if $\mathrm{I_c}(\Sigma, (\mathrm{B}, \sigma))$ then reading/writing the same location l from/to Σ or (B, σ) yields results related by $\mathrm{I_c}$ and $\mathrm{G_c}$, if l does not conflict with B. I.e.,

1. $\forall l, \mathtt{t}. \neg\texttt{objM}(l, \Sigma) \implies \Sigma(l) = \texttt{apply\_buffer}(\mathrm{B}(\mathtt{t}), \sigma)(l)$, and
2. $\forall l, v, \mathtt{t}. \neg\texttt{objM}(l, \Sigma) \wedge \neg\mathsf{conflict}_c((\emptyset, \{l\}), \mathtt{t}, \mathrm{B})$
$\implies \mathrm{I_c}(\Sigma\{l \rightsquigarrow v\}, (\mathrm{B}\{\mathtt{t} \rightsquigarrow \mathrm{B}(\mathtt{t}) \mathbin{++} (l, v)\}, \sigma))$
$\wedge \mathrm{G^t_c}((\Sigma, (\mathrm{B}, \sigma)), (\Sigma\{l \rightsquigarrow v\}, (\mathrm{B}\{\mathtt{t} \rightsquigarrow \mathrm{B}(\mathtt{t}) \mathbin{++} (l, v)\}, \sigma)))$

1 is obvious by definition of $\mathrm{I_c}$.

For 2, to show $\mathrm{I_c}(\Sigma\{l \rightsquigarrow v\}, (\mathrm{B}\{\mathtt{t} \rightsquigarrow \mathrm{B}(\mathtt{t}) \mathbin{++} (l, v)\}, \sigma))$, it suffices to prove $\neg\mathsf{bufConflict}(\mathrm{B}\{\mathtt{t} \rightsquigarrow \mathrm{B}(\mathtt{t}) \mathbin{++} (l, v)\})$, which is obvious by $\neg\mathsf{conflict}_c((\emptyset, \{l\}), \mathtt{t}, \mathrm{B})$. $\mathrm{G^t_c}((\Sigma, (\mathrm{B}, \sigma)), (\Sigma\{l \rightsquigarrow v\}, (\mathrm{B}\{\mathtt{t} \rightsquigarrow \mathrm{B}(\mathtt{t}) \mathbin{++} (l, v)\}, \sigma)))$. The latter is also obvious by $\neg\texttt{objM}(l, \Sigma)$.

□

D.4 Spinlock spec. and impl. satisfies our obj. correctness

We prove the lock specification and implementation in Fig. 10 satisfies our object correctness, i.e., $(tl_{\mathtt{tso}}, ge_{lock}, \pi_{lock}) \preccurlyeq^{\mathrm{o}} (sl_{\mathtt{CImp}}, ge_{lock}, \gamma_{lock})$.

Lemma 43 (Lock impl. correctness)**.**
$(tl_{\mathtt{tso}}, ge_{lock}, \pi_{lock}) \preccurlyeq^{\mathrm{o}} (sl_{\mathtt{CImp}}, ge_{lock}, \gamma_{lock})$

Proof. To prove object correctness, the key is to find correct $\mathrm{R_o}, \mathrm{G_o}, \mathrm{I_o}$.

For the spinlock spec. and impl., we define $\mathrm{R_o}, \mathrm{G_o}, \mathrm{I_o}$ in Fig. 21, where L is the lock variable.

The invariant $\mathrm{I_o}$ says the lock variable has three possible states, which is

1. $\mathrm{I_{available}}$: L in both SC and TSO memory is available, and there is no buffered write to L
2. $\mathrm{I_{unavailable}}$: L in both SC and TSO memory is unavailable, and there is no buffered write to L
3. $\mathrm{I_{buffered}}$: L is available in SC memory, but unavailable in TSO memory and there is no buffered write to L

It is obvious that $\texttt{RespectPartition}_o(\mathrm{R_o}, \mathrm{G_o}, \texttt{objM})$ and $\texttt{Sta}(\mathrm{I_o}, \mathrm{R^t_o})$ and $\forall \mathtt{t}_1 \neq \mathtt{t}_2. \mathrm{G^{t_1}_o} \Rightarrow \mathrm{R^{t_2}_o}$.

The proof of lock spec./impl. has the simulation $(sl_{\mathtt{CImp}}, ge_o, \gamma_o) \succcurlyeq^{\mathrm{o}}_{\mathrm{R^t_o};\mathrm{G^t_o};\mathrm{I_o}} (tl_{\mathtt{tso}}, ge_o, \pi_o)$ is also straight forward, and is mechanized in Coq.

□

$$\begin{array}{rcl}
\mathsf{I}_o & \stackrel{\text{def}}{=} & \mathsf{I}_{\text{available}} \vee \mathsf{I}_{\text{buffered}} \vee \mathsf{I}_{\text{unavailable}} \\
\mathrm{G}_o^t & \stackrel{\text{def}}{=} & \mathrm{G}^t_{\texttt{lock}} \vee \mathrm{G}^t_{\texttt{unlock}} \vee \texttt{Id} \\
\mathsf{R}_o^t & \stackrel{\text{def}}{=} & \left(\bigvee_{t\neq t'} \mathrm{G}_o^{t'}\right) \vee \mathsf{R}_{\text{buffer}} \\
\mathsf{I}_{\text{available}}(\Sigma, (\mathrm{B}, \sigma)) & \stackrel{\text{def}}{=} & \Sigma(L) = (1, \mathsf{None}\ ) \ \wedge\ \sigma(L) = 1 \ \wedge\ (\forall\, t.\ L \notin \mathsf{dom}(\mathrm{B}(t))) \\
\mathsf{I}_{\text{buffered}}(\Sigma, (\mathrm{B}, \sigma)) & \stackrel{\text{def}}{=} & \Sigma(L) = (1, \mathsf{None}\ ) \wedge \sigma(L) = 0 \wedge \exists\, \mathtt{t}, \texttt{buffered\_unlock}(\mathtt{t}, \mathrm{B}) \\
\mathsf{I}_{\text{unavailable}}(\Sigma, (\mathrm{B}, \sigma)) & \stackrel{\text{def}}{=} & \Sigma(L) = (0, \mathsf{None}\ ) \ \wedge\ \sigma(L) = 0 \ \wedge\ (\forall\, t.\ L \notin \mathsf{dom}(\mathrm{B}(t))) \\
\texttt{buffered\_unlock}(\mathtt{t}, \mathrm{B}) & \stackrel{\text{def}}{=} & \mathrm{B}(t) = \mathrm{b} +\!\!+ (L, 1) :: \mathrm{b}' \ \wedge\ L \notin \mathsf{dom}(\mathrm{b}) \cup \mathsf{dom}(\mathrm{b}') \ \wedge\ \forall\, t' \neq t.\ L \notin \mathsf{dom}(\mathrm{B}(t')) \\
\texttt{Id}((\Sigma, (\mathrm{B}, \sigma)), (\Sigma', (\mathrm{B}', \sigma))) & \stackrel{\text{def}}{=} & \Sigma = \Sigma' \ \wedge\ \sigma = \sigma' \ \wedge\ \mathrm{B} = \mathrm{B}' \\
\mathrm{G}^t_{\texttt{lock}}((\Sigma, (\mathrm{B}, \sigma)), (\Sigma', \mathrm{B}', \sigma')) & \stackrel{\text{def}}{=} & \Sigma(L) = (1, \mathsf{None}\ ) \ \wedge\ \sigma(L) = 1 \ \wedge\ \Sigma' = \Sigma\{L \leadsto (0, \mathsf{None}\ )\} \ \wedge\ \sigma' = \sigma\{L \leadsto 0\} \ \wedge\ \mathrm{B} = \mathrm{B}' \\
\mathrm{G}^t_{\texttt{unlock}}((\Sigma, (\mathrm{B}, \sigma)), (\Sigma', \mathrm{B}', \sigma')) & \stackrel{\text{def}}{=} & \Sigma(L) = (0, \mathsf{None}\ ) \ \wedge\ \sigma(L) = 0 \ \wedge\ \Sigma' = \Sigma\{L \leadsto (1, \mathsf{None}\ )\} \ \wedge\ \sigma = \sigma' \ \wedge\ \mathrm{B}' = \mathrm{B}\{t \leadsto \mathrm{B}(t) +\!\!+ (L, 1)\} \\
\mathsf{R}_{\text{buffer}}((\Sigma, (\mathrm{B}, \sigma)), (\Sigma', \mathrm{B}', \sigma')) & \stackrel{\text{def}}{=} & \mathsf{I}_{\text{buffered}} \ltimes \mathsf{I}_{\text{available}} \vee \mathsf{I}_{\text{available}} \ltimes \mathsf{I}_{\text{available}} \vee \mathsf{I}_{\text{buffered}} \ltimes \mathsf{I}_{\text{buffered}} \vee \mathsf{I}_{\text{unavailable}} \ltimes \mathsf{I}_{\text{unavailable}}
\end{array}$$

Figure 21. $\mathsf{R}_o, \mathrm{G}_o, \mathsf{I}_o$ instantiations for spinlock

$(BlockID)$	b	$\in$	$\mathbb{N}$
$(State)$	σ, Σ	$\in$	$BlockID \rightharpoonup_{\mathrm{fin}} \mathbb{N} \rightharpoonup_{\mathrm{fin}} Value$
$(\mathfrak{U})$	l	$::=$	$(b, i) \quad$ where $i \in \mathbb{N}$
$(Value)$	v	$::=$	$l \mid \ldots$
$(BSet)$	$S, \mathbb{S}$	$\in$	$\mathcal{P}(BlockID)$
$(FList)$	$F, \mathbb{F}$	$\in$	$\mathcal{P}^{\omega}(BlockID)$
$(LocSet)$	rs, ws, ls, ge	$\in$	$\mathcal{P}(\mathfrak{U})$
$(FtPrt)$	δ, Δ	$::=$	(rs, ws)

$\mathsf{locs}(\sigma) \stackrel{\mathrm{def}}{=} \{(b, i) \mid b \in \mathsf{dom}(\sigma) \wedge i \in \mathsf{dom}(\sigma(b))\}$

$\mathsf{blocks}(ls) \stackrel{\mathrm{def}}{=} \{b \mid \exists i.\ (b, i) \in ls\}$

$\llbracket S \rrbracket \stackrel{\mathrm{def}}{=} S \times \mathbb{N}$

$\mathsf{forward}(\sigma, \sigma')$ iff $(\mathsf{dom}(\sigma) \subseteq \mathsf{dom}(\sigma')) \wedge (\mathsf{locs}(\sigma') \cap \llbracket \mathsf{dom}(\sigma) \rrbracket \subseteq \mathsf{locs}(\sigma))$

$\sigma \overset{ls}{=\!=} \sigma'$ iff $\forall (b, i) \in ls.\ ((b, i) \notin \mathsf{locs}(\sigma) \cup \mathsf{locs}(\sigma')) \vee$
$\qquad ((b, i) \in \mathsf{locs}(\sigma) \cap \mathsf{locs}(\sigma')) \wedge (\sigma(b)(i) = \sigma'(b)(i))$

$\mathtt{LEqPre}(\sigma_1, \sigma_2, \delta, F)$ iff $\sigma_1 \overset{\delta.rs}{=\!=} \sigma_2 \wedge (\mathsf{locs}(\sigma_1) \cap \delta.ws) = (\mathsf{locs}(\sigma_2) \cap \delta.ws) \wedge (\mathsf{dom}(\sigma_1) \cap F) = (\mathsf{dom}(\sigma_2) \cap F)$

$\mathtt{LEqPost}(\sigma_1, \sigma_2, \delta, F)$ iff $\sigma_1 \overset{\delta.ws}{=\!=} \sigma_2 \wedge (\mathsf{dom}(\sigma_1) \cap F) = (\mathsf{dom}(\sigma_2) \cap F)$

$\mathtt{LEffect}(\sigma_1, \sigma_2, \delta, F)$ iff
$\qquad \sigma_1 \overset{\mathfrak{U}-\delta.ws}{=\!=} \sigma_2 \wedge (\mathsf{locs}(\sigma_2) - \mathsf{locs}(\sigma_1)) \subseteq \delta.ws \wedge (\mathsf{locs}(\sigma_1) - \mathsf{locs}(\sigma_2)) \subseteq \delta.ws \cap \delta.rs \wedge (\mathsf{dom}(\sigma_2) - \mathsf{dom}(\sigma_1)) \subseteq F$

Figure 22. The Abstract Concurrent Language

E The Framework Supporting Stack Pointer Escape

Our framework presented earlier disallows escape of pointers of stack variables. Actually it still works when escaping stack pointers are added back in, though we need to revise the definitions of simulations and the proofs would become more involved. In this section, we give the revised definitions of simulations, and the on-paper proofs of the key lemmas.

Our way to support stack pointer escape is actually very similar to the approach in Compositional CompCert, which is relatively orthogonal to our new framework for concurrent compiler correctness.

E.1 Revised Definitions and Key Lemmas

The state model. To make the problem of escaping stack pointers more concrete, we follow Compositional CompCert and instantiate the state model as a block-based state model. Fig. 22 gives the instantiation of Fig. 4. Now a memory location l is a pair of a block ID b and an offset (a natural number). $\mathfrak{U}$ is the set of all the possible memory locations. We need $\mathfrak{U}$ to take the complement of a set of memory locations. S and F are sets of block IDs, and footprints and ge are sets of locations. We use $\mathsf{dom}(\sigma)$ to get a set of block IDs, and $\mathsf{locs}(\sigma)$ to get a set of locations. The module-local semantics is still abstract. The preemptive and the non-preemptive global semantics is not changed.

We also adapt the definition of "well-defined languages" to the block-based state model. The definition of $\mathsf{wd}(tl)$ given in Def. 1 is not changed, but we need to adapt the predicates $\mathsf{forward}(\sigma, \sigma')$, $\mathtt{LEffect}(\sigma, \sigma', \delta, F)$, $\mathtt{LEqPre}(\sigma, \sigma_1, \delta, F)$ and $\mathtt{LEqPost}(\sigma', \sigma_1', \delta, F)$ used in the definition. Their new definitions are given at the bottom of Fig. 22.

Module-local downward simulation. The revised simulation is still parameterized with μ. But here $\mathbb{S}$ and S are sets of block IDs and f is a function mapping block IDs to addresses.

$$\mu \stackrel{\mathrm{def}}{=} (\mathbb{S}, S, f)\,, \qquad \text{where } \mathbb{S}, S \in \mathcal{P}(BlockID) \text{ and } f \in BlockID \rightharpoonup BlockID \times \mathbb{N}$$

Since we allow stack pointer escape, $\mathbb{S}$ (and S) may include escaped stack pointers, as well as ge. As a consequence, $\mathbb{S}$ (and S) may change during the executions of source and target code. The function f may include mappings from source-level escaped stack pointers to the corresponding target-level escaped stack pointers.

We give the useful auxiliary definitions in Fig. 23. Most definitions are similar to their counterparts in Fig. 8. Definition 44 defines the module-local downward simulation which supports escape of stack pointers.

$\mathsf{wf}(\mu, \Sigma, \sigma)$ iff
$\quad \mathsf{injective}(\mu.f, \Sigma) \wedge (\mu.\mathbb{S} = \mathsf{dom}(\mu.f)) \wedge (\mu.\mathbb{S} \subseteq \mathsf{dom}(\Sigma)) \wedge (\mu.f\{\mu.\mathbb{S}\} \subseteq \mu.S) \wedge (\mu.f\langle\!\langle \mathsf{locs}(\Sigma)\rangle\!\rangle \subseteq \mathsf{locs}(\sigma))$

$\mathsf{wf}(\mu, (\Sigma, \mathbb{F}), (\sigma, F))$ iff
$\quad \mathsf{wf}(\mu, \Sigma, \sigma) \wedge (\mu.f\{\mu.\mathbb{S} \cap \mathbb{F}\} \subseteq F) \wedge (\mu.f\{\mu.\mathbb{S} - \mathbb{F}\} \cap F = \emptyset)$

$\mathsf{FPmatch}(\mu, (\Delta, \Sigma), (\delta, F))$ iff
$\quad \mathsf{G}(\delta, F, \mathsf{LShare}(\mu, \Sigma)) \wedge (\delta.rs \cap \mathsf{LShare}(\mu, \Sigma) \subseteq \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta.rs\rangle\!\rangle) \wedge (\delta.ws \cap \mathsf{LShare}(\mu, \Sigma) \subseteq \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta.ws\rangle\!\rangle)$

$\mathsf{LShare}(\mu, \Sigma) \overset{\text{def}}{=} \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor \rangle\!\rangle$

$\mathsf{HGuar}(\mu, \Delta, \mathbb{F})$ iff $\mathsf{G}(\Delta, \mathbb{F}, \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor)$

$\mathsf{G}(\Delta, \mathbb{F}, ls)$ iff $\Delta.rs \cup \Delta.ws \subseteq \lfloor\!\lfloor \mathbb{F} \rfloor\!\rfloor \cup ls$

$\mathsf{Rely}((\mu, \mu'), (\Sigma, \Sigma', \mathbb{F}), (\sigma, \sigma', F))$ iff $\mathsf{R}(\Sigma, \Sigma', \mathbb{F}, \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor) \wedge \mathsf{R}(\sigma, \sigma', F, \mathsf{LShare}(\mu, \Sigma)) \wedge \mathsf{EvolveR}(\mu, \mu', \Sigma', \sigma', \mathbb{F}, F)$

$\mathsf{R}(\Sigma, \Sigma', \mathbb{F}, ls)$ iff $(\Sigma \xlongequal{\lfloor\!\lfloor \mathbb{F} \rfloor\!\rfloor - ls} \Sigma') \wedge \mathsf{forward}(\Sigma, \Sigma') \wedge ((\mathsf{dom}(\Sigma') - \mathsf{dom}(\Sigma)) \cap \mathbb{F} = \emptyset)$

$\mathsf{EvolveG}(\mu, \mu', \Sigma', \sigma', \mathbb{F}, F)$ iff $\mathsf{Evolve}(\mu, \mu', \Sigma', \sigma', \mathbb{F}, F) \wedge (\mu'.\mathbb{S} - \mu.\mathbb{S} \subseteq \mathbb{F})$

$\mathsf{EvolveR}(\mu, \mu', \Sigma', \sigma', \mathbb{F}, F)$ iff $\mathsf{Evolve}(\mu, \mu', \Sigma', \sigma', \mathbb{F}, F) \wedge ((\mu'.\mathbb{S} - \mu.\mathbb{S}) \cap \mathbb{F} = \emptyset)$

$\mathsf{Evolve}(\mu, \mu', \Sigma', \sigma', \mathbb{F}, F)$ iff $\mathsf{wf}(\mu', (\Sigma', \mathbb{F}), (\sigma', F)) \wedge (\mu.f \subseteq \mu'.f) \wedge \mathsf{Inv}(\mu'.f, \Sigma', \sigma') \wedge \mu'.\mathbb{S} = \mathsf{cl}(\mu.\mathbb{S}, \Sigma') \wedge \mu'.S = \mathsf{cl}(\mu.S, \sigma')$

$\mathsf{Inv}(f, \Sigma, \sigma)$ iff
$\quad \forall b, n, b', n'.\ ((b, n) \in \mathsf{locs}(\Sigma)) \wedge (f\langle (b, n)\rangle = (b', n')) \implies ((b', n') \in \mathsf{locs}(\sigma)) \wedge (\Sigma(b)(n) \overset{f}{\hookrightarrow} \sigma(b')(n'))$

$v_1 \overset{f}{\hookrightarrow} v_2$ iff
$\quad (v_1 \notin \mathfrak{U}) \wedge (v_1 = v_2) \vee v_1 \in \mathfrak{U} \wedge v_2 \in \mathfrak{U} \wedge f\langle v_1 \rangle = v_2$

$\mathsf{injective}(f, \Sigma)$ iff
$\quad \forall l_1, l_2, l_1', l_2'.\ l_1 \neq l_2 \wedge l_1 \in \mathsf{locs}(\Sigma) \wedge l_2 \in \mathsf{locs}(\Sigma) \wedge f\langle l_1 \rangle = l_1' \wedge f\langle l_2 \rangle = l_2' \implies l_1' \neq l_2'$

$\mathsf{injective}(f)$ iff
$\quad \forall l_1, l_2, l_1', l_2'.\ l_1 \neq l_2 \wedge f\langle l_1 \rangle = l_1' \wedge f\langle l_2 \rangle = l_2' \implies l_1' \neq l_2'$

$\mathsf{initM}(\varphi, ge, \Sigma, \sigma)$ iff $ge \subseteq \mathsf{locs}(\Sigma) \wedge \mathsf{closed}(\mathsf{dom}(\Sigma), \Sigma) \wedge \mathsf{locs}(\sigma) = \varphi\langle\!\langle \mathsf{locs}(\Sigma) \rangle\!\rangle \wedge \mathsf{dom}(\sigma) = \varphi\{\mathsf{dom}(\Sigma)\} \wedge \mathsf{Inv}(\varphi, \Sigma, \sigma)$

$f\{\mathbb{S}\} \overset{\text{def}}{=} \{b' \mid \exists b.\ (b \in \mathbb{S}) \wedge f(b) = (b', \_)\}$

$f\langle l \rangle \overset{\text{def}}{=} (b', n + n')$, if $l = (b, n) \wedge f(b) = (b', n')$

$f\langle\!\langle ws \rangle\!\rangle \overset{\text{def}}{=} \{l' \mid \exists l.\ (l \in ws) \wedge f\langle l \rangle = l'\}$

$f|_{\mathbb{S}} \overset{\text{def}}{=} \{(b, f(b)) \mid b \in (\mathbb{S} \cap \mathsf{dom}(f))\}$

$f_2 \circ f_1 \overset{\text{def}}{=} \{(b_1, (b_3, n_2 + n_3)) \mid \exists b_2.\ f_1(b_1) = (b_2, n_2) \wedge f_2(b_2) = (b_3, n_3)\}$

$\mathsf{closed}(\mathbb{S}, \Sigma)$ iff $\mathsf{cl}(\mathbb{S}, \Sigma) \subseteq \mathbb{S}$
$\mathsf{cl}(\mathbb{S}, \Sigma) \overset{\text{def}}{=} \bigcup_k \mathsf{cl}_k(\mathbb{S}, \Sigma)$, where $\mathsf{cl}_k(\mathbb{S}, \Sigma)$ is inductively defined:
$\quad \mathsf{cl}_0(\mathbb{S}, \Sigma) \overset{\text{def}}{=} \mathbb{S}$
$\quad \mathsf{cl}_{k+1}(\mathbb{S}, \Sigma) \overset{\text{def}}{=} \{b' \mid \exists b, n, n'.\ (b \in \mathsf{cl}_k(\mathbb{S}, \Sigma)) \wedge \Sigma(b)(n) = (b', n')\}$

Figure 23. Footprint Matching and Evolution of μ in Our Simulation

The key to support stack pointer escape is to describe the possible changes of μ (i.e., $\mathbb{S}$, S and the mapping f between them) during executions. In Definition 44, we are allowed to pick a new μ' at an interaction point (see case 2). The new μ' and the earlier μ should be related by $\mathsf{EvolveG}(\mu, \mu', \ldots)$, which is defined in Fig. 23. In its definition, $\mathsf{Evolve}(\mu, \mu', \ldots)$ requires that μ' is well-formed, $\mu'.\mathbb{S}$ and $\mu'.S$ are closed and the mapping $\mu'.f$ can only be enlarged. The condition $(\mu'.\mathbb{S} - \mu.\mathbb{S} \subseteq \mathbb{F})$ says that a module cannot leak out arbitrary addresses, instead it can only leak out pointers pointing to its own stack.

Similarly, the environment step (see case 3 in Definition 44) may also modify μ, but the updates are confined using $\mathsf{EvolveR}(\mu, \mu', \ldots)$, which is also defined in Fig. 23. Here the condition $((\mu'.\mathbb{S} - \mu.\mathbb{S}) \cap \mathbb{F} = \emptyset)$ says that it is impossible for the environment to leak out pointers pointing to the stack of the current module.

Other parts in the simulation definition (Definition 44) are very similar to Definition 2. In particular, the silent step case (case 1) is not changed (though we slightly modified the presentation), since the shared state updates are not exposed to the environment between two interaction points.

Definition 44 (Module-Local Downward Simulation)**.**
$(sl, ge, \gamma) \preccurlyeq_{\varphi} (tl, ge', \pi)$ iff
for all $\mathsf{f}, \Bbbk, \Sigma, \sigma, \mathbb{F}, F$, and $\mu = (\mathsf{dom}(\Sigma), \mathsf{dom}(\sigma), \varphi|_{\mathsf{dom}(\Sigma)})$, if $sl.\mathsf{InitCore}(\gamma, \mathsf{f}) = \Bbbk$, $\varphi\langle\!\langle ge\rangle\!\rangle = ge'$, $\mathsf{initM}(\varphi, ge, \Sigma, \sigma)$, and $\mathbb{F} \cap \mathsf{dom}(\Sigma) = F \cap \mathsf{dom}(\sigma) = \emptyset$, then there exists $i \in \mathtt{index}$ and κ such that $tl.\mathsf{InitCore}(\pi, \mathsf{f}) = \kappa$, and
$(\mathbb{F}, (\Bbbk, \Sigma), \mathsf{emp}, \bullet) \preccurlyeq^{i}_{\mu} (F, (\kappa, \sigma), \mathsf{emp}, \bullet)$.

Here we define $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \beta) \preccurlyeq^{i}_{\mu} (F, (\kappa, \sigma), \delta_0, \beta)$ (where the bit β is either $\circ$ or $\bullet$[8]) as the largest relation such that, whenever $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \beta) \preccurlyeq^{i}_{\mu} (F, (\kappa, \sigma), \delta_0, \beta)$, then $\mathsf{wf}(\mu, (\Sigma, \mathbb{F}), (\sigma, F))$, and the following are true:

1. $\forall \Bbbk', \Sigma', \Delta$. if $\beta = \circ$ and $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\tau}{\underset{\Delta}{\longmapsto}} (\Bbbk', \Sigma')$ and $\mathsf{HGuar}(\mu, \Delta_0 \cup \Delta, \mathbb{F})$, then one of the following holds:
   a. there exists j such that
      i. $j < i$, and
      ii. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa, \sigma), \delta_0, \circ)$;
   b. or, there exist κ', σ', δ and j such that
      i. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{+} (\kappa', \sigma')$, and
      ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta, F))$, and
      iii. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa', \sigma'), \delta_0 \cup \delta, \circ)$;
2. $\forall \Bbbk', \iota$. if $\beta = \circ$, $\iota \neq \tau$ and $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\iota}{\underset{\mathsf{emp}}{\longmapsto}} (\Bbbk', \Sigma)$ and $\mathsf{HGuar}(\mu, \Delta_0, \mathbb{F})$,
   then there exist κ', δ, σ', κ', μ' and j such that
   a. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{*} (\kappa', \sigma')$, and
      $F \vdash (\kappa', \sigma') \overset{\iota}{\underset{\mathsf{emp}}{\longmapsto}} (\kappa'', \sigma')$, and
   b. $\mathsf{FPmatch}(\mu, (\Delta_0, \Sigma), (\delta_0 \cup \delta, F))$, and
   c. $\mathsf{EvolveG}(\mu, \mu', \Sigma, \sigma', \mathbb{F}, F)$, and
   d. $(\mathbb{F}, (\Bbbk', \Sigma), \mathsf{emp}, \bullet) \preccurlyeq^{j}_{\mu'} (F, (\kappa'', \sigma'), \mathsf{emp}, \bullet)$;
3. $\forall \sigma', \Sigma', \mu'$, if $\beta = \bullet$ and $\mathsf{Rely}((\mu, \mu'), (\Sigma, \Sigma', \mathbb{F}), (\sigma, \sigma', F))$, then there exists j such that
   $(\mathbb{F}, (\Bbbk, \Sigma'), \mathsf{emp}, \circ) \preccurlyeq^{j}_{\mu'} (F, (\kappa, \sigma'), \mathsf{emp}, \circ)$.

Lemma 45 (Transitivity)**.**
If $(sl, ge, \gamma) \preccurlyeq_{\varphi} (sl_1, ge_1, \gamma_1)$, $(sl_1, ge_1, \gamma_1) \preccurlyeq_{\varphi'} (tl, ge', \pi)$, $\varphi\langle\!\langle ge\rangle\!\rangle = ge'$ and $\forall \mathbb{S}.\ \varphi\{\mathbb{S}\} \subseteq \mathsf{dom}(\varphi')$, then $(sl, ge, \gamma) \preccurlyeq_{\varphi' \circ \varphi} (tl, ge', \pi)$.

Compositionality and the non-preemptive global downward simulation. To support stack pointer escape, the whole program simulations also allow one to pick a new μ' at interaction points (see case 3 in Definition 46). The new definition of reach-close modules (see Definition 47) also allows $\mathbb{S}$ to change, which is similar to the reach-closed definition in Compositional CompCert.

Definition 46 (Whole-Program Downward Simulation)**.**
Let $\hat{\mathbb{P}} = \mathbf{let}\,\Gamma\,\mathbf{in}\ \mathsf{f}_1| \ldots |\mathsf{f}_m$, $\hat{P} = \mathbf{let}\,\Pi\,\mathbf{in}\ \mathsf{f}_1| \ldots |\mathsf{f}_m$, $\Gamma = \{(sl_1, ge_1, \gamma_1), \ldots, (sl_m, ge_m, \gamma_m)\}$ and $\Pi = \{(tl_1, ge'_1, \pi_1), \ldots, (tl_m, ge'_m, \pi_m)\}$.

[8] β is introduced to simplify the presentation only. It is used to make the rely step explicit (i.e. take the clause 3 out from the clause 2 in Def. 44), so the proofs can be cleaner.

$\hat{\mathbb{P}} \preccurlyeq_{\varphi} \hat{P}$ iff $\forall i \in \{1, \ldots, m\}.\ \varphi\langle\!\langle ge_i \rangle\!\rangle = ge'_i$, and

for any $\Sigma, \sigma, \mu, \widehat{\mathbb{W}}$, if $\mathsf{initM}(\varphi, \bigcup ge_i, \Sigma, \sigma)$, $\mu = (\mathsf{dom}(\Sigma), \mathsf{dom}(\sigma), \varphi)$, and $(\hat{\mathbb{P}}, \Sigma) \overset{load}{:\Longrightarrow} \widehat{\mathbb{W}}$, then there exists $\widehat{W}$, $i \in \mathtt{index}$ such that $(\hat{P}, \sigma) \overset{load}{:\Longrightarrow} \widehat{W}$ and $(\widehat{\mathbb{W}}, \mathsf{emp}) \preccurlyeq^i_{\mu} (\widehat{W}, \mathsf{emp})$.

Here we define $(\widehat{\mathbb{W}}, \Delta_0) \preccurlyeq^i_{\mu} (\widehat{W}, \delta_0)$ as the largest relation such that whenever $(\widehat{\mathbb{W}}, \Delta_0) \preccurlyeq^i_{\mu} (\widehat{W}, \delta_0)$, then the following are true:

1. $\widehat{\mathbb{W}}.\mathtt{t} = \widehat{W}.\mathtt{t}$, and $\widehat{\mathbb{W}}.\mathbb{d} = \widehat{W}.\mathbb{d}$, and $\mathsf{wf}(\mu, \widehat{\mathbb{W}}.\Sigma, \widehat{W}.\sigma)$;
2. $\forall \widehat{\mathbb{W}}', \Delta$. if $\widehat{\mathbb{W}} \underset{\Delta}{\overset{\tau}{:\Rightarrow}} \widehat{\mathbb{W}}'$, then one of the following holds:
   a. there exists j such that
      i. $j < i$, and
      ii. $(\widehat{\mathbb{W}}', \Delta_0 \cup \Delta) \preccurlyeq^j_{\mu} (\widehat{W}, \delta_0)$;
   b. or, there exist $\widehat{W}'$, δ and j such that
      i. $\widehat{W} \underset{\delta}{\overset{\tau}{:\Rightarrow}}{}^{+} \widehat{W}'$, and
      ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta, \mathsf{curF}(\widehat{W})))$, and
      iii. $(\widehat{\mathbb{W}}', \Delta_0 \cup \Delta) \preccurlyeq^j_{\mu} (\widehat{W}', \delta_0 \cup \delta)$;
3. $\forall \mathbb{T}', \mathbb{d}', \Sigma', o$. if $o \neq \tau$ and $\exists \mathtt{t}'.\ \widehat{\mathbb{W}} \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma')$, then there exist $\widehat{W}', \delta, \mu', T', \sigma'$ and j such that for any $\mathtt{t}' \in \mathsf{dom}(\mathbb{T}')$, we have
   a. $\widehat{W} \underset{\delta}{\overset{\tau}{:\Rightarrow}}{}^{*} \widehat{W}'$, and $\widehat{W}' \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (T', \mathtt{t}', \mathbb{d}', \sigma')$, and
   b. $\mathsf{FPmatch}(\mu, (\Delta_0, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta, \mathsf{curF}(\widehat{W})))$, and
   c. $((\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma'), \mathsf{emp}) \preccurlyeq^j_{\mu'} ((T', \mathtt{t}', \mathbb{d}', \sigma'), \mathsf{emp})$;
4. if $\widehat{\mathbb{W}} \underset{\mathsf{emp}}{\overset{\tau}{:\Longrightarrow}}$ **done**, then there exist $\widehat{W}'$ and δ such that
   a. $\widehat{W} \underset{\delta}{\overset{\tau}{:\Rightarrow}}{}^{*} \widehat{W}'$, and $\widehat{W}' \underset{\mathsf{emp}}{\overset{\tau}{:\Longrightarrow}}$ **done**, and
   b. $\mathsf{FPmatch}(\mu, (\Delta_0, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta, \mathsf{curF}(\widehat{W})))$.

Definition 47 (Reach Closed Module). $\mathsf{ReachClose}(sl, ge, \gamma)$ iff , for all $\mathtt{f}$, $\mathbb{k}$, Σ, $\mathbb{F}$ and $\mathbb{S}$, if $sl.\mathsf{InitCore}(\gamma, \mathtt{f}) = \mathbb{k}$, $\mathbb{S} = \mathsf{dom}(\Sigma)$, $ge \subseteq \mathbb{S}$, $\mathbb{F} \cap \mathbb{S} = \emptyset$, and $\mathsf{closed}(\mathbb{S}, \Sigma)$, then $\mathsf{RC}(\mathbb{F}, \mathbb{S}, (\mathbb{k}, \Sigma), \bullet)$.

Here RC is defined as the largest relation such that, whenever $\mathsf{RC}(\mathbb{F}, \mathbb{S}, (\mathbb{k}, \Sigma), \beta)$, then all the following hold:

1. for all $\mathbb{k}'$, Σ' and Δ, if $\beta = \circ$ and $\mathbb{F} \vdash (\mathbb{k}, \Sigma') \underset{\Delta}{\overset{\tau}{\longmapsto}} (\mathbb{k}', \Sigma')$, then $\mathsf{G}(\Delta, \mathbb{F}, \mathbb{S})$ and $\mathsf{RC}(\mathbb{F}, \mathbb{S}, (\mathbb{k}', \Sigma'), \circ)$.
2. for all $\mathbb{k}'$ and ι, if $\beta = \circ$ and $\iota \neq \tau$ and $\mathbb{F} \vdash (\mathbb{k}, \Sigma) \underset{\mathsf{emp}}{\overset{\iota}{\longmapsto}} (\mathbb{k}', \Sigma)$, then there exists $\mathbb{S}'$ such that $\mathbb{S}' = \mathsf{cl}(\mathbb{S}, \Sigma)$, and $\mathbb{S}' - \mathbb{S} \subseteq \mathbb{F}$, and $\mathsf{RC}(\mathbb{F}, \mathbb{S}', (\mathbb{k}', \Sigma), \bullet)$.
3. for all Σ' and $\mathbb{S}'$, if $\beta = \bullet$ and $\mathsf{R}(\Sigma, \Sigma', \mathbb{F}, \mathbb{S})$ and $\mathbb{S}' = \mathsf{cl}(\mathbb{S}, \Sigma')$ and $(\mathbb{S}' - \mathbb{S}) \cap \mathbb{F} = \emptyset$, then $\mathsf{RC}(\mathbb{F}, \mathbb{S}', (\mathbb{k}, \Sigma'), \circ)$.

Lemma 48 (Compositionality).
For any $\mathtt{f}_1, \ldots, \mathtt{f}_n$, φ, $\Gamma = \{(sl_1, ge_1, \gamma_1), \ldots, (sl_m, ge_m, \gamma_m)\}$, $\Pi = \{(tl_1, ge'_1, \pi_1), \ldots, (tl_m, ge'_m, \pi_m)\}$, if

$$\forall i \in \{1, \ldots, m\}.\ \mathsf{wd}(sl_i) \wedge \mathsf{wd}(tl_i) \wedge \mathsf{ReachClose}(sl_i, ge_i, \gamma_i) \wedge (sl_i, ge_i, \gamma_i) \preccurlyeq_{\varphi} (tl_i, ge'_i, \pi_i)\ ,$$

then

$$\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_n \preccurlyeq_{\varphi} \mathbf{let}\ \Pi\ \mathbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_n\ .$$

Flip of the non-preemptive global simulation.

Definition 49 (Whole-Program Upward Simulation).
Let $\hat{\mathbb{P}} = \mathbf{let}\,\Gamma\,\mathbf{in}\ \mathtt{f}_1|\ldots|\mathtt{f}_m$, $\hat{P} = \mathbf{let}\,\Pi\,\mathbf{in}\ \mathtt{f}_1|\ldots|\mathtt{f}_m$, $\Gamma = \{(sl_1, ge_1, \gamma_1), \ldots, (sl_m, ge_m, \gamma_m)\}$ and $\Pi = \{(tl_1, ge'_1, \pi_1), \ldots, (tl_m, ge'_m, \pi_m)\}$.
$\hat{P} \leqslant_{\varphi} \hat{\mathbb{P}}$ iff $\forall i \in \{1, \ldots, m\}.\ \varphi\langle\!\langle ge_i \rangle\!\rangle = ge'_i$, and

for any $\Sigma, \sigma, \mu, \widehat{W}$, if $\mathsf{initM}(\varphi, \bigcup ge_i, \Sigma, \sigma)$, $\mu = (\mathsf{dom}(\Sigma), \mathsf{dom}(\sigma), \varphi)$, and $(\hat{P}, \sigma) \overset{load}{:\Longrightarrow} \widehat{W}$, then there exists $\widehat{\mathbb{W}}$, $i \in \mathtt{index}$ such that $(\hat{\mathbb{P}}, \Sigma) \overset{load}{:\Longrightarrow} \widehat{\mathbb{W}}$ and $(\widehat{W}, \mathsf{emp}) \leqslant^i_{\mu} (\widehat{\mathbb{W}}, \mathsf{emp})$.

Here we define $(\widehat{W}, \delta_0) \leqslant^i_{\mu} (\widehat{\mathbb{W}}, \Delta_0)$ as the largest relation such that whenever $(\widehat{W}, \delta_0) \leqslant^i_{\mu} (\widehat{\mathbb{W}}, \Delta_0)$, then the following are true:

1. $\widehat{\mathbb{W}}.\mathtt{t} = \widehat{W}.\mathtt{t}$, and $\widehat{\mathbb{W}}.\mathbb{d} = \widehat{W}.\mathbb{d}$, and $\mathsf{wf}(\mu, \widehat{\mathbb{W}}.\Sigma, \widehat{W}.\sigma)$;
2. $\forall \widehat{W}', \delta$. if $\widehat{W} \overset{\tau}{\underset{\delta}{:\Rightarrow}} \widehat{W}'$, then one of the following holds:
   a. there exists j such that
      i. $j < i$, and
      ii. $(\widehat{W}', \delta_0 \cup \delta) \leqslant^j_\mu (\widehat{\mathbb{W}}, \Delta_0)$;
   b. there exist $\widehat{\mathbb{W}}'$, Δ and j such that
      i. $\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^+ \widehat{\mathbb{W}}'$, and
      ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta, \mathsf{curF}(\widehat{W})))$, and
      iii. $(\widehat{W}', \delta_0 \cup \delta) \leqslant^j_\mu (\widehat{\mathbb{W}}', \Delta_0 \cup \Delta)$;
3. $\forall T', \mathbb{d}', \sigma', o$. if $o \neq \tau$ and $\exists \mathtt{t}'$. $\widehat{W} \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} (T', \mathtt{t}', \mathbb{d}', \sigma')$, then there exist $\widehat{\mathbb{W}}', \Delta, \mu', \mathbb{T}', \Sigma'$ and j such that for any $\mathtt{t}' \in \mathsf{dom}(T')$, we have
   a. $\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^* \widehat{\mathbb{W}}'$, and $\widehat{\mathbb{W}}' \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} (\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma')$, and
   b. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, and
   c. $((T', \mathtt{t}', \mathbb{d}', \sigma'), \mathsf{emp}) \leqslant^j_{\mu'} ((\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma'), \mathsf{emp})$;
4. if $\widehat{W} \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **done**, then there exist $\widehat{\mathbb{W}}'$ and Δ such that
   a. $\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^* \widehat{\mathbb{W}}'$, and $\widehat{\mathbb{W}}' \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **done**, and
   b. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0, \mathsf{curF}(\widehat{W})))$.

Lemma 50 (Flip).
For any $\mathtt{f}_1, \ldots, \mathtt{f}_n, \varphi, \Gamma = \{(sl_1, ge_1, \gamma_1), \ldots, (sl_m, ge_m, \gamma_m)\}$, $\Pi = \{(tl_1, ge'_1, \pi_1), \ldots, (tl_m, ge'_m, \pi_m)\}$, if $\forall i.\ \mathsf{det}(tl_i)$ and

$$\textbf{let } \Gamma \textbf{ in } \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m \preccurlyeq_\varphi \textbf{let } \Pi \textbf{ in } \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m\ ,$$

then

$$\textbf{let } \Pi \textbf{ in } \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m \leqslant_\varphi \textbf{let } \Gamma \textbf{ in } \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m\ .$$

NPDRF preservation.

Lemma 51 (NPDRF Preservation).
For any $\hat{\mathbb{P}}, \hat{P}, \varphi, \Sigma$ and σ, if $\hat{P} \leqslant_\varphi \hat{\mathbb{P}}$, $\mathsf{initM}(\varphi, \mathsf{GE}(\hat{\mathbb{P}}.\Gamma), \Sigma, \sigma)$, and $\mathsf{NPDRF}(\hat{\mathbb{P}}, \Sigma)$, then $\mathsf{NPDRF}(\hat{P}, \sigma)$.

E.2 Proof of Transitivity of Simulation (Lemma 45)

For all $\mathtt{f}, \mathbb{k}, \Sigma, \sigma, \mathbb{F}, F$, and $\mathbb{S} = \mathsf{dom}(\Sigma)$, and $S = \mathsf{dom}(\sigma)$, and $\mu = (\mathbb{S}, S, (\varphi' \circ \varphi)|_{\mathbb{S}})$, if $sl.\mathsf{InitCore}(\gamma, \mathtt{f}) = \mathbb{k}$, $(\varphi' \circ \varphi)\langle\!\langle ge\rangle\!\rangle = ge'$, $\mathsf{initM}(\varphi' \circ \varphi, ge, \Sigma, \sigma)$, and $\mathbb{F} \cap \mathsf{dom}(\Sigma) = F \cap \mathsf{dom}(\sigma) = \emptyset$, we know $S = (\varphi' \circ \varphi)\{\mathbb{S}\}$. Let $\mathbb{S}_1 = \varphi\{\mathbb{S}\}$ and $L_1 = \varphi\langle\!\langle \mathsf{locs}(\Sigma)\rangle\!\rangle$. We know

$$\mathbb{S}_1 \subseteq \mathsf{dom}(\varphi')\ ,\quad \varphi'\{\mathbb{S}_1\} = (\varphi' \circ \varphi)\{\mathbb{S}\} = \mathsf{dom}(\sigma)\ ,\quad \mathsf{locs}(\sigma) = \varphi'\langle\!\langle L_1\rangle\!\rangle\ .$$

Since $\mathsf{Inv}(\varphi' \circ \varphi, \Sigma, \sigma)$, we know there exists Σ_1 such that

$$\mathsf{dom}(\Sigma_1) = \mathbb{S}_1\ ,\quad \mathsf{locs}(\Sigma_1) = L_1\ ,\quad \mathsf{Inv}(\varphi, \Sigma, \Sigma_1)\ ,\quad \mathsf{Inv}(\varphi', \Sigma_1, \sigma)\ .$$

Since $\varphi\langle\!\langle ge\rangle\!\rangle = ge'$ and $\mathsf{dom}(\varphi' \circ \varphi) \subseteq \mathsf{dom}(\varphi)$, we know

$$ge \subseteq \mathsf{locs}(\Sigma)\ ,\quad \mathsf{dom}(\Sigma) = \mathbb{S} \subseteq \mathsf{dom}(\varphi)\ ,\quad ge' \subseteq \mathsf{locs}(\Sigma_1)\ ,\quad \mathsf{dom}(\Sigma_1) = \mathbb{S}_1 \subseteq \mathsf{dom}(\varphi')\ .$$

Pick $\mathbb{F}_1$ such that $\mathbb{F}_1 \cap \mathbb{S}_1 = \emptyset$. Let $\mu_1 = (\mathbb{S}, \mathbb{S}_1, \varphi|_{\mathbb{S}})$ and $\mu_2 = (\mathbb{S}_1, \varphi'\{\mathbb{S}_1\}, \varphi'|_{\mathbb{S}_1})$. From $(sl, ge, \gamma) \preccurlyeq_\varphi (sl_1, ge_1, \gamma_1)$, we know there exists $\mathbb{k}_1$ such that $sl_1.\mathsf{InitCore}(\gamma_1, \mathtt{f}) = \mathbb{k}_1$ and there exists $i_1 \in \mathtt{index}$ such that

$$(\mathbb{F}, (\mathbb{k}, \Sigma), \mathsf{emp}, \circ) \preccurlyeq^{i_1}_{\mu_1} (\mathbb{F}_1, (\mathbb{k}_1, \Sigma_1), \mathsf{emp}, \circ)\ .$$

Thus we know

$$\mathbb{S}_1 = \mathsf{cl}(\mathbb{S}_1, \Sigma_1)\ .$$

From $(sl_1, ge_1, \gamma_1) \preccurlyeq_{\varphi'} (tl, ge', \pi)$, we know there exists κ such that $tl.\mathsf{InitCore}(\pi)(\mathtt{f}) = \kappa$ and there exists $i_2 \in \mathtt{index}$ such that

$$(\mathbb{F}_1, (\mathbb{k}_1, \Sigma_1), \mathsf{emp}, \circ) \preccurlyeq^{i_2}_{\mu_2} (F, (\kappa, \sigma), \mathsf{emp}, \circ)\ .$$

By Lemma 52, let $i = (i_2, i_1)$, we know

$$(\mathbb{F}, (\mathbb{k}, \Sigma), \mathsf{emp}, \circ) \preccurlyeq^{i}_{\mu} (F, (\kappa, \sigma), \mathsf{emp}, \circ)\ .$$

Thus we are done.

Lemma 52. If

1. $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \beta) \preccurlyeq^{i_1}_{\mu_1} (\mathbb{F}_1, (\Bbbk_1, \Sigma_1), \Delta_{10}, \beta)$;
2. $(\mathbb{F}_1, (\Bbbk_1, \Sigma_1), \Delta_{10}, \beta) \preccurlyeq^{i_2}_{\mu_2} (F, (\kappa, \sigma), \delta_0, \beta)$;
3. $\mu_1 = (\mathbb{S}, \mathbb{S}_1, f_1)$, $\mu_2 = (\mathbb{S}_1, S, f_2)$, $\mu = (\mathbb{S}, S, f_2 \circ f_1)$, $i = (i_2, i_1)$;
4. if $\beta = \bullet$, then $\mathsf{Inv}(f_1, \Sigma, \Sigma_1)$ and $\mathsf{Inv}(f_2, \Sigma_1, \sigma)$.

then $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \beta) \preccurlyeq^{i}_{\mu} (F, (\kappa, \sigma), \delta_0, \beta)$.

Proof. By co-induction. From the premises, we know

$$\mathsf{wf}(\mu_1, (\Sigma, \mathbb{F}), (\Sigma_1, \mathbb{F}_1)), \quad \mathbb{S} \subseteq \mathsf{dom}(\Sigma), \quad \mathbb{S}_1 \subseteq \mathsf{dom}(\Sigma_1), \quad \mathsf{wf}(\mu_2, (\Sigma_1, \mathbb{F}_1), (\sigma, F)).$$

Thus we know

$$\begin{array}{lllll} \mathsf{injective}(f_1, \Sigma), & \mathbb{S} = \mathsf{dom}(f_1), & f_1\{\mathbb{S}\} \subseteq \mathbb{S}_1, & f_1\{\mathbb{S} \cap \mathbb{F}\} \subseteq \mathbb{F}_1, & f_1\{\mathbb{S} - \mathbb{F}\} \cap \mathbb{F}_1 = \emptyset, \\ \mathsf{injective}(f_2, \Sigma_1), & \mathbb{S}_1 = \mathsf{dom}(f_2), & f_2\{\mathbb{S}_1\} \subseteq S, & f_2\{\mathbb{S}_1 \cap \mathbb{F}_1\} \subseteq F, & f_2\{\mathbb{S}_1 - \mathbb{F}_1\} \cap F = \emptyset. \end{array}$$

From $\mathsf{injective}(f_1, \Sigma)$, $\mathsf{injective}(f_2, \Sigma_1)$ and $\mathsf{Inv}(f_1, \Sigma, \Sigma_1)$, we know

$$\mathsf{injective}(f_2 \circ f_1, \Sigma) .$$

Thus we know

$$\mathsf{wf}(\mu, (\Sigma, \mathbb{F}), (\sigma, F)) .$$

Also, since $\mathsf{Inv}(f_1, \Sigma, \Sigma_1)$ and $\mathsf{Inv}(f_2, \Sigma_1, \sigma)$, we know $\mathsf{Inv}(f_2 \circ f_1, \Sigma, \sigma)$. Also we need to prove the following:

1. $\forall \Bbbk', \Sigma', \Delta.$ if $\beta = \circ$ and $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\tau}{\underset{\Delta}{\longmapsto}} (\Bbbk', \Sigma')$ and $\mathsf{HGuar}(\mu, \Delta_0 \cup \Delta, \mathbb{F})$, then one of the following holds:
   a. there exists j such that
      i. $j < i$, and
      ii. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa, \sigma), \delta_0, \circ)$;
   b. or, there exist κ', σ', δ and j such that
      i. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{+} (\kappa', \sigma')$, and
      ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta, F))$, and
      iii. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa', \sigma'), \delta_0 \cup \delta, \circ)$.

   Proof. From $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \beta) \preccurlyeq^{i_1}_{\mu_1} (\mathbb{F}_1, (\Bbbk_1, \Sigma_1), \Delta_{10}, \beta)$, we know one of the following holds:
   (A) there exists j_1 such that
      (I) $j_1 < i_1$, and
      (II) $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j_1}_{\mu_1} (\mathbb{F}_1, (\Bbbk_1, \Sigma_1), \Delta_{10}, \circ)$;
   (B) or, there exist $\Bbbk_1'$, Σ_1', Δ_1 and j_1 such that
      (I) $\mathbb{F}_1 \vdash (\Bbbk_1, \Sigma_1) \overset{\tau}{\underset{\Delta_1}{\longmapsto}}{}^{+} (\Bbbk_1', \Sigma_1')$, and
      (II) $\mathsf{FPmatch}(\mu_1, (\Delta_0 \cup \Delta, \Sigma), (\Delta_{10} \cup \Delta_1, \mathbb{F}_1))$, and
      (III) $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j_1}_{\mu_1} (\mathbb{F}_1, (\Bbbk_1', \Sigma_1'), \Delta_{10} \cup \Delta_1, \circ)$.

   For case (A), let $j = (i_2, j_1)$. By the co-induction hypothesis, we know
   $$(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa, \sigma), \delta_0, \circ) \quad .$$
   Since $j_1 < i_1$, we know
   $$j < i \quad .$$
   Thus we are done.
   For case (B), from $\mathsf{FPmatch}(\mu_1, (\Delta_0 \cup \Delta, \Sigma), (\Delta_{10} \cup \Delta_1, \mathbb{F}_1))$, we know
   $$\mathsf{G}(\Delta_{10} \cup \Delta_1, \mathbb{F}_1, \mathsf{LShare}(\mu_1, \Sigma)) .$$
   Thus we know
   $$\mathsf{HGuar}(\mu_2, \Delta_{10} \cup \Delta_1, \mathbb{F}_1) .$$
   From $(\mathbb{F}_1, (\Bbbk_1, \Sigma_1), \Delta_{10}, \beta) \preccurlyeq^{i_2}_{\mu_2} (F, (\kappa, \sigma), \delta_0, \beta)$, by Lemma 53, we know one of the following holds:
   (B1) there exists j_2 such that

i. $j_2 < i_2$, and

ii. $(\mathbb{F}_1, (\Bbbk'_1, \Sigma'_1), \Delta_{10} \cup \Delta_1, \circ) \preccurlyeq^{j_2}_{\mu_2} (F, (\kappa, \sigma), \delta_0, \circ)$;

(B2) or, there exist κ', σ', δ and j_2 such that

i. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{+} (\kappa', \sigma')$, and

ii. $\mathsf{FPmatch}(\mu_2, (\Delta_{10} \cup \Delta_1, \Sigma_1), (\delta_0 \cup \delta, F))$, and

iii. $(\mathbb{F}_1, (\Bbbk'_1, \Sigma'_1), \Delta_{10} \cup \Delta_1, \circ) \preccurlyeq^{j_2}_{\mu_2} (F, (\kappa', \sigma'), \delta_0 \cup \delta, \circ)$.

For case (B1), let $j = (j_2, j_1)$. By the co-induction hypothesis, we know

$$(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa, \sigma), \delta_0, \circ) \ .$$

Since $j_2 < i_2$, we know

$$j < i \ .$$

For case (B2), let $j = (j_2, j_1)$. By the co-induction hypothesis, we know

$$(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa', \sigma'), \delta_0 \cup \delta, \circ) \ .$$

From $\mathsf{FPmatch}(\mu_1, (\Delta_0 \cup \Delta, \Sigma), (\Delta_{10} \cup \Delta_1, \mathbb{F}_1))$ and $\mathsf{FPmatch}(\mu_2, (\Delta_{10} \cup \Delta_1, \Sigma_1), (\delta_0 \cup \delta, F))$, by Lemma 55, we know

$$\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta, F)) \ .$$

Thus we are done. □

2. $\forall \Bbbk', \iota$. if $\beta = \circ$, $\iota \neq \tau$ and $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\iota}{\underset{\mathsf{emp}}{\longmapsto}} (\Bbbk', \Sigma)$ and $\mathsf{HGuar}(\mu, \Delta_0, \mathbb{F})$, then there exist κ', δ, σ', κ', μ' and j such that

a. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{*} (\kappa', \sigma')$, and
   $F \vdash (\kappa', \sigma') \overset{\iota}{\underset{\mathsf{emp}}{\longmapsto}} (\kappa'', \sigma')$, and

b. $\mathsf{FPmatch}(\mu, (\Delta_0, \Sigma), (\delta_0 \cup \delta, F))$, and

c. $\mathsf{EvolveG}(\mu, \mu', \Sigma, \sigma', \mathbb{F}, F)$, and

d. $(\mathbb{F}, (\Bbbk', \Sigma), \mathsf{emp}, \bullet) \preccurlyeq^{j}_{\mu'} (F, (\kappa'', \sigma'), \mathsf{emp}, \bullet)$.

Proof. From $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \beta) \preccurlyeq^{i_1}_{\mu_1} (\mathbb{F}_1, (\Bbbk_1, \Sigma_1), \Delta_{10}, \beta)$, we know there exist $\Bbbk'_1$, Δ_1, Σ'_1, $\Bbbk'_1$, μ'_1 and j_1 such that

a. $\mathbb{F}_1 \vdash (\Bbbk_1, \Sigma_1) \overset{\tau}{\underset{\Delta_1}{\longmapsto}}{}^{*} (\Bbbk'_1, \Sigma'_1)$, and $\mathbb{F}_1 \vdash (\Bbbk'_1, \Sigma'_1) \overset{\iota}{\underset{\mathsf{emp}}{\longmapsto}} (\Bbbk''_1, \Sigma'_1)$, and

b. $\mathsf{FPmatch}(\mu_1, (\Delta_0, \Sigma), (\Delta_{10} \cup \Delta_1, \mathbb{F}_1))$, and

c. $\mathsf{EvolveG}(\mu_1, \mu'_1, \Sigma, \Sigma'_1, \mathbb{F}, \mathbb{F}_1)$, and

d. $(\mathbb{F}, (\Bbbk', \Sigma), \mathsf{emp}, \bullet) \preccurlyeq^{j_1}_{\mu'_1} (\mathbb{F}_1, (\Bbbk''_1, \Sigma'_1), \mathsf{emp}, \bullet)$.

From $\mathsf{FPmatch}(\mu_1, (\Delta_0, \Sigma), (\Delta_{10} \cup \Delta_1, \mathbb{F}_1))$, we know

$$\mathsf{G}(\Delta_{10} \cup \Delta_1, \mathbb{F}_1, \mathsf{LShare}(\mu_1, \Sigma)) \ .$$

Thus we know

$$\mathsf{HGuar}(\mu_2, \Delta_{10} \cup \Delta_1, \mathbb{F}_1) \ .$$

From $(\mathbb{F}_1, (\Bbbk_1, \Sigma_1), \Delta_{10}, \beta) \preccurlyeq^{i_2}_{\mu_2} (F, (\kappa, \sigma), \delta_0, \beta)$, by Lemma 54, we know there exist κ', δ, σ', κ', μ'_2 and j_2 such that

a. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{*} (\kappa', \sigma')$, and $F \vdash (\kappa', \sigma') \overset{\iota}{\underset{\mathsf{emp}}{\longmapsto}} (\kappa'', \sigma')$, and

b. $\mathsf{FPmatch}(\mu_2, (\Delta_{10} \cup \Delta_1, \Sigma_1), (\delta_0 \cup \delta, F))$, and

c. $\mathsf{EvolveG}(\mu_2, \mu'_2, \Sigma'_1, \sigma', \mathbb{F}_1, F)$, and

d. $(\mathbb{F}_1, (\Bbbk''_1, \Sigma'_1), \mathsf{emp}, \bullet) \preccurlyeq^{j_2}_{\mu'_2} (F, (\kappa'', \sigma'), \mathsf{emp}, \bullet)$.

Suppose $\mu'_1 = (\mathbb{S}', \mathbb{S}'_1, f'_1)$ and $\mu'_2 = (\mathbb{S}''_1, S', f'_2)$. Since $\mathsf{EvolveG}(\mu_1, \mu'_1, \Sigma, \Sigma'_1, \mathbb{F}, \mathbb{F}_1)$ and $\mathsf{EvolveG}(\mu_2, \mu'_2, \Sigma'_1, \sigma', \mathbb{F}_1, F)$, we know

$$\mathbb{S}'_1 = \mathsf{cl}(\mathbb{S}_1, \Sigma'_1) \quad \text{and} \quad \mathbb{S}''_1 = \mathsf{cl}(\mathbb{S}_1, \Sigma'_1) \ .$$

Thus $\mathbb{S}'_1 = \mathbb{S}''_1$. Let $\mu' = (\mathbb{S}', S', f'_2 \circ f'_1)$. Let $j = (j_2, j_1)$. By the co-induction hypothesis, we know

$$(\mathbb{F}, (\Bbbk', \Sigma), \mathsf{emp}, \bullet) \preccurlyeq^{j}_{\mu'} (F, (\kappa'', \sigma'), \mathsf{emp}, \bullet).$$

From $\mathsf{FPmatch}(\mu_1, (\Delta_0, \Sigma), (\Delta_{10} \cup \Delta_1, \mathbb{F}_1))$ and $\mathsf{FPmatch}(\mu_2, (\Delta_{10} \cup \Delta_1, \Sigma_1), (\delta_0 \cup \delta, F))$, by Lemma 55, we know

$$\mathsf{FPmatch}(\mu, (\Delta_0, \Sigma), (\delta_0 \cup \delta, F)) \ .$$

From $\mathsf{EvolveG}(\mu_1, \mu'_1, \Sigma, \Sigma'_1, \mathbb{F}, \mathbb{F}_1)$ and $\mathsf{EvolveG}(\mu_2, \mu'_2, \Sigma'_1, \sigma', \mathbb{F}_1, F)$, by Lemma 56, we know

$$\mathsf{EvolveG}(\mu, \mu', \Sigma, \sigma', \mathbb{F}, F) \ .$$

Thus we are done. □

3. $\forall \sigma', \Sigma', \mu'$, if $\beta = \bullet$ and $\mathsf{Rely}((\mu, \mu'), (\Sigma, \Sigma', \mathbb{F}), (\sigma, \sigma', F))$, then there exists j such that $(\mathbb{F}, (\Bbbk, \Sigma'), \mathsf{emp}, \circ) \preccurlyeq^{j}_{\mu'} (F, (\kappa, \sigma'), \mathsf{emp}, \circ)$.

Proof. Suppose $\mu' = (\mathbb{S}', S', f')$. Since $\mathsf{EvolveR}(\mu, \mu', \Sigma', \sigma', \mathbb{F}, F)$, we know

$$\mathsf{injective}(f', \Sigma') \;,\quad \mathbb{S}' = \mathsf{dom}(f') \;,\quad f'\{\mathbb{S}'\} \subseteq S' \;,\quad f'\{\mathbb{S}' \cap \mathbb{F}\} \subseteq F \;,\quad f'\{\mathbb{S}' - \mathbb{F}\} \cap F = \emptyset \;,$$
$$(f_2 \circ f_1) \subseteq f' \;,\quad \mathsf{Inv}(f', \Sigma', \sigma') \;,\quad \mathbb{S}' = \mathsf{cl}(\mathbb{S}, \Sigma') \subseteq \mathsf{dom}(\Sigma') \;,\quad S' = \mathsf{cl}(S, \sigma') \;,\quad (\mathbb{S}' - \mathbb{S}) \cap \mathbb{F} = \emptyset \;.$$

We first construct f_1'. Let $\mathbb{S}'' = \mathbb{S}' - \mathbb{S}$. Thus $\mathbb{S}'' \cap \mathbb{F} = \emptyset$. From $\mathbb{S}' = \mathsf{cl}(\mathbb{S}, \Sigma')$, we know $\mathbb{S}' = \mathbb{S} \uplus \mathbb{S}''$. Pick an arbitrary set of fresh blocks $\mathbb{S}_1''$ such that

$$|\mathbb{S}_1''| = |\mathbb{S}''| \;,\quad \mathbb{S}_1'' \cap \mathsf{dom}(\Sigma_1) = \emptyset \;,\quad \mathbb{S}_1'' \cap \mathbb{S}_1 = \emptyset \;,\quad \mathbb{S}_1'' \cap \mathbb{F}_1 = \emptyset \;.$$

Pick an arbitrary function fb_1 mapping blocks in $\mathbb{S}''$ to the blocks of $\mathbb{S}_1''$ such that

$$\mathsf{injective}(fb_1) \quad \text{and} \quad \mathsf{dom}(fb_1) = \mathbb{S}'' \quad \text{and} \quad \mathsf{range}(fb_1) = \mathbb{S}_1'' \;.$$

Define the function f_1'' as follows.

$$\forall b \in \mathbb{S}''.\ f_1''(b) = (fb_1(b), 0) \;.$$

Thus we know

$$\mathsf{injective}(f_1'') \quad \text{and} \quad \mathsf{dom}(f_1'') = \mathbb{S}'' \quad \text{and} \quad f_1''\{\mathbb{S}''\} = \mathbb{S}_1'' \;.$$

Let $f_1' = f_1 \cup f_1''$. Since $\mathsf{wf}(\mu_1, \Sigma, \mathbb{F}, \mathbb{F}_1)$, we know

$$\mathsf{injective}(f_1', \Sigma') \quad \text{and} \quad \mathsf{dom}(f_1') = \mathbb{S}' \quad \text{and} \quad f_1'\{\mathbb{S}'\} \subseteq \mathbb{S}_1 \cup \mathbb{S}_1'' \quad \text{and}$$
$$f_1'\{\mathbb{S}' \cap \mathbb{F}\} \subseteq \mathbb{F}_1 \quad \text{and} \quad f_1'\{\mathbb{S}' - \mathbb{F}\} \cap \mathbb{F}_1 = \emptyset \;.$$

Next we construct $\mathbb{S}_1'$. Let $\mathbb{S}_1' = \mathbb{S}_1 \cup \mathbb{S}_1''$.
Then we construct f_2'. Define a function f_2'' as follows:

$$\mathsf{dom}(f_2'') = \mathbb{S}_1'' \quad \text{and} \quad \forall b \in \mathbb{S}_1''.\ f_2''(b) = f'(fb_1^{-1}(b)) \;.$$

Since $f'\{\mathbb{S}'\} \subseteq S'$, we know

$$f_2''\{\mathbb{S}_1''\} = f'\{\mathbb{S}''\} \subseteq S' \;.$$

Let $f_2' = f_2 \cup f_2''$. Since $\mathsf{wf}(\mu_2, \Sigma_1, \mathbb{F}_1, F)$ and $f'\{\mathbb{S}' - \mathbb{F}\} \cap F = \emptyset$, we know

$$\mathsf{dom}(f_2') = \mathbb{S}_1' \quad \text{and} \quad f_2'\{\mathbb{S}_1'\} \subseteq S' \quad \text{and}$$
$$f_2'\{\mathbb{S}_1' \cap \mathbb{F}_1\} \subseteq F \quad \text{and} \quad f_2'\{\mathbb{S}_1' - \mathbb{F}_1\} \cap F = \emptyset \;.$$

Also we know

$$f_2' \circ f_1' = (f_2 \circ f_1) \cup (f_2'' \circ f_1'') = f' \;.$$

Finally we construct Σ_1'. Define

$\mathsf{dom}(\Sigma_1') = \mathsf{dom}(\Sigma_1) \cup \mathbb{S}_1''$

$\forall b_0 \in \mathsf{dom}(\Sigma_1) - \mathbb{S}_1.\ \mathsf{dom}(\Sigma_1'(b_0)) = \mathsf{dom}(\Sigma_1(b_0)) \qquad \forall i.\ \Sigma_1'(b_0)(i) = \Sigma_1(b_0)(i)$

$\forall b_{10} \in \mathbb{S}_1 - f_1\{\mathbb{S}\} - \mathbb{F}_1.\ \mathsf{dom}(\Sigma_1'(b_{10})) = \emptyset$

$\forall b_{11} \in f_1\{\mathbb{S}\} - \mathbb{F}_1.\ \mathsf{dom}(\Sigma_1'(b_{11})) = \{i \mid \exists b, n.\ f_1(b) = (b_{11}, n) \wedge (b, i - n) \in \mathsf{locs}(\Sigma')\}$

$$\forall i.\ \Sigma_1'(b_{11})(i) = \begin{cases} v' & \text{if } \exists b, n.\ f_1(b) = (b_{11}, n) \wedge \Sigma'(b)(i - n) = v' \notin \mathfrak{U} \\ v & \text{if } \exists b, n, b', i', b_1', n'.\ f_1(b) = (b_{11}, n) \wedge \Sigma'(b)(i - n) = (b', i') \wedge f_1'(b') = (b_1', n') \wedge v = (b_1', i' + n') \end{cases}$$

$\forall b_1 \in \mathbb{S}_1 \cap \mathbb{F}_1.\ \mathsf{dom}(\Sigma_1'(b_1)) = \{i \mid (b_1, i) \in (f_1\langle\!\langle \mathsf{locs}(\Sigma') \cap \lfloor\!\lfloor \mathbb{S} \rfloor\!\rfloor \rangle\!\rangle) \cup (\mathsf{locs}(\Sigma_1) - f_1\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor\!\lfloor \mathbb{S} \rfloor\!\rfloor \rangle\!\rangle)\}$

$\forall i.\ (b_1, i) \in f_1\langle\!\langle \mathsf{locs}(\Sigma') \cap \lfloor\!\lfloor \mathbb{S} \rfloor\!\rfloor \rangle\!\rangle \Longrightarrow$

$$\Sigma_1'(b_1)(i) = \begin{cases} v' & \text{if } \exists b, n.\ f_1\langle(b, n)\rangle = (b_1, i) \wedge \Sigma'(b)(n) = v' \notin \mathfrak{U} \\ v & \text{if } \exists b, n, b', i'.\ f_1\langle(b, n)\rangle = (b_1, i) \wedge \Sigma'(b)(n) = (b', i') \wedge v = f_1'\langle(b', i')\rangle \end{cases}$$

$\forall i.\ (b_1, i) \in \mathsf{locs}(\Sigma_1) - f_1\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor\!\lfloor \mathbb{S} \rfloor\!\rfloor \rangle\!\rangle \Longrightarrow \Sigma_1'(b_1)(i) = \Sigma_1(b_1)(i)$

$\forall b_2 \in \mathbb{S}_1''.\ \mathsf{dom}(\Sigma_1'(b_2)) = \mathsf{dom}(\Sigma'(fb_1^{-1}(b_2)))$

$$\forall i.\ \Sigma_1'(b_2)(i) = \begin{cases} v' & \text{if } \exists b_2'.\ fb_1^{-1}(b_2) = b_2' \wedge \Sigma'(b_2')(i) = v' \notin \mathfrak{U} \\ v & \text{if } \exists b_2', b', i', b, n.\ fb_1^{-1}(b_2) = b_2' \wedge \Sigma'(b_2')(i) = (b', i') \wedge f_1'(b') = (b, n) \wedge v = (b, i' + n) \end{cases}$$

Below we prove the following:

- $\mathsf{forward}(\Sigma_1, \Sigma_1')$.
  We only need to prove $\mathsf{locs}(\Sigma_1') \cap \lfloor\!\lfloor \mathsf{dom}(\Sigma_1) \rfloor\!\rfloor \subseteq \mathsf{locs}(\Sigma_1)$. By the definition of Σ_1', we only need to prove: for any b_1 and i, if $b_1 \in (f_1\{\mathbb{S}\} - \mathbb{F}_1) \cup (\mathbb{S}_1 \cap \mathbb{F}_1)$ and $i \in \mathsf{dom}(\Sigma_1'(b_1))$, then $(b_1, i) \in \mathsf{locs}(\Sigma_1)$.
  If $b_1 \in (f_1\{\mathbb{S}\} - \mathbb{F}_1)$, we know there exist b and n such that $f_1(b) = (b_1, n)$ and $(b, i - n) \in \mathsf{locs}(\Sigma')$. Since $b \in \mathbb{S} \subseteq \mathsf{dom}(\Sigma)$ and $\mathsf{locs}(\Sigma') \cap \lfloor\!\lfloor \mathsf{dom}(\Sigma) \rfloor\!\rfloor \subseteq \mathsf{locs}(\Sigma)$, we know $(b, i - n) \in \mathsf{locs}(\Sigma)$. Since $\mathsf{Inv}(f_1, \Sigma, \Sigma_1)$, we know $(b_1, i) \in \mathsf{locs}(\Sigma_1)$.

If $b_1 \in (\mathbb{S}_1 \cap \mathbb{F}_1)$, we know $(b_1, i) \in (f_1\langle\!\langle \mathsf{locs}(\Sigma') \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle) \cup (\mathsf{locs}(\Sigma_1) - f_1\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle)$. Since $\mathsf{locs}(\Sigma') \cap \llbracket \mathsf{dom}(\Sigma) \rrbracket \subseteq \mathsf{locs}(\Sigma)$, we know $f_1\langle\!\langle \mathsf{locs}(\Sigma') \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle \subseteq f_1\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle$. Since $\mathsf{injective}(f_1, \Sigma, \Sigma_1)$, we know $f_1\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle \subseteq \mathsf{locs}(\Sigma_1)$. Thus $(b_1, i) \in \mathsf{locs}(\Sigma_1)$.

- $\mathsf{injective}(f_2', \Sigma_1')$.
  We prove the following.
  - $\mathsf{injective}(f_2, \Sigma_1')$.
    From $\mathsf{injective}(f_2, \Sigma_1)$ and $\mathsf{forward}(\Sigma_1, \Sigma_1')$, we know $\mathsf{injective}(f_2, \Sigma_1')$.
  - $\mathsf{injective}(f_2'', \Sigma_1')$.
    For any b_1, b_2, i_1, i_2, b_1', b_2', n_1 and n_2, if $(b_1, i_1) \neq (b_2, i_2)$, $(b_1, i_1) \in \mathsf{locs}(\Sigma_1')$, $(b_2, i_2) \in \mathsf{locs}(\Sigma_1')$, $f_2''(b_1) = (b_1', n_1)$ and $f_2''(b_2) = (b_2', n_2)$, by the definition of f_2'', we know
    $$b_1 \in \mathbb{S}_1'', \qquad b_2 \in \mathbb{S}_1'', \qquad f_2''(b_1) = f'(fb_1^{-1}(b_1)), \qquad f_2''(b_2) = f'(fb_1^{-1}(b_2)).$$
    Also by the definition of Σ_1', we know
    $$\mathsf{dom}(\Sigma_1'(b_1)) = \mathsf{dom}(\Sigma'(fb_1^{-1}(b_1))) \quad \text{and} \quad \mathsf{dom}(\Sigma_1'(b_2)) = \mathsf{dom}(\Sigma'(fb_1^{-1}(b_2))).$$
    Thus
    $$(fb_1^{-1}(b_1), i_1) \in \mathsf{locs}(\Sigma') \quad \text{and} \quad (fb_1^{-1}(b_2), i_2) \in \mathsf{locs}(\Sigma').$$
    Since $\mathsf{injective}(f', \Sigma')$, we know $b_1' \neq b_2'$ or $i_1 + n_1 \neq i_2 + n_2$. Thus $\mathsf{injective}(f_2'', \Sigma_1')$.
  - $\forall b_1, b_2, i_1, i_2, b_1', b_2', n_1, n_2.\ (b_1, i_1) \in \mathsf{locs}(\Sigma_1') \wedge (b_2, i_2) \in \mathsf{locs}(\Sigma_1') \wedge f_2(b_1) = (b_1', n_1) \wedge f_2''(b_2) = (b_2', n_2) \implies (b_1' \neq b_2') \vee (i_1 + n_1 \neq i_2 + n_2)$.
    Since $b_1 \in \mathbb{S}_1$, we consider three cases:
    * $b_1 \in \mathbb{S}_1 - f_1\{\mathbb{S}\} - \mathbb{F}_1$. Since $\mathsf{dom}(\Sigma_1'(b_1)) = \emptyset$, the above holds trivially.
    * $b_1 \in f_1\{\mathbb{S}\} - \mathbb{F}_1$.
      By the definition of f_2'', we know $b_2 \in \mathbb{S}_1''$ and $f_2''(b_2) = f'(fb_1^{-1}(b_2))$. Also by the definition of Σ_1', we know
      $$\mathsf{dom}(\Sigma_1'(b_1)) = \{i \mid \exists b, n.\ f_1(b) = (b_1, n) \wedge (b, i - n) \in \mathsf{locs}(\Sigma')\} \quad \text{and} \quad \mathsf{dom}(\Sigma_1'(b_2)) = \mathsf{dom}(\Sigma'(fb_1^{-1}(b_2))).$$
      Since $f_2 \circ f_1 \subseteq f'$, we know there exist b and n such that $f_1(b) = (b_1, n)$, $(b, i_1 - n) \in \mathsf{locs}(\Sigma')$ and $f'(b) = (b_1', n_1 + n)$. Also $(fb_1^{-1}(b_2), i_2) \in \mathsf{locs}(\Sigma')$. Since $\mathsf{injective}(f', \Sigma')$, we know $b_1' \neq b_2'$ or $i_1 + n_1 \neq i_2 + n_2$.
    * $b_1 \in \mathbb{S}_1 \cap \mathbb{F}_1$.
      Since $f_2\{\mathbb{S}_1 \cap \mathbb{F}_1\} \subseteq F$, we know $b_1' \in F$. By the definition of f_2'', we know $b_2 \in \mathbb{S}_1''$ and $f_2''(b_2) = f'(fb_1^{-1}(b_2))$. Since $\mathbb{S}'' \cap \mathbb{F} = \emptyset$, we know $fb_1^{-1}(b_2) \notin \mathbb{F}$. Since $f'\{\mathbb{S}' - \mathbb{F}\} \cap F = \emptyset$, we know $b_2' \notin F$. Thus $b_1' \neq b_2'$.

  From $\mathsf{injective}(f_2''')$, and the definition of f_2', we are done.
- $\sigma \overset{\llbracket F \rrbracket - f_2\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \llbracket \mathbb{S}_1 \rrbracket \rangle\!\rangle}{=\!=\!=} \sigma'$.
  Since $f_1\{\mathbb{S}\} \subseteq \mathbb{S}_1$ and $\mathsf{Inv}(f_1, \Sigma, \Sigma_1)$, we know $f_1\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle \subseteq \mathsf{locs}(\Sigma_1) \cap \llbracket \mathbb{S}_1 \rrbracket$. Thus $\llbracket F \rrbracket - f_2\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \llbracket \mathbb{S}_1 \rrbracket \rangle\!\rangle \subseteq \llbracket F \rrbracket - (f_2 \circ f_1)\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle$. From $\sigma \overset{\llbracket F \rrbracket - (f_2 \circ f_1)\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle}{=\!=\!=} \sigma'$, we are done.
- $\Sigma_1 \overset{\llbracket \mathbb{F}_1 - \mathbb{S}_1 \rrbracket}{=\!=\!=} \Sigma_1'$ and $\Sigma_1 \overset{\llbracket \mathbb{F}_1 \rrbracket - f_1\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle}{=\!=\!=} \Sigma_1'$.
  Since $f_1\{\mathbb{S}\} \subseteq \mathbb{S}_1$, we know $\llbracket \mathbb{F}_1 - \mathbb{S}_1 \rrbracket \subseteq \llbracket \mathbb{F}_1 \rrbracket - f_1\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle$. Thus we only need to prove $\Sigma_1 \overset{\llbracket \mathbb{F}_1 \rrbracket - f_1\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle}{=\!=\!=} \Sigma_1'$. By the above definition of Σ_1', we are done.
- $\mathsf{Inv}(f_2', \Sigma_1', \sigma')$.
  For any b_1, i, b_1', i'. if $(b_1, i) \in \mathsf{locs}(\Sigma_1')$ and $f_2'\langle(b_1, i)\rangle = (b_1', i')$, by the definition of Σ_1', we consider the following cases:
  - $b_1 \in f_1\{\mathbb{S}\} - \mathbb{F}_1$. Thus there exist b and n such that $f_1(b) = (b_1, n)$, $(b, i - n) \in \mathsf{locs}(\Sigma')$ and $\Sigma'(b)(i - n) \overset{f_1'}{\hookrightarrow} \Sigma_1'(b_1)(i)$.
    Since $f_2'\langle(b_1, i)\rangle = (b_1', i')$, we know
    $$(f_2' \circ f_1)\langle(b, i - n)\rangle = (b_1', i').$$
    From $\mathsf{Inv}(f', \Sigma', \sigma')$, we know $(b_1', i') \in \mathsf{locs}(\sigma')$ and $\Sigma'(b)(i - n) \overset{f'}{\hookrightarrow} \sigma'(b_1')(i')$. Since $f_2' \circ f_1' = f'$, we know $\Sigma_1'(b_1)(i) \overset{f_2'}{\hookrightarrow} \sigma'(b_1')(i')$.

- $b_1 \in \mathbb{S}_1 \cap \mathbb{F}_1$ and $(b_1, i) \in f_1\langle\!\langle \mathsf{locs}(\Sigma') \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle$. Thus by the definition of Σ'_1, we know there exist b and n such that $f_1\langle(b, n)\rangle = (b_1, i)$ and $\Sigma'(b)(n) \overset{f'_1}{\hookrightarrow} \Sigma'_1(b_1)(i)$. Since $f'_2\langle(b_1, i)\rangle = (b'_1, i')$ and $f'_2 \circ f'_1 = f'$, we know $f'\langle(b, n)\rangle = (b'_1, i')$. From $\mathsf{Inv}(f', \Sigma', \sigma')$, we know $(b'_1, i') \in \mathsf{locs}(\sigma')$ and $\Sigma'(b)(n) \overset{f'}{\hookrightarrow} \sigma'(b'_1)(i')$. Thus $\Sigma'_1(b_1)(i) \overset{f'_2}{\hookrightarrow} \sigma'(b'_1)(i')$.
- $b_1 \in \mathbb{S}_1 \cap \mathbb{F}_1$ and $(b_1, i) \in \mathsf{locs}(\Sigma_1) - f_1\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle$. Thus by the definition of Σ'_1, we know $\Sigma'_1(b_1)(i) = \Sigma_1(b_1)(i)$. Since $b_1 \in \mathbb{S}_1$, we know $f_2\langle(b_1, i)\rangle = (b'_1, i')$. Since $\mathsf{Inv}(f_2, \Sigma_1, \sigma)$, we know $(b'_1, i') \in \mathsf{locs}(\sigma)$ and $\Sigma_1(b_1)(i) \overset{f_2}{\hookrightarrow} \sigma(b'_1)(i')$. From $\sigma \xlongequal{\llbracket F \rrbracket - (f_2 \circ f_1)\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S} \rrbracket \rangle} \sigma'$, we know $(b'_1, i') \in \mathsf{locs}(\sigma')$ and $\sigma(b'_1)(i') = \sigma'(b'_1)(i')$. Thus $\Sigma'_1(b_1)(i) \overset{f'_2}{\hookrightarrow} \sigma'(b'_1)(i')$.
- $b_1 \in \mathbb{S}''_1$. Thus by the definition of Σ'_1, we know there exists b such that $fb_1^{-1}(b_1) = b$ and $\Sigma'(b)(i) \overset{f'_1}{\hookrightarrow} \Sigma'_1(b_1)(i)$. By the definition of f''_2, we know $f'(b) = (b'_1, i' - i)$. From $\mathsf{Inv}(f', \Sigma', \sigma')$, we know $(b'_1, i') \in \mathsf{locs}(\sigma')$ and $\Sigma'(b)(i) \overset{f'}{\hookrightarrow} \sigma'(b'_1)(i')$. Since $f'_2 \circ f'_1 = f'$, we know $\Sigma'_1(b_1)(i) \overset{f'_2}{\hookrightarrow} \sigma'(b'_1)(i')$.

Thus we are done.

- $\mathsf{Inv}(f'_1, \Sigma', \Sigma'_1)$.
  For any b, i, b_1, i_1. if $(b, i) \in \mathsf{locs}(\Sigma')$ and $f'_1\langle(b, i)\rangle = (b_1, i_1)$, we know $b \in \mathbb{S}'$. Consider the following cases:
  - $b \in \mathbb{S} - \mathbb{F}$. Thus $f_1(b) = (b_1, i_1 - i)$. Since $f_1\{\mathbb{S} - \mathbb{F}\} \cap \mathbb{F}_1 = \emptyset$, we know $b_1 \in f_1\{\mathbb{S}\} - \mathbb{F}_1$. By the definition of Σ'_1, we know $(b_1, i_1) \in \mathsf{locs}(\Sigma'_1)$ and $\Sigma'(b)(i) \overset{f'_1}{\hookrightarrow} \Sigma'_1(b_1)(i_1)$.
  - $b \in \mathbb{S} \cap \mathbb{F}$. Thus $f_1(b) = (b_1, i_1 - i)$. Since $f_1\{\mathbb{S} \cap \mathbb{F}\} \subseteq \mathbb{F}_1$, we know $b_1 \in \mathbb{S}_1 \cap \mathbb{F}_1$. Also $(b_1, i_1) \in f_1\langle\!\langle \mathsf{locs}(\Sigma') \cap \llbracket \mathbb{S} \rrbracket \rangle\!\rangle$. By the definition of Σ'_1, and since $\mathsf{injective}(f'_1, \Sigma')$, we know $(b_1, i_1) \in \mathsf{locs}(\Sigma'_1)$ and $\Sigma'(b)(i) \overset{f'_1}{\hookrightarrow} \Sigma'_1(b_1)(i_1)$.
  - $b \in \mathbb{S}''$. Thus $f''_1(b) = (b_1, i_1 - i)$. By the definition of f''_1, we know $i_1 = i$ and $fb_1(b) = b_1 \in \mathbb{S}''_1$. By the definition of Σ'_1, we know $(b_1, i_1) \in \mathsf{locs}(\Sigma'_1)$ and $\Sigma'(b)(i) \overset{f'_1}{\hookrightarrow} \Sigma'_1(b_1)(i_1)$.
- $\mathbb{S}'_1 \subseteq \mathsf{cl}(\mathbb{S}_1, \Sigma'_1)$.
  For any $b \in \mathbb{S}'_1$, we consider the following cases:
  - $b \in \mathbb{S}_1$. We know $b \in \mathsf{cl}_0(\mathbb{S}_1, \Sigma'_1)$.
  - $b \in \mathbb{S}''_1$. Thus there exists b' such that $b' \in \mathbb{S}''$, $b = fb_1(b')$ and $f''_1(b') = (b, 0)$. Since $\mathbb{S}' = \mathsf{cl}(\mathbb{S}, \Sigma')$, we know there exists k such that
  $$b' \in \mathsf{cl}_{k+1}(\mathbb{S}, \Sigma') .$$
  We need to prove
  $$\forall b', b.\ b' \in \mathsf{cl}_{k+1}(\mathbb{S}, \Sigma') \wedge f'_1(b') = (b, \_) \implies b \in \mathsf{cl}_{k+1}(\mathbb{S}_1, \Sigma'_1) \tag{E.1}$$
  By induction over k.
    * $k = 0$. Thus there exist b_0, n_0 and n' such that $b_0 \in \mathbb{S}$ and $\Sigma'(b_0)(n_0) = (b', n')$. Then we know there exist b_1 and n_1 such that $f_1\langle(b_0, n_0)\rangle = (b_1, n_1)$ and $b_1 \in \mathbb{S}_1$. Since $\mathsf{Inv}(f'_1, \Sigma', \Sigma'_1)$, we know
    $$\Sigma'(b_0)(n_0) \overset{f'_1}{\hookrightarrow} \Sigma'_1(b_1)(n_1) .$$
    Thus there exists n such that
    $$\Sigma'_1(b_1)(n_1) = f'_1\langle(b', n')\rangle = (b, n) .$$
    Thus $b \in \mathsf{cl}_1(\mathbb{S}_1, \Sigma'_1)$.
    * $k = m + 1$. Thus there exist b_0, n_0 and n' such that $b_0 \in \mathsf{cl}_{m+1}(\mathbb{S}, \Sigma')$ and $\Sigma'(b_0)(n_0) = (b', n')$. Then we know there exist b_1 and n_1 such that $f'_1\langle(b_0, n_0)\rangle = (b_1, n_1)$. By the induction hypothesis, we know $b_1 \in \mathsf{cl}_{m+1}(\mathbb{S}_1, \Sigma'_1)$. Since $\mathsf{Inv}(f'_1, \Sigma', \Sigma'_1)$, we know
    $$\Sigma'(b_0)(n_0) \overset{f'_1}{\hookrightarrow} \Sigma'_1(b_1)(n_1) .$$
    Thus there exists n such that
    $$\Sigma'_1(b_1)(n_1) = f'_1\langle(b', n')\rangle = (b, n) .$$
    Thus $b \in \mathsf{cl}_{k+1}(\mathbb{S}_1, \Sigma'_1)$.
- $\mathsf{cl}(\mathbb{S}_1, \Sigma'_1) \subseteq \mathbb{S}'_1$.

We need to prove: for any k, $\mathsf{cl}_k(\mathbb{S}_1, \Sigma'_1) \subseteq \mathbb{S}'_1$. By induction over k.

- $k = 0$. We know $\mathsf{cl}_0(\mathbb{S}_1, \Sigma'_1) = \mathbb{S}_1 \subseteq \mathbb{S}'_1$.
- $k = m + 1$. By the induction hypothesis, we know
$$\mathsf{cl}_{m+1}(\mathbb{S}_1, \Sigma'_1) \subseteq \{b' \mid \exists b, i, i'. (b \in \mathbb{S}'_1) \wedge \Sigma'_1(b)(i) = (b', i')\}$$
We only need to prove the following:
$$\forall b, i, b', i'. (b \in \mathbb{S}'_1) \wedge \Sigma'_1(b)(i) = (b', i') \implies b' \in \mathbb{S}'_1$$
By the definition of Σ'_1, we consider the following cases:
  * $b \in f_1\{\mathbb{S}\} - \mathbb{F}_1$. Thus there exist b_0 and i_0 such that $f_1\langle(b_0, i_0)\rangle = (b, i)$, $(b_0, i_0) \in \mathsf{locs}(\Sigma')$ and $\Sigma'(b_0)(i_0) \overset{f'_1}{\hookrightarrow} \Sigma'_1(b)(i)$. Thus there exist b'_0 and i'_0 such that $\Sigma'(b_0)(i_0) = (b'_0, i'_0)$ and $f'_1\langle(b'_0, i'_0)\rangle = (b', i')$. Thus $b' \in \mathbb{S}'_1$.
  * $b \in \mathbb{S}_1 \cap \mathbb{F}_1$ and $(b, i) \in f_1\langle\!\langle \mathsf{locs}(\Sigma') \cap \lfloor\!\lfloor \mathbb{S} \rfloor\!\rfloor \rangle\!\rangle$. The same as the earlier case.
  * $b \in \mathbb{S}_1 \cap \mathbb{F}_1$ and $(b, i) \in \mathsf{locs}(\Sigma_1) - f_1\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor\!\lfloor \mathbb{S} \rfloor\!\rfloor \rangle\!\rangle$. Thus $\Sigma'_1(b)(i) = \Sigma_1(b)(i)$. Since $\mathbb{S}_1 = \mathsf{cl}(\mathbb{S}_1, \Sigma_1)$, we know $b' \in \mathbb{S}_1$.
  * $b \in \mathbb{S}''_1$. Thus there exists b_0 such that $fb_1^{-1}(b) = b_0$ and $\Sigma'(b_0)(i) \overset{f'_1}{\hookrightarrow} \Sigma'_1(b)(i)$. Thus there exist b'_0 and i'_0 such that $\Sigma'(b_0)(i) = (b'_0, i'_0)$ and $f'_1\langle(b'_0, i'_0)\rangle = (b', i')$. Thus $b' \in \mathbb{S}'_1$.

Let $\mu'_1 = (\mathbb{S}', \mathbb{S}'_1, f'_1)$ and $\mu'_2 = (\mathbb{S}'_1, S', f'_2)$. Thus we know
$$\mathsf{EvolveR}(\mu_1, \mu'_1, \Sigma', \Sigma'_1, \mathbb{F}, \mathbb{F}_1) \quad \text{and} \quad \mathsf{EvolveR}(\mu_2, \mu'_2, \Sigma'_1, \sigma', \mathbb{F}_1, F) \quad \text{and} \quad \mu' = (\mathbb{S}', S', f'_2 \circ f'_1) .$$
Thus from $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \beta) \preccurlyeq^{i_1}_{\mu_1} (\mathbb{F}_1, (\Bbbk_1, \Sigma_1), \Delta_{10}, \beta)$ and $(\mathbb{F}_1, (\Bbbk_1, \Sigma_1), \Delta_{10}, \beta) \preccurlyeq^{i_2}_{\mu_2} (F, (\kappa, \sigma), \delta_0, \beta)$, we know there exists j_1 and j_2 such that
$$(\mathbb{F}, (\Bbbk, \Sigma'), \mathsf{emp}, \circ) \preccurlyeq^{j_1}_{\mu'_1} (\mathbb{F}_1, (\Bbbk_1, \Sigma'_1), \mathsf{emp}, \circ) \quad \text{and} \quad (\mathbb{F}_1, (\Bbbk_1, \Sigma'_1), \mathsf{emp}, \circ) \preccurlyeq^{j_2}_{\mu'_2} (F, (\kappa, \sigma'), \mathsf{emp}, \circ) .$$
Let $j = (j_2, j_1)$. By the co-induction hypothesis, we know
$$(\mathbb{F}, (\Bbbk, \Sigma'), \mathsf{emp}, \circ) \preccurlyeq^{j}_{\mu'} (F, (\kappa, \sigma'), \mathsf{emp}, \circ) .$$
Thus we are done. □

Thus we have finished the proof. □

Lemma 53. If $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \circ) \preccurlyeq^{i}_{\mu} (F, (\kappa, \sigma), \delta_0, \circ)$ and $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\tau}{\underset{\Delta}{\longmapsto}}{}^{n+1} (\Bbbk', \Sigma')$ and $\mathsf{HGuar}(\mu, \Delta_0 \cup \Delta, \mathbb{F})$, then one of the following holds:

1. there exists j such that
   a. $j < i$, and
   b. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa, \sigma), \delta_0, \circ)$;
2. or, there exist κ', σ', δ and j such that
   a. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{+} (\kappa', \sigma')$, and
   b. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta, F))$, and
   c. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa', \sigma'), \delta_0 \cup \delta, \circ)$.

Proof. By induction over n.

1. $n = 0$. Trivial.
2. $n = m + 1$. We know there exists $\Bbbk''$, Σ'', Δ' and Δ'' such that
$$\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\tau}{\underset{\Delta'}{\longmapsto}} (\Bbbk'', \Sigma'') , \quad \mathbb{F} \vdash (\Bbbk'', \Sigma'') \overset{\tau}{\underset{\Delta''}{\longmapsto}}{}^{m+1} (\Bbbk', \Sigma') , \quad \Delta = \Delta' \cup \Delta'' .$$
Since $\mathsf{HGuar}(\mu, \Delta_0 \cup \Delta, \mathbb{F})$, we know
$$\mathsf{HGuar}(\mu, \Delta_0 \cup \Delta', \mathbb{F}) .$$
Since $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \circ) \preccurlyeq^{i}_{\mu} (F, (\kappa, \sigma), \delta_0, \circ)$, we know one of the following holds:
   (a) there exists i' such that
      i. $i' < i$, and
      ii. $(\mathbb{F}, (\Bbbk'', \Sigma''), \Delta_0 \cup \Delta', \circ) \preccurlyeq^{i'}_{\mu} (F, (\kappa, \sigma), \delta_0, \circ)$;
   (b) or, there exist κ'', σ'', δ' and i' such that
      i. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta'}{\longmapsto}}{}^{+} (\kappa'', \sigma'')$, and
      ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta', \Sigma), (\delta_0 \cup \delta', F))$, and

iii. $(\mathbb{F}, (\Bbbk'', \Sigma''), \Delta_0 \cup \Delta', \circ) \preccurlyeq^{i'}_{\mu} (F, (\kappa'', \sigma''), \delta_0 \cup \delta', \circ)$.

For case (a), by the induction hypothesis, we know one of the following holds:

(a1) there exists j such that
  i. $j < i'$, and
  ii. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa, \sigma), \delta_0, \circ)$;

(a2) or, there exist κ', σ', δ and j such that
  i. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\mapsto}}{}^{+} (\kappa', \sigma')$, and
  ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma''), (\delta_0 \cup \delta, F))$, and
  iii. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa', \sigma'), \delta_0 \cup \delta \circ)$.

Since $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\tau}{\underset{\Delta'}{\mapsto}} (\Bbbk'', \Sigma'')$, we know $\mathsf{forward}(\Sigma, \Sigma'')$ and $(\mathsf{locs}(\Sigma) - \mathsf{locs}(\Sigma'')) \subseteq \Delta'.ws \cap \Delta'.rs$. Thus, if $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma''), (\delta_0 \cup \delta, F))$, by Lemma 57, then

$$\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta, F)) .$$

Thus we are done.

For case (b), by the induction hypothesis, we know one of the following holds:

(b1) there exists j such that
  i. $j < i'$, and
  ii. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa'', \sigma''), \delta_0 \cup \delta', \circ)$;

(b2) or, there exist κ', σ', δ'' and j such that
  i. $F \vdash (\kappa'', \sigma'') \overset{\tau}{\underset{\delta''}{\mapsto}}{}^{+} (\kappa', \sigma')$, and
  ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma''), (\delta_0 \cup \delta' \cup \delta'', F))$, and
  iii. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^{j}_{\mu} (F, (\kappa', \sigma'), \delta_0 \cup \delta' \cup \delta'', \circ)$.

For case (b1), since $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta', \Sigma), (\delta_0 \cup \delta', F))$, we know

$$\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta', F)) \ .$$

Thus we are done.

For case (b2), we know

$$F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta' \cup \delta''}{\mapsto}}{}^{+} (\kappa', \sigma') \ .$$

Thus we are done.

Thus we have finished the proof. □

Lemma 54. If $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \circ) \preccurlyeq^{i}_{\mu} (F, (\kappa, \sigma), \delta_0, \circ)$ and $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\tau}{\underset{\Delta}{\mapsto}}{}^{n} (\Bbbk', \Sigma')$ and $\mathbb{F} \vdash (\Bbbk', \Sigma') \overset{\iota}{\underset{\mathsf{emp}}{\mapsto}} (\Bbbk'', \Sigma')$ and $\iota \neq \tau$ and $\mathsf{HGuar}(\mu, \Delta_0 \cup \Delta, \mathbb{F})$, then there exist κ', δ, σ', κ', μ' and j such that

1. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\mapsto}}{}^{*} (\kappa', \sigma')$, and $F \vdash (\kappa', \sigma') \overset{\iota}{\underset{\mathsf{emp}}{\mapsto}} (\kappa'', \sigma')$, and
2. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta, F))$, and
3. $\mathsf{EvolveG}(\mu, \mu', \Sigma', \sigma', \mathbb{F}, F)$, and
4. $(\mathbb{F}, (\Bbbk'', \Sigma'), \mathsf{emp}, \bullet) \preccurlyeq^{j}_{\mu'} (F, (\kappa'', \sigma'), \mathsf{emp}, \bullet)$.

Proof. By induction over n.

1. $n = 0$. Trivial.
2. $n = m + 1$. We know there exists $\Bbbk'''$, Σ''', Δ' and Δ'' such that

$$\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\tau}{\underset{\Delta'}{\mapsto}} (\Bbbk''', \Sigma''') , \quad \mathbb{F} \vdash (\Bbbk''', \Sigma''') \overset{\tau}{\underset{\Delta''}{\mapsto}}{}^{m} (\Bbbk', \Sigma') , \quad \Delta = \Delta' \cup \Delta'' .$$

Since $\mathsf{HGuar}(\mu, \Delta_0 \cup \Delta, \mathbb{F})$, we know

$$\mathsf{HGuar}(\mu, \Delta_0 \cup \Delta', \mathbb{F}) .$$

Since $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \circ) \preccurlyeq^{i}_{\mu} (F, (\kappa, \sigma), \delta_0, \circ)$, we know one of the following holds:

(a) there exists i' such that
  i. $i' < i$, and
  ii. $(\mathbb{F}, (\Bbbk''', \Sigma'''), \Delta_0 \cup \Delta', \circ) \preccurlyeq^{i'}_{\mu} (F, (\kappa, \sigma), \delta_0, \circ)$;

(b) or, there exist κ''', σ''', δ' and i' such that
  i. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta'}{\mapsto}}{}^{+} (\kappa''', \sigma''')$, and

ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta', \Sigma), (\delta_0 \cup \delta', F))$, and
iii. $(\mathbb{F}, (\Bbbk''', \Sigma'''), \Delta_0 \cup \Delta', \circ) \preccurlyeq_{\mu}^{i'} (F, (\kappa''', \sigma'''), \delta_0 \cup \delta', \circ)$.

For case (a), by the induction hypothesis, we are done.

For case (b), by the induction hypothesis, we know there exist κ', δ, σ', κ', μ' and j such that

(1) $F \vdash (\kappa''', \sigma''') \overset{\tau}{\underset{\delta}{\mapsto}}{}^{*} (\kappa', \sigma')$, and $F \vdash (\kappa', \sigma') \overset{\iota}{\underset{\mathsf{emp}}{\mapsto}} (\kappa'', \sigma')$, and
(2) $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma'''), (\delta_0 \cup \delta' \cup \delta, F))$, and
(3) $\mathsf{EvolveG}(\mu, \mu', \Sigma', \sigma', \mathbb{F}, F)$, and
(4) $(\mathbb{F}, (\Bbbk'', \Sigma'), \mathsf{emp}, \bullet) \preccurlyeq_{\mu'}^{j} (F, (\kappa'', \sigma'), \mathsf{emp}, \bullet)$.

Since $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\tau}{\underset{\Delta'}{\mapsto}} (\Bbbk''', \Sigma''')$, we know $\mathsf{forward}(\Sigma, \Sigma''')$ and $(\mathsf{locs}(\Sigma) - \mathsf{locs}(\Sigma''')) \subseteq \Delta'.ws \cap \Delta'.rs$. Thus, from $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma'''), (\delta_0 \cup \delta' \cup \delta, F))$, by Lemma 57, then

$$\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta' \cup \delta, F)) .$$

Thus we are done.

Thus we have finished the proof. □

Lemma 55. If $\mathsf{FPmatch}(\mu_1, (\Delta_1, \Sigma_1), (\Delta_2, \mathbb{F}_2))$, $\mathsf{FPmatch}(\mu_2, (\Delta_2, \Sigma_2), (\Delta_3, \mathbb{F}_3))$, $\mathsf{wf}(\mu_1, (\Sigma_1, \mathbb{F}_1), (\Sigma_2, \mathbb{F}_2))$, $\mathsf{wf}(\mu_2, (\Sigma_2, \mathbb{F}_2), (\Sigma_3, \mathbb{F}_3))$, $\mu_1 = (\mathbb{S}_1, \mathbb{S}_2, f_1)$, $\mu_2 = (\mathbb{S}_2, \mathbb{S}_3, f_2)$ and $\mu = (\mathbb{S}_1, \mathbb{S}_3, f_2 \circ f_1)$, then $\mathsf{FPmatch}(\mu, (\Delta_1, \Sigma_1), (\Delta_3, \mathbb{F}_3))$.

Proof. From $\mathsf{FPmatch}(\mu_1, (\Delta_1, \Sigma_1), (\Delta_2, \mathbb{F}_2))$ and $\mathsf{FPmatch}(\mu_2, (\Delta_2, \Sigma_2), (\Delta_3, \mathbb{F}_3))$, we know

$$\begin{gathered}\Delta_2.rs \cup \Delta_2.ws \subseteq \lfloor\!\lfloor \mathbb{F}_2 \rfloor\!\rfloor \cup f_1\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle, \\ \Delta_2.rs \cap f_1\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle \subseteq f_1\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \Delta_1.rs \rangle\!\rangle, \quad \Delta_2.ws \cap f_1\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle \subseteq f_1\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \Delta_1.ws \rangle\!\rangle, \\ \Delta_3.rs \cup \Delta_3.ws \subseteq \lfloor\!\lfloor \mathbb{F}_3 \rfloor\!\rfloor \cup f_2\langle\!\langle \mathsf{locs}(\Sigma_2) \cap \lfloor\!\lfloor \mathbb{S}_2 \rfloor\!\rfloor \rangle\!\rangle, \\ \Delta_3.rs \cap f_2\langle\!\langle \mathsf{locs}(\Sigma_2) \cap \lfloor\!\lfloor \mathbb{S}_2 \rfloor\!\rfloor \rangle\!\rangle \subseteq f_2\langle\!\langle \mathsf{locs}(\Sigma_2) \cap \Delta_2.rs \rangle\!\rangle, \quad \Delta_3.ws \cap f_2\langle\!\langle \mathsf{locs}(\Sigma_2) \cap \lfloor\!\lfloor \mathbb{S}_2 \rfloor\!\rfloor \rangle\!\rangle \subseteq f_2\langle\!\langle \mathsf{locs}(\Sigma_2) \cap \Delta_2.ws \rangle\!\rangle \ .\end{gathered}$$

From $\mathsf{wf}(\mu_1, (\Sigma_1, \mathbb{F}_1), (\Sigma_2, \mathbb{F}_2))$ and $\mathsf{wf}(\mu_2, (\Sigma_2, \mathbb{F}_2), (\Sigma_3, \mathbb{F}_3))$, we know

$$\begin{array}{lllll}\mathsf{injective}(f_1, \Sigma_1)\ , & \mathbb{S}_1 \subseteq \mathsf{dom}(f_1)\ , & f_1\{\mathbb{S}_1\} \subseteq \mathbb{S}_2\ , & f_1\{\mathbb{F}_1 \cap \mathbb{S}_1\} \subseteq \mathbb{F}_2\ , & f_1\{\mathbb{S}_1 - \mathbb{F}_1\} \cap \mathbb{F}_2 = \emptyset\ , \\ \mathsf{injective}(f_2, \Sigma_2)\ , & \mathbb{S}_2 \subseteq \mathsf{dom}(f_2)\ , & f_2\{\mathbb{S}_2\} \subseteq \mathbb{S}_3\ , & f_2\{\mathbb{F}_2 \cap \mathbb{S}_2\} \subseteq \mathbb{F}_3\ , & f_2\{\mathbb{S}_2 - \mathbb{F}_2\} \cap \mathbb{F}_3 = \emptyset\ .\end{array}$$

Thus

$$\begin{aligned}&\Delta_3.rs \subseteq \lfloor\!\lfloor \mathbb{F}_3 \rfloor\!\rfloor \cup (\Delta_3.rs \cap f_2\langle\!\langle \mathsf{locs}(\Sigma_2) \cap \lfloor\!\lfloor \mathbb{S}_2 \rfloor\!\rfloor \rangle\!\rangle) \\ &\subseteq \lfloor\!\lfloor \mathbb{F}_3 \rfloor\!\rfloor \cup f_2\langle\!\langle \Delta_2.rs \rangle\!\rangle \\ &\subseteq \lfloor\!\lfloor \mathbb{F}_3 \rfloor\!\rfloor \cup f_2\langle\!\langle (\lfloor\!\lfloor \mathbb{F}_2 \rfloor\!\rfloor \cup f_1\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle) \rangle\!\rangle \\ &\subseteq \lfloor\!\lfloor \mathbb{F}_3 \rfloor\!\rfloor \cup \lfloor\!\lfloor \mathbb{F}_3 \rfloor\!\rfloor \cup (f_2 \circ f_1)\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle = \lfloor\!\lfloor \mathbb{F}_3 \rfloor\!\rfloor \cup (f_2 \circ f_1)\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle\end{aligned}$$

Similarly, we can also prove $\Delta_3.ws \subseteq \lfloor\!\lfloor \mathbb{F}_3 \rfloor\!\rfloor \cup (f_2 \circ f_1)\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle$.

Also, since $f_1\langle\!\langle \mathsf{locs}(\Sigma_1) \rangle\!\rangle \subseteq \mathsf{locs}(\Sigma_2)$ and $f_1\{\mathbb{S}_1\} \subseteq \mathbb{S}_2$, we know

$$f_1\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle \subseteq \mathsf{locs}(\Sigma_2) \cap \lfloor\!\lfloor \mathbb{S}_2 \rfloor\!\rfloor \ .$$

Since $\mathsf{injective}(f_2, \Sigma_2)$, we know

$$f_2\langle\!\langle \mathsf{locs}(\Sigma_2) \cap \Delta_2.rs \rangle\!\rangle \cap f_2\langle\!\langle (f_1\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle) \rangle\!\rangle \subseteq f_2\langle\!\langle (\Delta_2.rs \cap f_1\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle) \rangle\!\rangle \ .$$

Thus

$$\begin{aligned}&\Delta_3.rs \cap (f_2 \circ f_1)\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle \\ &\subseteq (\Delta_3.rs \cap f_2\langle\!\langle \mathsf{locs}(\Sigma_2) \cap \lfloor\!\lfloor \mathbb{S}_2 \rfloor\!\rfloor \rangle\!\rangle) \cap (f_2 \circ f_1)\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle \\ &\subseteq f_2\langle\!\langle \mathsf{locs}(\Sigma_2) \cap \Delta_2.rs \rangle\!\rangle \cap f_2\langle\!\langle (f_1\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle) \rangle\!\rangle \\ &\subseteq f_2\langle\!\langle (\Delta_2.rs \cap f_1\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle) \rangle\!\rangle \\ &\subseteq (f_2 \circ f_1)\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \Delta_1.rs \rangle\!\rangle\end{aligned}$$

Similarly, we can also prove $\Delta_3.ws \cap (f_2 \circ f_1)\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \lfloor\!\lfloor \mathbb{S}_1 \rfloor\!\rfloor \rangle\!\rangle \subseteq (f_2 \circ f_1)\langle\!\langle \mathsf{locs}(\Sigma_1) \cap \Delta_1.rs \rangle\!\rangle$. Thus we are done. □

Lemma 56. If $\mathsf{EvolveG}(\mu_1, \mu_1', \Sigma_1', \Sigma_2', \mathbb{F}_1, \mathbb{F}_2)$, $\mathsf{EvolveG}(\mu_2, \mu_2', \Sigma_2', \Sigma_3', \mathbb{F}_2, \mathbb{F}_3)$, $\mu_1 = (\mathbb{S}_1, \mathbb{S}_2, f_1)$, $\mu_2 = (\mathbb{S}_2, \mathbb{S}_3, f_2)$, $\mu = (\mathbb{S}_1, \mathbb{S}_3, f_2 \circ f_1)$, $\mu_1' = (\mathbb{S}_1', \mathbb{S}_2', f_1')$, $\mu_2' = (\mathbb{S}_2', \mathbb{S}_3', f_2')$, and $\mu' = (\mathbb{S}_1', \mathbb{S}_3', f_2' \circ f_1')$, then $\mathsf{EvolveG}(\mu, \mu', \Sigma_1', \Sigma_3', \mathbb{F}_1, \mathbb{F}_3)$.

Proof. From $\mathsf{EvolveG}(\mu_1, \mu_1', \Sigma_1', \Sigma_2', \mathbb{F}_1, \mathbb{F}_2)$ and $\mathsf{EvolveG}(\mu_2, \mu_2', \Sigma_2', \Sigma_3', \mathbb{F}_2, \mathbb{F}_3)$, we know

$$\begin{array}{c}\mathsf{wf}(\mu_1', \Sigma_1', \mathbb{F}_1, \mathbb{F}_2), \quad f_1 \subseteq f_1', \quad \mathsf{Inv}(f_1', \Sigma_1', \Sigma_2'), \quad \mathbb{S}_1' = \mathsf{cl}(\mathbb{S}_1, \Sigma_1') \subseteq \mathsf{dom}(\Sigma_1'), \quad \mathbb{S}_2' = \mathsf{cl}(\mathbb{S}_2, \Sigma_2') \subseteq \mathsf{dom}(\Sigma_2'),\\ \mathsf{wf}(\mu_2', \Sigma_2', \mathbb{F}_2, \mathbb{F}_3), \quad f_2 \subseteq f_2', \quad \mathsf{Inv}(f_2', \Sigma_2', \Sigma_3'), \quad \mathbb{S}_2' = \mathsf{cl}(\mathbb{S}_2, \Sigma_2') \subseteq \mathsf{dom}(\Sigma_2'), \quad \mathbb{S}_3' = \mathsf{cl}(\mathbb{S}_3, \Sigma_3') \subseteq \mathsf{dom}(\Sigma_3'),\\ \mathbb{S}_1' - \mathbb{S}_1 \subseteq \mathbb{F}_1, \quad \mathbb{S}_2' - \mathbb{S}_2 \subseteq \mathbb{F}_2 \,.\end{array}$$

Thus we know

$$\begin{array}{c}\mathsf{wf}(\mu', \Sigma_1', \mathbb{F}_1, \mathbb{F}_3), \quad (f_2 \circ f_1) \subseteq (f_2' \circ f_1'), \quad \mathsf{Inv}(f_2' \circ f_1', \Sigma_1', \Sigma_3'), \quad \mathbb{S}_1' = \mathsf{cl}(\mathbb{S}_1, \Sigma_1') \subseteq \mathsf{dom}(\Sigma_1'), \quad \mathbb{S}_3' = \mathsf{cl}(\mathbb{S}_3, \Sigma_3') \subseteq \mathsf{dom}(\Sigma_3'),\\ \mathbb{S}_1' - \mathbb{S}_1 \subseteq \mathbb{F}_1.\end{array}$$

Thus we are done. □

Lemma 57. If $\mathsf{FPmatch}(\mu, (\Delta, \Sigma'), (\delta, F))$, $\mathsf{dom}(\mu.f) = \mu.\mathbb{S} \subseteq \mathsf{dom}(\Sigma)$, $\mathsf{forward}(\Sigma, \Sigma')$ and $(\mathsf{locs}(\Sigma) - \mathsf{locs}(\Sigma')) \subseteq \Delta.ws \cap \Delta.rs$, then $\mathsf{FPmatch}(\mu, (\Delta, \Sigma), (\delta, F))$.

Proof. From $\mathsf{FPmatch}(\mu, (\Delta, \Sigma'), (\delta, F))$, we know

$$\begin{array}{c}\delta.rs \cup \delta.ws \ \subseteq\ \lfloor\!\lfloor F \rfloor\!\rfloor \cup \mu.f\langle\!\langle \mathsf{locs}(\Sigma') \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor \rangle\!\rangle\ ,\\ \delta.rs \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma') \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor \rangle\!\rangle \ \subseteq\ \mu.f\langle\!\langle \mathsf{locs}(\Sigma') \cap \Delta.rs \rangle\!\rangle\ ,\\ \delta.ws \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma') \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor \rangle\!\rangle \ \subseteq\ \mu.f\langle\!\langle \mathsf{locs}(\Sigma') \cap \Delta.ws \rangle\!\rangle\ .\end{array}$$

From $\mathsf{forward}(\Sigma, \Sigma')$, we know $\mathsf{locs}(\Sigma') \cap \lfloor\!\lfloor \mathsf{dom}(\Sigma) \rfloor\!\rfloor \subseteq \mathsf{locs}(\Sigma)$. Since $\mu.\mathbb{S} \subseteq \mathsf{dom}(\Sigma)$, we know

$$\mathsf{locs}(\Sigma') \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor \subseteq \mathsf{locs}(\Sigma) \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor\,.$$

Thus

$$\begin{array}{c}\mu.f\langle\!\langle \mathsf{locs}(\Sigma') \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor \rangle\!\rangle \subseteq \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor \rangle\!\rangle\ ,\\ \mu.f\langle\!\langle \mathsf{locs}(\Sigma') \cap \Delta.ws \rangle\!\rangle \subseteq \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta.ws \rangle\!\rangle\ , \qquad \mu.f\langle\!\langle \mathsf{locs}(\Sigma') \cap \Delta.rs \rangle\!\rangle \subseteq \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta.rs \rangle\!\rangle\,.\end{array}$$

Thus we have

$$\delta.rs \cup \delta.ws \ \subseteq\ \lfloor\!\lfloor F \rfloor\!\rfloor \cup \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor \rangle\!\rangle\,.$$

Since $(\mathsf{locs}(\Sigma) - \mathsf{locs}(\Sigma')) \subseteq \Delta.ws \cap \Delta.rs$, we know

$$\mathsf{locs}(\Sigma) \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor \subseteq (\mathsf{locs}(\Sigma') \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor) \cup (\Delta.ws \cap \Delta.rs \cap \mathsf{locs}(\Sigma))\,.$$

Thus

$$\begin{array}{c}\delta.rs \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor \rangle\!\rangle \ \subseteq\ \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta.rs \rangle\!\rangle\ ,\\ \delta.ws \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor\!\lfloor \mu.\mathbb{S} \rfloor\!\rfloor \rangle\!\rangle \ \subseteq\ \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta.ws \rangle\!\rangle\ .\end{array}$$

Thus we are done. □

E.3 Proof of Compositionality of Simulation (Lemma 48)

For any $\mathbb{T}$, $\mathtt{t}$, $\mathbb{d}$ and Σ, if $(\mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_n, \Sigma) \overset{load}{:\Longrightarrow} (\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma)$, we know there exist $\mathbb{F}_1, \ldots, \mathbb{F}_n, \mathbb{K}_1, \ldots, \mathbb{K}_n$ such that

$$\begin{array}{c}\mathsf{initFList}(\Sigma, (\mathbb{F}_1, \ldots, \mathbb{F}_n))\ , \qquad \forall i \in \{1, \ldots, n\}.\, \mathbb{K}_i = \mathsf{initRtMd}(\Gamma, \mathtt{f}_i, \mathbb{F}_i)\ ,\\ \mathbb{T} = \{\mathtt{t}_1 \rightsquigarrow \mathbb{K}_1, \ldots, \mathtt{t}_n \rightsquigarrow \mathbb{K}_n\}\ , \qquad \mathtt{t} \in \{\mathtt{t}_1, \ldots, \mathtt{t}_n\}\ , \qquad \mathbb{d} = \{\mathtt{t}_1 \rightsquigarrow 0, \ldots, \mathtt{t}_n \rightsquigarrow 0\}\,.\end{array}$$

From $\mathsf{initFList}(\Sigma, (\mathbb{F}_1, \ldots, \mathbb{F}_n))$, we know

$$\forall i \in \{1, \ldots, n\}.\, (\mathsf{dom}(\Sigma) \cap \mathbb{F}_i = \emptyset) \wedge (\forall j \neq i.\, \mathbb{F}_i \cap \mathbb{F}_j = \emptyset)\,.$$

From $\forall i \in \{1, \ldots, n\}.\, \mathbb{K}_i = \mathsf{initRtMd}(\Gamma, \mathtt{f}_i, \mathbb{F}_i)$, we know there exist $a_1, \ldots, a_n, \Bbbk_1, \ldots, \Bbbk_n$ such that

$$\forall i \in \{1, \ldots, n\}.\, a_i \in \{1, \ldots, m\} \wedge \Bbbk_i = sl_{a_i}.\mathsf{InitCore}(\gamma_{a_i}, \mathtt{f}_i) \wedge \mathbb{K}_i = (sl_{a_i}, (\gamma_{a_i}, \mathbb{F}_i), \Bbbk_i)\,.$$

Similarly, for any T and σ, if $(\mathbf{let}\ \Pi\ \mathbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_n, \sigma) \overset{load}{:\Longrightarrow} (T, \mathtt{t}, \mathbb{d}, \sigma)$, we know there exist $F_1, \ldots, F_n, K_1, \ldots, K_n$ such that

$$\mathsf{initFList}(\sigma, (F_1, \ldots, F_n))\ , \qquad \forall i \in \{1, \ldots, n\}.\, K_i = \mathsf{initRtMd}(\Pi, \mathtt{f}_i, F_i)\ , \qquad T = \{\mathtt{t}_1 \rightsquigarrow K_1, \ldots, \mathtt{t}_n \rightsquigarrow K_n\}\,.$$

From $\mathsf{initFList}(\sigma, (F_1, \ldots, F_n))$, we know

$$\forall i \in \{1, \ldots, n\}.\, (\mathsf{dom}(\sigma) \cap F_i = \emptyset) \wedge (\forall j \neq i.\, F_i \cap F_j = \emptyset)\,.$$

From $\forall i \in \{1, \ldots, n\}.\, K_i = \mathsf{initRtMd}(\Pi, \mathtt{f}_i, F_i)$, $\forall i \in \{1, \ldots, m\}.\, \mathsf{dom}(sl_i.\mathsf{InitCore}(\gamma_i)) = \mathsf{dom}(tl_i.\mathsf{InitCore}(\pi_i))$ and the assumption that entries of different modules are different, we know there exist $\kappa_1, \ldots, \kappa_n$ such that

$$\forall i \in \{1, \ldots, n\}.\, \kappa_i = tl_{a_i}.\mathsf{InitCore}(\pi_{a_i}, \mathtt{f}_i) \wedge K_i = (tl_{a_i}, (\pi_{a_i}, F_i), \kappa_i)\,.$$

For any μ, $\mathbb{S}$ and S, if $ge \subseteq \mathsf{locs}(\Sigma)$, $\mathsf{dom}(\Sigma) = \mathbb{S} \subseteq \mathsf{dom}(\varphi)$, $\mathsf{cl}(\mathbb{S}, \Sigma) = \mathbb{S}$, $\mathsf{locs}(\sigma) = \varphi\langle\!\langle \mathsf{locs}(\Sigma) \rangle\!\rangle$, $\mathsf{Inv}(\varphi, \Sigma, \sigma)$ and $\mu = (\mathbb{S}, \varphi\{\mathbb{S}\}, \varphi|_{\mathbb{S}})$, since $ge = \bigcup_i ge_i$, we know

$$\forall i \in \{1, \ldots, m\}.\, ge_i \subseteq \mathsf{locs}(\Sigma) \ .$$

From

$$\forall a_i \in \{1, \ldots, m\}.\, (sl_{a_i}, ge_{a_i}, \gamma_{a_i}) \preccurlyeq_\varphi (tl_{a_i}, ge'_{a_i}, \pi_{a_i}) \ ,$$

we know

$$\forall a_i \in \{1, \ldots, m\}.\, \exists j_i \in \mathtt{index}.\, (\mathbb{F}_i, (\mathbb{k}_i, \Sigma), \mathsf{emp}, \bullet) \preccurlyeq_\mu^{j_i} (F_i, (\kappa_i, \sigma), \mathsf{emp}, \bullet) \ .$$

Since $\mathtt{t} \in \{\mathtt{t}_1, \ldots, \mathtt{t}_n\}$, from

$$(\mathbb{F}_\mathtt{t}, (\mathbb{k}_\mathtt{t}, \Sigma), \mathsf{emp}, \bullet) \preccurlyeq_\mu^{j_\mathtt{t}} (F_\mathtt{t}, (\kappa_\mathtt{t}, \sigma), \mathsf{emp}, \bullet) \ ,$$

we know there exists j such that

$$(\mathbb{F}_\mathtt{t}, (\mathbb{k}_\mathtt{t}, \Sigma), \mathsf{emp}, \circ) \preccurlyeq_\mu^{j} (F_\mathtt{t}, (\kappa_\mathtt{t}, \sigma), \mathsf{emp}, \circ) \ .$$

Then, by the following Lemma 58, we know

$$((\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma), \mathsf{emp}) \preccurlyeq_\mu^{j} ((T, \mathtt{t}, \mathbb{d}, \sigma), \mathsf{emp}) \ .$$

Lemma 58. If

1. $\mathbb{T} = \{\mathtt{t}_1 \rightsquigarrow (sl_1, \mathbb{F}_1, \mathbb{k}_1), \ldots, \mathtt{t}_n \rightsquigarrow (sl_n, \mathbb{F}_n, \mathbb{k}_n)\}$, $T = \{\mathtt{t}_1 \rightsquigarrow (tl_1, F_1, \kappa_1), \ldots, \mathtt{t}_n \rightsquigarrow (tl_n, F_n, \kappa_n)\}$,
2. $\mathtt{t} \in \{\mathtt{t}_1, \ldots, \mathtt{t}_n\}$, $(\mathbb{F}_\mathtt{t}, (\mathbb{k}_\mathtt{t}, \Sigma), \Delta_0, \circ) \preccurlyeq_\mu^{i} (F_\mathtt{t}, (\kappa_\mathtt{t}, \sigma), \delta_0, \circ)$, and $\mathsf{RC}(\mathbb{F}_\mathtt{t}, \mu.\mathbb{S}, (\mathbb{k}_\mathtt{t}, \Sigma), \circ)$,
3. for any $\mathtt{t}' \neq \mathtt{t}$, there exists $i_{\mathtt{t}'} \in \mathtt{index}$ such that $(\mathbb{F}_{\mathtt{t}'}, (\mathbb{k}_{\mathtt{t}'}, \Sigma_0), \mathsf{emp}, \bullet) \preccurlyeq_\mu^{i_{\mathtt{t}'}} (F_{\mathtt{t}'}, (\kappa_{\mathtt{t}'}, \sigma_0), \mathsf{emp}, \bullet)$, and $\mathsf{RC}(\mathbb{F}_{\mathtt{t}'}, \mu.\mathbb{S}, (\mathbb{k}_{\mathtt{t}'}, \Sigma_0), \bullet)$,
4. $\forall \mathtt{t}' \neq \mathtt{t}.\, (\mathbb{F}_{\mathtt{t}'} \cap \mathbb{F}_\mathtt{t} = \emptyset) \wedge (F_{\mathtt{t}'} \cap F_\mathtt{t} = \emptyset)$, and for any $\mathtt{t}$, we have $\mathsf{wd}(sl_\mathtt{t})$ and $\mathsf{wd}(tl_\mathtt{t})$,
5. $\Sigma_0 \overset{\mathcal{U}-\Delta_0.ws}{=\!=\!=} \Sigma$, $\sigma_0 \overset{\mathcal{U}-\delta_0.ws}{=\!=\!=} \sigma$, $\mathsf{forward}(\Sigma_0, \Sigma)$, $\mathsf{forward}(\sigma_0, \sigma)$,

then $((\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma), \Delta_0) \preccurlyeq_\mu^{i} ((T, \mathtt{t}, \mathbb{d}, \sigma), \delta_0)$.

Proof. By co-induction. Let $\widehat{\mathbb{W}} = (\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma)$ and $\widehat{W} = (T, \mathtt{t}, \mathbb{d}, \sigma)$. We need to prove the following:

1. if $(\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma) \underset{\Delta}{\overset{\tau}{:\Rightarrow}} \widehat{\mathbb{W}}'$, then one of the following holds:
   a. there exists j such that
      i. $j < i$, and
      ii. $(\widehat{\mathbb{W}}', \Delta_0 \cup \Delta) \preccurlyeq_\mu^{j} (\widehat{W}, \delta_0)$;
   b. or, there exist $\widehat{W}'$, δ and j such that
      i. $(T, \mathtt{t}, \mathbb{d}, \sigma) \underset{\delta}{\overset{\tau}{:\Rightarrow}}{}^{+} \widehat{W}'$, and
      ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta, F_\mathtt{t}))$, and
      iii. $(\widehat{\mathbb{W}}', \Delta_0 \cup \Delta) \preccurlyeq_\mu^{j} (\widehat{W}', \delta_0 \cup \delta)$;

   Proof. From $(\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma) \underset{\Delta}{\overset{\tau}{:\Rightarrow}} \widehat{\mathbb{W}}'$, we know there exist $\mathbb{k}'_\mathtt{t}$ and Σ' such that

   $$\widehat{\mathbb{W}}' = (\mathbb{T}\{\mathtt{t} \rightsquigarrow (sl_\mathtt{t}, \mathbb{F}_\mathtt{t}, \mathbb{k}'_\mathtt{t})\}, \mathtt{t}, \mathbb{d}, \Sigma') \ , \qquad \mathbb{F}_\mathtt{t} \vdash (\mathbb{k}_\mathtt{t}, \Sigma) \underset{\Delta}{\overset{\tau}{\longmapsto}} (\mathbb{k}'_\mathtt{t}, \Sigma')$$

   From $(\mathbb{F}_\mathtt{t}, (\mathbb{k}_\mathtt{t}, \Sigma), \Delta_0, \circ) \preccurlyeq_\mu^{i} (F_\mathtt{t}, (\kappa_\mathtt{t}, \sigma), \delta_0, \circ)$ and $\mathsf{RC}(\mathbb{F}_\mathtt{t}, \mu.\mathbb{S}, (\mathbb{k}_\mathtt{t}, \Sigma), \circ)$, we know one of the following holds:

   (A) there exists j such that
      (1) $j < i$, and
      (2) $(\mathbb{F}_\mathtt{t}, (\mathbb{k}'_\mathtt{t}, \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq_\mu^{j} (F_\mathtt{t}, (\kappa_\mathtt{t}, \sigma), \delta_0, \circ)$;

   (B) or, there exist $\kappa'_\mathtt{t}$, σ', δ and j such that
      (3) $F_\mathtt{t} \vdash (\kappa_\mathtt{t}, \sigma) \underset{\delta}{\overset{\tau}{\longmapsto}}{}^{+} (\kappa'_\mathtt{t}, \sigma')$, and
      (4) $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta, F_\mathtt{t}))$, and
      (5) $(\mathbb{F}_\mathtt{t}, (\mathbb{k}'_\mathtt{t}, \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq_\mu^{j} (F_\mathtt{t}, (\kappa'_\mathtt{t}, \sigma'), \delta_0 \cup \delta, \circ)$;

   For case (A):
   From $\mathbb{F}_\mathtt{t} \vdash (\mathbb{k}_\mathtt{t}, \Sigma) \underset{\Delta}{\overset{\tau}{\longmapsto}} (\mathbb{k}'_\mathtt{t}, \Sigma')$, by $\mathsf{wd}(sl_\mathtt{t})$, we know

   $$\Sigma \overset{\mathcal{U}-\Delta.ws}{=\!=\!=} \Sigma' \ , \quad \mathsf{forward}(\Sigma, \Sigma') \ .$$

From $\Sigma_0 \xlongequal{\mathfrak{U}-\Delta_0.ws} \Sigma$ and $\mathsf{forward}(\Sigma_0, \Sigma)$, we know

$$\Sigma_0 \xlongequal{\mathfrak{U}-(\Delta_0.ws \cup \Delta.ws)} \Sigma' , \quad \mathsf{forward}(\Sigma_0, \Sigma') .$$

From (2), by the co-induction hypothesis, we know

$$(\widehat{\mathbb{W}}', \Delta_0 \cup \Delta) \preccurlyeq^j_\mu (\widehat{W}, \delta_0) .$$

For case (B):

From (3), we know there exists $\widehat{W}'$ such that

$$\widehat{W}' = (T\{\mathsf{t} \rightsquigarrow (tl_\mathsf{t}, F_\mathsf{t}, \kappa'_\mathsf{t})\}, \mathsf{t}, \mathbb{d}, \sigma') , \qquad (T, \mathsf{t}, \mathbb{d}, \sigma) \underset{\delta}{\overset{\tau}{:\Longrightarrow}}{}^+ \widehat{W}'$$

From $\mathbb{F}_\mathsf{t} \vdash (\mathbb{k}_\mathsf{t}, \Sigma) \underset{\Delta}{\overset{\tau}{\longmapsto}} (\mathbb{k}'_\mathsf{t}, \Sigma')$, by $\mathsf{wd}(sl_\mathsf{t})$, we know

$$\Sigma \xlongequal{\mathfrak{U}-\Delta.ws} \Sigma' , \quad \mathsf{forward}(\Sigma, \Sigma') .$$

From $\Sigma_0 \xlongequal{\mathfrak{U}-\Delta_0.ws} \Sigma$ and $\mathsf{forward}(\Sigma_0, \Sigma)$, we know

$$\Sigma_0 \xlongequal{\mathfrak{U}-(\Delta_0.ws \cup \Delta.ws)} \Sigma' , \quad \mathsf{forward}(\Sigma_0, \Sigma') .$$

Similarly, from $\sigma_0 \xlongequal{\mathfrak{U}-\delta_0.ws} \sigma$, $\mathsf{forward}(\sigma_0, \sigma)$, and (3), by $\mathsf{wd}(tl_\mathsf{t})$, we know

$$\sigma_0 \xlongequal{\mathfrak{U}-(\delta_0.ws \cup \delta.ws)} \sigma' , \quad \mathsf{forward}(\sigma_0, \sigma') .$$

From (5), by the co-induction hypothesis, we know

$$(\widehat{\mathbb{W}}', \Delta_0 \cup \Delta) \preccurlyeq^j_\mu (\widehat{W}', \delta_0 \cup \delta) .$$

Thus we are done for this clause. □

2. $\forall \mathbb{T}', \mathbb{d}', o.$ if $o \neq \tau$ and $\exists \mathsf{t}'. (\mathbb{T}, \mathsf{t}, \mathbb{d}, \Sigma) \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (\mathbb{T}', \mathsf{t}', \mathbb{d}', \Sigma)$, then there exist $\widehat{W}'$, δ, μ', T', σ' and j such that for any $\mathsf{t}' \in \mathsf{dom}(\mathbb{T}')$,
   a. $(T, \mathsf{t}, \mathbb{d}, \sigma) \underset{\delta}{\overset{\tau}{:\Longrightarrow}}{}^* \widehat{W}'$, and $\widehat{W}' \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (T', \mathsf{t}', \mathbb{d}', \sigma')$, and
   b. $\mathsf{FPmatch}(\mu, (\Delta_0, \Sigma), (\delta_0 \cup \delta, T(\mathsf{t}).F))$, and
   c. $((\mathbb{T}', \mathsf{t}', \mathbb{d}', \Sigma), \mathsf{emp}) \preccurlyeq^j_{\mu'} ((T', \mathsf{t}', \mathbb{d}', \sigma'), \mathsf{emp})$;

Proof. Since $\exists \mathsf{t}'. (\mathbb{T}, \mathsf{t}, \mathbb{d}, \Sigma) \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (\mathbb{T}', \mathsf{t}', \mathbb{d}', \Sigma)$, we know one of the following two cases hold:

(A) there exist ι and $\mathbb{k}'_\mathsf{t}$ such that

$$\iota \neq \mathtt{ret} , \qquad \iota \neq \tau , \qquad \mathbb{F}_\mathsf{t} \vdash (\mathbb{k}_\mathsf{t}, \Sigma) \underset{\mathsf{emp}}{\overset{\iota}{\longmapsto}} (\mathbb{k}'_\mathsf{t}, \Sigma) , \qquad \mathbb{T}' = \mathbb{T}\{\mathsf{t} \rightsquigarrow (sl_\mathsf{t}, \mathbb{F}_\mathsf{t}, \mathbb{k}'_\mathsf{t})\}.$$

From $(\mathbb{F}_\mathsf{t}, (\mathbb{k}_\mathsf{t}, \Sigma), \Delta_0, \circ) \preccurlyeq^i_\mu (F_\mathsf{t}, (\kappa_\mathsf{t}, \sigma), \delta_0, \circ)$ and $\mathsf{RC}(\mathbb{F}_\mathsf{t}, \mu.\mathbb{S}, (\mathbb{k}_\mathsf{t}, \Sigma), \circ)$, we know there exist κ'_t, σ', δ, κ''_t, μ' and j_t such that

(1) $F_\mathsf{t} \vdash (\kappa_\mathsf{t}, \sigma) \underset{\delta}{\overset{\tau}{\longmapsto}}{}^* (\kappa'_\mathsf{t}, \sigma')$, and $F_\mathsf{t} \vdash (\kappa'_\mathsf{t}, \sigma') \underset{\mathsf{emp}}{\overset{\iota}{\longmapsto}} (\kappa''_\mathsf{t}, \sigma')$, and
(2) $\mathsf{blocks}(\Delta_0.rs \cup \Delta_0.ws) \subseteq \mathbb{F}_\mathsf{t} \cup \mu.\mathbb{S}$, and
(3) $\mathsf{FPmatch}(\mu, (\Delta_0, \Sigma), (\delta_0 \cup \delta, F_\mathsf{t}))$, and
(4) $\mathsf{EvolveG}(\mu, \mu', \Sigma, \sigma', \mathbb{F}_\mathsf{t}, F_\mathsf{t})$, and
(5) $(\mathbb{F}_\mathsf{t}, (\mathbb{k}'_\mathsf{t}, \Sigma), \mathsf{emp}, \bullet) \preccurlyeq^{j_\mathsf{t}}_{\mu'} (F_\mathsf{t}, (\kappa''_\mathsf{t}, \sigma'), \mathsf{emp}, \bullet)$.

Let $\widehat{W}' = (T\{\mathsf{t} \rightsquigarrow (tl_\mathsf{t}, F_\mathsf{t}, \kappa'_\mathsf{t})\}, \mathsf{t}, \mathbb{d}, \sigma')$. From (1), we know there exists T' such that for any t',

$$(T, \mathsf{t}, \mathbb{d}, \sigma) \underset{\delta}{\overset{\tau}{:\Longrightarrow}}{}^* \widehat{W}' , \qquad \widehat{W}' \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (T', \mathsf{t}', \mathbb{d}', \sigma') , \qquad T' = T\{\mathsf{t} \rightsquigarrow (tl_\mathsf{t}, F_\mathsf{t}, \kappa''_\mathsf{t})\}.$$

From (5), we know there exists j'_t such that

$$(\mathbb{F}_\mathsf{t}, (\mathbb{k}'_\mathsf{t}, \Sigma), \mathsf{emp}, \circ) \preccurlyeq^{j'_\mathsf{t}}_{\mu'} (F_\mathsf{t}, (\kappa''_\mathsf{t}, \sigma'), \mathsf{emp}, \circ) .$$

For any $\mathsf{t}' \neq \mathsf{t}$, from $\mathbb{F}_{\mathsf{t}'} \cap \mathbb{F}_\mathsf{t} = \emptyset$ and $\mathsf{blocks}(\Delta_0.ws) \subseteq \mathbb{F}_\mathsf{t} \cup \mu.\mathbb{S}$, we know

$$\lfloor\!\lfloor \mathbb{F}_{\mathsf{t}'} - \mu.\mathbb{S} \rfloor\!\rfloor \subseteq \mathfrak{U} - \Delta_0.ws .$$

From $\Sigma_0 \xLongequal{\mathfrak{U}-\Delta_0.ws} \Sigma$, we know

$$\Sigma_0 \xLongequal{\lfloor\mathbb{F}_{t'}-\mu.\mathbb{S}\rfloor} \Sigma\ .$$

For the target states $\sigma_{t'}$ and σ, from $\mathsf{FPmatch}(\mu, (\Delta_0, \Sigma), (\delta_0 \cup \delta, F_t))$, we know

$$\delta_0.ws \cup \delta.ws \subseteq \lfloor F_t \rfloor \cup \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor\mu.\mathbb{S}\rfloor \rangle\!\rangle\ .$$

Since $\mathsf{forward}(\Sigma_0, \Sigma)$ and $\mu.\mathbb{S} \subseteq \mathsf{dom}(\Sigma_0)$, we know

$$\mathsf{locs}(\Sigma) \cap \lfloor\mu.\mathbb{S}\rfloor \subseteq \mathsf{locs}(\Sigma_0) \cap \lfloor\mu.\mathbb{S}\rfloor\ .$$

Then, from $F_{t'} \cap F_t = \emptyset$, we know

$$\lfloor F_{t'} \rfloor - \mu.f\langle\!\langle \mathsf{locs}(\Sigma_0) \cap \lfloor\mu.\mathbb{S}\rfloor \rangle\!\rangle \ \subseteq\ \mathfrak{U} - (\delta_0.ws \cup \delta.ws)\ .$$

From $\sigma_0 \xLongequal{\mathfrak{U}-\delta_0.ws} \sigma$, $\mathsf{forward}(\sigma_0, \sigma)$, and (1), by $\mathsf{wd}(tl_t)$, we know

$$\sigma_0 \xLongequal{\mathfrak{U}-(\delta_0.ws \cup \delta.ws)} \sigma'\ , \quad \mathsf{forward}(\sigma_0, \sigma')\ .$$

Thus

$$\sigma_0 \xLongequal{\lfloor F_{t'} \rfloor - \mu.f\langle \mathsf{locs}(\Sigma_0) \cap \lfloor\mu.\mathbb{S}\rfloor \rangle} \sigma'\ .$$

Also, since $\mathsf{EvolveG}(\mu, \mu', \Sigma, \sigma', \mathbb{F}_t, F_t)$ and $\mathbb{F}_{t'} \cap \mathbb{F}_t = \emptyset$ and $F_{t'} \cap F_t = \emptyset$, we know

$$\mathsf{EvolveR}(\mu, \mu', \Sigma, \sigma', \mathbb{F}_{t'}, F_{t'})\ .$$

From $(\mathbb{F}_{t'}, (\Bbbk_{t'}, \Sigma_0), \mathsf{emp}, \bullet) \preccurlyeq_{\mu}^{i_{t'}} (F_{t'}, (\kappa_{t'}, \sigma_0), \mathsf{emp}, \bullet)$ and $\mathsf{RC}(\mathbb{F}_{t'}, \mu.\mathbb{S}, (\Bbbk_{t'}, \Sigma_0), \bullet)$, we know there exists $j_{t'}$ such that

$$(\mathbb{F}_{t'}, (\Bbbk_{t'}, \Sigma), \mathsf{emp}, \circ) \preccurlyeq_{\mu'}^{j_{t'}} (F_{t'}, (\kappa_{t'}, \sigma'), \mathsf{emp}, \circ)\ .$$

Also, by Lemma 59, we know

$$(\mathbb{F}_{t'}, (\Bbbk_{t'}, \Sigma), \mathsf{emp}, \bullet) \preccurlyeq_{\mu'}^{i_{t'}} (F_{t'}, (\kappa_{t'}, \sigma'), \mathsf{emp}, \bullet)\ .$$

Thus, for any $\mathtt{t}'$ (no matter whether $\mathtt{t}' = \mathtt{t}$ or $\mathtt{t}' \neq \mathtt{t}$), by the co-induction hypothesis, we know there exists j such that

$$((\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma), \mathsf{emp}) \preccurlyeq_{\mu'}^{j} ((T', \mathtt{t}', \mathbb{d}', \sigma'), \mathsf{emp})\ .$$

(B) there exists $\Bbbk_t'$ such that

$$\mathbb{F}_t \vdash (\Bbbk_t, \Sigma) \overset{\mathsf{ret}}{\underset{\mathsf{emp}}{\longmapsto}} (\Bbbk_t', \Sigma)\ , \qquad \mathbb{T}' = \mathbb{T}\backslash\mathtt{t}\ , \qquad \mathbb{d}' = \mathbb{d}\backslash\mathtt{t}\ .$$

From $(\mathbb{F}_t, (\Bbbk_t, \Sigma), \Delta_0, \circ) \preccurlyeq_{\mu}^{i} (F_t, (\kappa_t, \sigma), \delta_0, \circ)$ and $\mathsf{RC}(\mathbb{F}_t, \mu.\mathbb{S}, (\Bbbk_t, \Sigma), \circ)$, we know there exist κ_t', σ', δ, κ_t'', μ' and j_t such that

(1) $F_t \vdash (\kappa_t, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{*} (\kappa_t', \sigma')$, and $F_t \vdash (\kappa_t', \sigma') \overset{\mathsf{ret}}{\underset{\mathsf{emp}}{\longmapsto}} (\kappa_t'', \sigma')$, and
(2) $\mathsf{blocks}(\Delta_0.rs \cup \Delta_0.ws) \subseteq \mathbb{F}_t \cup \mu.\mathbb{S}$, and
(3) $\mathsf{FPmatch}(\mu, (\Delta_0, \Sigma), (\delta_0 \cup \delta, F_t))$, and
(4) $\mathsf{EvolveG}(\mu, \mu', \Sigma, \sigma', \mathbb{F}_t, F_t)$.

Let $\widehat{W}' = (T\{\mathtt{t} \leadsto (tl_t, F_t, \kappa_t')\}, \mathtt{t}, \mathbb{d}, \sigma')$. From (1), we know there exists T' such that for any $\mathtt{t}' \in \mathsf{dom}(T\backslash\mathtt{t})$, we have

$$(T, \mathtt{t}, \mathbb{d}, \sigma) \overset{\tau}{\underset{\delta}{:\Longrightarrow}}{}^{*} \widehat{W}'\ , \qquad \widehat{W}' \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} (T', \mathtt{t}', \mathbb{d}', \sigma')\ , \qquad T' = T\backslash\mathtt{t}\ .$$

Similarly as (A), we know there exists $j_{t'}$ such that

$$(\mathbb{F}_{t'}, (\Bbbk_{t'}, \Sigma), \mathsf{emp}, \circ) \preccurlyeq_{\mu'}^{j_{t'}} (F_{t'}, (\kappa_{t'}, \sigma'), \mathsf{emp}, \circ)\ .$$

Thus, by the co-induction hypothesis, we know

$$((\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma), \mathsf{emp} \preccurlyeq_{\mu'}^{j_{t'}} ((T', \mathtt{t}', \mathbb{d}', \sigma'), \mathsf{emp})\ .$$

Thus we are done for this clause. □

3. if $(\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma) \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **done**, then there exist $\widehat{W}'$ and δ such that
   a. $(T, \mathtt{t}, \mathbb{d}, \sigma) \overset{\tau}{\underset{\delta}{:\Longrightarrow}}{}^{*} \widehat{W}'$, and $\widehat{W}' \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **done**, and
   b. $\mathsf{FPmatch}(\mu, (\Delta_0, \Sigma), (\delta_0 \cup \delta, F_t))$.

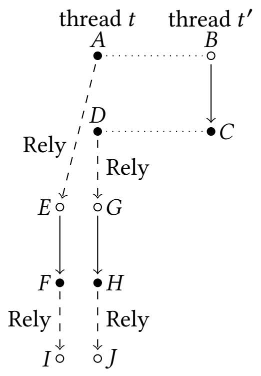


Figure 24. The execution in the auxiliary lemmas for the compositionality proofs.

Proof. From $(\mathbb{T}, \mathsf{t}, \mathbb{d}, \Sigma) :\overset{\tau}{\underset{\mathsf{emp}}{\Longrightarrow}}$ **done**, we know

$$\mathbb{F}_{\mathsf{t}} \vdash (\mathbb{k}_{\mathsf{t}}, \Sigma) \overset{\mathsf{ret}}{\underset{\mathsf{emp}}{\longmapsto}} (\mathbb{k}'_{\mathsf{t}}, \Sigma)\ , \qquad \mathsf{dom}(\mathbb{T}) = \{\mathsf{t}\}\ .$$

From $(\mathbb{F}_{\mathsf{t}}, (\mathbb{k}_{\mathsf{t}}, \Sigma), \Delta_0, \circ) \preccurlyeq^i_{\mu} (F_{\mathsf{t}}, (\kappa_{\mathsf{t}}, \sigma), \delta_0, \circ)$ and $\mathsf{RC}(\mathbb{F}_{\mathsf{t}}, \mu.\mathbb{S}, (\mathbb{k}_{\mathsf{t}}, \Sigma), \circ)$, we know there exist κ'_{t}, δ, σ', κ''_{t} such that

$$F_{\mathsf{t}} \vdash (\kappa_{\mathsf{t}}, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^{*} (\kappa'_{\mathsf{t}}, \sigma')\ , \qquad F_{\mathsf{t}} \vdash (\kappa'_{\mathsf{t}}, \sigma') \overset{\mathsf{ret}}{\underset{\mathsf{emp}}{\longmapsto}} (\kappa''_{\mathsf{t}}, \sigma')\ , \qquad \mathsf{FPmatch}(\mu, (\Delta_0, \Sigma), (\delta_0 \cup \delta, F_{\mathsf{t}}))\ .$$

Let $\widehat{W}' = (T\{\mathsf{t} \rightsquigarrow (tl_{\mathsf{t}}, F_{\mathsf{t}}, \kappa'_{\mathsf{t}})\}, \mathsf{t}, \mathbb{d}, \sigma')$. Since $\mathsf{dom}(T) = \{\mathsf{t}\}$, we know

$$(T, \mathsf{t}, \mathbb{d}, \sigma) :\overset{\tau}{\underset{\delta}{\Rightarrow}}{}^{*} \widehat{W}'\ , \qquad \widehat{W}' :\overset{\tau}{\underset{\mathsf{emp}}{\Longrightarrow}} \textbf{done}\ .$$

Thus we are done. □

Thus we have finished the proof. □

Lemma 59 (From A to D in Fig. 24)**.** If

1. $(\mathbb{F}, (\mathbb{k}, \Sigma_A), \mathsf{emp}, \bullet) \preccurlyeq^i_{\mu_A} (F, (\kappa, \sigma_A), \mathsf{emp}, \bullet)$,
2. $\Sigma_A \xlongequal{\lfloor\!\lfloor \mathbb{F}-\mu_A.\mathbb{S} \rfloor\!\rfloor} \Sigma_D$, $\sigma_A \xlongequal{\lfloor\!\lfloor F \rfloor\!\rfloor - \mu_A.f\langle \mathsf{locs}(\Sigma) \cap \lfloor\!\lfloor \mu_A.\mathbb{S} \rfloor\!\rfloor \rangle} \sigma_D$, $\mathsf{forward}(\Sigma_A, \Sigma_D)$, $\mathsf{forward}(\sigma_A, \sigma_D)$, and $\mathsf{EvolveR}(\mu_A, \mu_D, \Sigma_D, \sigma_D, \mathbb{F}, F)$,

then $(\mathbb{F}, (\mathbb{k}, \Sigma_D), \mathsf{emp}, \bullet) \preccurlyeq^i_{\mu_D} (F, (\kappa, \sigma_D), \mathsf{emp}, \bullet)$.

Proof. By co-induction. From $(\mathbb{F}, (\mathbb{k}, \Sigma_A), \mathsf{emp}, \bullet) \preccurlyeq^i_{\mu_A} (F, (\kappa, \sigma_A), \mathsf{emp}, \bullet)$, we know

$$\mathsf{wf}(\mu_A, (\Sigma_A, \mathbb{F}), (\sigma_A, F))\ .$$

From $\mathsf{EvolveR}(\mu_A, \mu_D, \Sigma_D, \sigma_D, \mathbb{F}, F)$, we know

$$\begin{gathered}\mathsf{wf}(\mu_D, (\Sigma_D, \mathbb{F}), (\sigma_D, F)), \quad \mu_A.f \subseteq \mu_D.f, \quad \mathsf{Inv}(\mu_D.f, \Sigma_D, \sigma_D), \\ \mu_D.\mathbb{S} = \mathsf{cl}(\mu_A.\mathbb{S}, \Sigma_D), \quad \mu_D.S = \mathsf{cl}(\mu_A.S, \sigma_D), \quad (\mu_D.\mathbb{S} - \mu_A.\mathbb{S}) \cap \mathbb{F} = \emptyset\ .\end{gathered}$$

For any $\sigma_G, \Sigma_G, \mu_G$, suppose

$$\Sigma_D \xlongequal{\lfloor\!\lfloor \mathbb{F}-\mu_D.\mathbb{S} \rfloor\!\rfloor} \Sigma_G,\ \sigma_D \xlongequal{\lfloor\!\lfloor F \rfloor\!\rfloor - \mu_D.f\langle \mathsf{locs}(\Sigma_D) \cap \lfloor\!\lfloor \mu_D.\mathbb{S} \rfloor\!\rfloor \rangle} \sigma_G,\ \mathsf{forward}(\Sigma_D, \Sigma_G),\ \mathsf{forward}(\sigma_D, \sigma_G),\ \mathsf{EvolveR}(\mu_D, \mu_G, \Sigma_G, \sigma_G, \mathbb{F}, F).$$

From $(\mu_D.\mathbb{S} - \mu_A.\mathbb{S}) \cap \mathbb{F} = \emptyset$ and $\Sigma_D \xlongequal{\lfloor\!\lfloor \mathbb{F}-\mu_D.\mathbb{S} \rfloor\!\rfloor} \Sigma_G$, we know

$$\Sigma_D \xlongequal{\lfloor\!\lfloor \mathbb{F}-\mu_A.\mathbb{S} \rfloor\!\rfloor} \Sigma_G\ .$$

From $\Sigma_A \xlongequal{\lfloor\!\lfloor \mathbb{F}-\mu_A.\mathbb{S} \rfloor\!\rfloor} \Sigma_D$, we know

$$\Sigma_A \xlongequal{\lfloor\!\lfloor \mathbb{F}-\mu_A.\mathbb{S} \rfloor\!\rfloor} \Sigma_G\ .$$

From $(\mu_D.\mathbb{S} - \mu_A.\mathbb{S}) \cap \mathbb{F} = \emptyset$, we know

$$\mu_D.f\{\mu_D.\mathbb{S} - \mu_A.\mathbb{S}\} \subseteq \mu_D.f\{\mu_D.\mathbb{S} - \mathbb{F}\}\ .$$

Since $\mu_D.f\{\mu_D.\mathbb{S} - \mathbb{F}\} \cap F = \emptyset$, we know

$$\mu_D.f\{\mu_D.\mathbb{S} - \mu_A.\mathbb{S}\} \cap F = \emptyset\ .$$

Thus

$$\mu_D.f\langle\!\langle(\mathsf{locs}(\Sigma_D) \cap \llbracket\mu_D.\mathbb{S}\rrbracket) - (\mathsf{locs}(\Sigma_A) \cap \llbracket\mu_A.\mathbb{S}\rrbracket)\rangle\!\rangle \cap \llbracket F\rrbracket = \emptyset\ .$$

From $\mu_A.f \subseteq \mu_D.f$ and $\mu_A.\mathbb{S} \subseteq \mathsf{dom}(\mu.f)$, we know

$$(\mu_D.f\langle\!\langle\mathsf{locs}(\Sigma_D) \cap \llbracket\mu_D.\mathbb{S}\rrbracket\rangle\!\rangle - \mu_A.f\langle\!\langle\mathsf{locs}(\Sigma_A) \cap \llbracket\mu_A.\mathbb{S}\rrbracket\rangle\!\rangle) \cap \llbracket F\rrbracket = \emptyset\ .$$

From $\sigma_D \xlongequal{\llbracket F\rrbracket - \mu_D.f\langle\mathsf{locs}(\Sigma_D)\cap\llbracket\mu_D.\mathbb{S}\rrbracket\rangle} \sigma_G$, we know

$$\sigma_D \xlongequal{\llbracket F\rrbracket - \mu_A.f\langle\mathsf{locs}(\Sigma_A)\cap\llbracket\mu_A.\mathbb{S}\rrbracket\rangle} \sigma_G\ .$$

From $\sigma_A \xlongequal{\llbracket F\rrbracket - \mu_A.f\langle\mathsf{locs}(\Sigma_A)\cap\llbracket\mu_A.\mathbb{S}\rrbracket\rangle} \sigma_D$, we know

$$\sigma_A \xlongequal{\llbracket F\rrbracket - \mu_A.f\langle\mathsf{locs}(\Sigma_A)\cap\llbracket\mu_A.\mathbb{S}\rrbracket\rangle} \sigma_G\ .$$

From $\mathsf{forward}(\Sigma_A, \Sigma_D)$, $\mathsf{forward}(\sigma_A, \sigma_D)$, $\mathsf{forward}(\Sigma_D, \Sigma_G)$ and $\mathsf{forward}(\sigma_D, \sigma_G)$, we know

$$\mathsf{forward}(\Sigma_A, \Sigma_G), \qquad \mathsf{forward}(\sigma_A, \sigma_G)\ .$$

Let $\mu_E = (\mathbb{S}_E, S_E, f_E)$, where $\mathbb{S}_E = \mathsf{cl}(\mu_A.\mathbb{S}, \Sigma_G)$, $S_E = \mathsf{cl}(\mu_A.S, \sigma_G)$ and $f_E = (\mu_G.f)|_{\mathbb{S}_E}$. Since $\mu_D.\mathbb{S} = \mathsf{cl}(\mu_A.\mathbb{S}, \Sigma_D)$, we know $\mu_A.\mathbb{S} \subseteq \mu_D.\mathbb{S}$. Since $\mu_G.\mathbb{S} = \mathsf{cl}(\mu_D.\mathbb{S}, \Sigma_G)$, we know $\mathbb{S}_E \subseteq \mu_G.\mathbb{S}$. Similarly we know $S_E \subseteq \mu_G.S$. Since $\mathsf{dom}(\mu_G.f) = \mu_G.\mathbb{S}$, we know $\mathsf{dom}(f_E) = \mathbb{S}_E$. Since $\mu_A.f \subseteq \mu_D.f \subseteq \mu_G.f$ and $\mu_A.\mathbb{S} \subseteq \mathbb{S}_E$, we know

$$\mu_A.f \subseteq f_E.$$

Since $\mathsf{Inv}(\mu_G.f, \Sigma_G, \sigma_G)$, $\mathsf{dom}(\mu_A.f) = \mu_A.\mathbb{S}$ and $(\mu_A.f)\{\mu_A.\mathbb{S}\} \subseteq \mu_A.S$, by Lemma 61, we know $(\mu_G.f)\{\mathbb{S}_E\} \subseteq S_E$. Thus

$$f_E\{\mathbb{S}_E\} \subseteq S_E.$$

Since $\mathsf{wf}(\mu_G, (\Sigma_G, \mathbb{F}), (\sigma_G, F))$, we know

$$\mathsf{wf}(\mu_E, (\Sigma_G, \mathbb{F}), (\sigma_G, F))\ .$$

Since $\mathsf{Inv}(\mu_G.f, \Sigma_G, \sigma_G)$ and $\mathbb{S}_E = \mathsf{cl}(\mathbb{S}_E, \Sigma_G)$, by Lemma 62, we know

$$\mathsf{Inv}(f_E, \Sigma_G, \sigma_G)\ .$$

Since $\mathbb{S}_E \subseteq \mu_G.\mathbb{S}$, $(\mu_G.\mathbb{S} - \mu_D.\mathbb{S}) \cap \mathbb{F} = \emptyset$ and $(\mu_D.\mathbb{S} - \mu_A.\mathbb{S}) \cap \mathbb{F} = \emptyset$, we know

$$(\mathbb{S}_E - \mu_A.\mathbb{S}) \cap \mathbb{F} = \emptyset\ .$$

Thus we know

$$\mathsf{EvolveR}(\mu_A, \mu_E, \Sigma_G, \sigma_G, \mathbb{F}, F).$$

From $(\mathbb{F}, (\Bbbk, \Sigma_A), \mathsf{emp}, \bullet) \preccurlyeq^i_{\mu_A} (F, (\kappa, \sigma_A), \mathsf{emp}, \bullet)$, we know there exists j such that

$$(\mathbb{F}, (\Bbbk, \Sigma_G), \mathsf{emp}, \circ) \preccurlyeq^j_{\mu_E} (F, (\kappa, \sigma_G), \mathsf{emp}, \circ).$$

Since $(\mu_G.\mathbb{S} - \mu_D.\mathbb{S}) \cap \mathbb{F} = \emptyset$, $(\mu_D.\mathbb{S} - \mu_A.\mathbb{S}) \cap \mathbb{F} = \emptyset$ and $(\mathbb{S}_E - \mu_A.\mathbb{S}) \cap \mathbb{F} = \emptyset$, we know

$$(\mu_G.\mathbb{S} - \mathbb{S}_E) \cap \mathbb{F} = \emptyset\ .$$

By Lemma 60, we know

$$(\mathbb{F}, (\Bbbk, \Sigma_G), \mathsf{emp}, \circ) \preccurlyeq^j_{\mu_G} (F, (\kappa, \sigma_G), \mathsf{emp}, \circ).$$

Thus we are done. □

Lemma 60 (From E to G in Fig. 24). If

1. $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \circ) \preccurlyeq^i_{\mu_E} (F, (\kappa, \sigma), \delta_0, \circ)$,
2. $\Sigma_G \xlongequal{\mathfrak{U}-\Delta_0.ws} \Sigma$, $\sigma_G \xlongequal{\mathfrak{U}-\delta_0.ws} \sigma$, $\mathsf{forward}(\Sigma_G, \Sigma)$, $\mathsf{forward}(\sigma_G, \sigma)$, $\mu_E = (\mathbb{S}_E, S_E, f_E)$, $\mu_G = (\mathbb{S}_G, S_G, f_G)$, $\mathsf{wf}(\mu_E, (\Sigma_G, \mathbb{F}), (\sigma_G, F))$, $\mathsf{wf}(\mu_G, (\Sigma_G, \mathbb{F}), (\sigma_G, F))$, $\mathbb{S}_E = \mathsf{cl}(\mathbb{S}_E, \Sigma_G)$, $\mathbb{S}_G = \mathsf{cl}(\mathbb{S}_G, \Sigma_G)$, $S_E = \mathsf{cl}(S_E, \sigma_G)$, $S_G = \mathsf{cl}(S_G, \sigma_G)$, $\mathbb{S}_E \subseteq \mathbb{S}_G$, $S_E \subseteq S_G$, $f_E = f_G|_{\mathbb{S}_E}$, $(\mathbb{S}_G - \mathbb{S}_E) \cap \mathbb{F} = \emptyset$,

then $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \circ) \preccurlyeq^i_{\mu_G} (F, (\kappa, \sigma), \delta_0, \circ)$.

Proof. By co-induction. Since $\mathsf{forward}(\Sigma_G, \Sigma)$ and $\mathsf{wf}(\mu_G, (\Sigma_G, \mathbb{F}), (\sigma, F))$, we know

$$\mathsf{wf}(\mu_G, (\Sigma, \mathbb{F}), (\sigma, F)).$$

Also we need to prove the following.

1. $\forall \Bbbk', \Sigma', \Delta$. if $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\tau}{\underset{\Delta}{\longmapsto}} (\Bbbk', \Sigma')$, then one of the following holds:
   a. there exists j such that
      i. $j < i$, and
      ii. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^j_{\mu_G} (F, (\kappa, \sigma), \delta_0, \circ)$;
   b. or, there exist κ', σ', δ and j such that
      i. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^+ (\kappa', \sigma')$, and
      ii. $\mathsf{FPmatch}(\mu_G, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta, F))$, and
      iii. $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^j_{\mu_G} (F, (\kappa', \sigma'), \delta_0 \cup \delta, \circ)$.

   Proof. From $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \circ) \preccurlyeq^i_{\mu_E} (F, (\kappa, \sigma), \delta_0, \circ)$, we know one of the following holds:

   (A) there exists j such that
   (1) $j < i$, and
   (2) $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^j_{\mu_E} (F, (\kappa, \sigma), \delta_0, \circ)$;
   (B) or, there exist κ', σ', δ and j such that
   (3) $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\longmapsto}}{}^+ (\kappa', \sigma')$, and
   (4) $\mathsf{FPmatch}(\mu_E, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta, F))$, and
   (5) $(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^j_{\mu_E} (F, (\kappa', \sigma'), \delta_0 \cup \delta, \circ)$.

   For case (A):
   From $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\tau}{\underset{\Delta}{\longmapsto}} (\Bbbk', \Sigma')$, by $\mathsf{wd}(sl)$, we know

   $$\Sigma \xlongequal{\mathfrak{U}-\Delta.ws} \Sigma', \quad \mathsf{forward}(\Sigma, \Sigma') .$$

   Since $\Sigma_G \xlongequal{\mathfrak{U}-\Delta_0.ws} \Sigma$ and $\mathsf{forward}(\Sigma_G, \Sigma)$, we know

   $$\Sigma_G \xlongequal{\mathfrak{U}-(\Delta_0 \cup \Delta).ws} \Sigma', \qquad \mathsf{forward}(\Sigma_G, \Sigma') .$$

   From (2), by the co-induction hypothesis, we know

   $$(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^j_{\mu_G} (F, (\kappa, \sigma), \delta_0, \circ) .$$

   For case (B):
   Since $(\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\tau}{\underset{\Delta}{\longmapsto}} (\Bbbk', \Sigma')$, $\Sigma_G \xlongequal{\mathfrak{U}-\Delta_0.ws} \Sigma$ and $\mathsf{forward}(\Sigma_G, \Sigma)$, we know

   $$\Sigma_G \xlongequal{\mathfrak{U}-(\Delta_0 \cup \Delta).ws} \Sigma', \qquad \mathsf{forward}(\Sigma_G, \Sigma') .$$

   From (3), $\sigma_G \xlongequal{\mathfrak{U}-\delta_0.ws} \sigma$ and $\mathsf{forward}(\sigma_G, \sigma)$, by $\mathsf{wd}(tl)$, we know

   $$\sigma_G \xlongequal{\mathfrak{U}-(\delta_0 \cup \delta).ws} \sigma', \qquad \mathsf{forward}(\sigma_G, \sigma') .$$

   From (5), by the co-induction hypothesis, we know

   $$(\mathbb{F}, (\Bbbk', \Sigma'), \Delta_0 \cup \Delta, \circ) \preccurlyeq^j_{\mu_G} (F, (\kappa', \sigma'), \delta_0 \cup \delta, \circ) .$$

   From (4), by Lemma 63, we know

   $$\mathsf{FPmatch}(\mu_G, (\Delta_0 \cup \Delta, \Sigma), (\delta_0 \cup \delta, F)) .$$

   Thus we have done for this clause. □

2. $\forall \Bbbk', \iota$. if $\iota \neq \tau$ and $\mathbb{F} \vdash (\Bbbk, \Sigma) \overset{\iota}{\underset{\text{emp}}{\mapsto}} (\Bbbk', \Sigma)$,
   then there exist $\kappa', \delta, \sigma', \kappa', \mu_H$ and j such that
   a. $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\mapsto}}{}^* (\kappa', \sigma')$, and
      $F \vdash (\kappa', \sigma') \overset{\iota}{\underset{\text{emp}}{\mapsto}} (\kappa'', \sigma')$, and
   b. $\text{blocks}(\Delta_0.rs \cup \Delta_0.ws) \subseteq \mathbb{F} \cup \mathbb{S}_G$ and
   c. $\text{FPmatch}(\mu_G, (\Delta_0, \Sigma), (\delta_0 \cup \delta, F))$, and
   d. $\text{EvolveG}(\mu_G, \mu_H, \Sigma, \sigma', \mathbb{F}, F)$, and
   e. $(\mathbb{F}, (\Bbbk', \Sigma), \text{emp}, \bullet) \preccurlyeq^{j}_{\mu_H} (F, (\kappa'', \sigma'), \text{emp}, \bullet)$.

   Proof. From $(\mathbb{F}, (\Bbbk, \Sigma), \Delta_0, \circ) \preccurlyeq^{i}_{\mu_E} (F, (\kappa, \sigma), \delta_0, \circ)$, we know there exist $\kappa', \delta, \sigma', \kappa', \mu_F$ and j such that

(1) $F \vdash (\kappa, \sigma) \overset{\tau}{\underset{\delta}{\mapsto}}{}^* (\kappa', \sigma')$, and
    $F \vdash (\kappa', \sigma') \overset{\iota}{\underset{\text{emp}}{\mapsto}} (\kappa'', \sigma')$, and
(2) $\text{blocks}(\Delta_0.rs \cup \Delta_0.ws) \subseteq \mathbb{F} \cup \mathbb{S}_E$ and
(3) $\text{FPmatch}(\mu_E, (\Delta_0, \Sigma), (\delta_0 \cup \delta, F))$, and
(4) $\text{EvolveG}(\mu_E, \mu_F, \Sigma, \sigma', \mathbb{F}, F)$, and
(5) $(\mathbb{F}, (\Bbbk', \Sigma), \text{emp}, \bullet) \preccurlyeq^{j}_{\mu_F} (F, (\kappa'', \sigma'), \text{emp}, \bullet)$.

From (2), since $\mathbb{S}_E \subseteq \mathbb{S}_G$, we know
$$\text{blocks}(\Delta_0.rs \cup \Delta_0.ws) \subseteq \mathbb{F} \cup \mathbb{S}_G .$$
From (3), by Lemma 63, we know
$$\text{FPmatch}(\mu_G, (\Delta_0, \Sigma), (\delta_0 \cup \delta, F)) .$$
Suppose $\mu_F = (\mathbb{S}_F, S_F, f_F)$. From (4), we know
$$\text{wf}(\mu_F, (\Sigma, \mathbb{F}), (\sigma', F)), \quad f_E \subseteq f_F, \quad \text{Inv}(f_F, \Sigma, \sigma'),$$
$$\mathbb{S}_F = \text{cl}(\mathbb{S}_E, \Sigma), \quad S_F = \text{cl}(S_E, \sigma'), \quad (\mathbb{S}_F - \mathbb{S}_E) \subseteq \mathbb{F} .$$
Let $\mu_H = (\mathbb{S}_H, S_H, f_H)$, where $\mathbb{S}_H = \text{cl}(\mathbb{S}_G, \Sigma)$, $S_H = \text{cl}(S_G, \sigma')$ and $f_H = f_G \cup (f_F|_{\mathbb{S}_H - \mathbb{S}_G})$. Thus we know $\mathbb{S}_G \subseteq \mathbb{S}_H$, $S_G \subseteq S_H$ and $f_G \subseteq f_H$. Since $\mathbb{S}_E \subseteq \mathbb{S}_G$ and $S_E \subseteq S_G$, we know $\mathbb{S}_F \subseteq \mathbb{S}_H$ and $S_F \subseteq S_H$. By Lemma 64, we know
$$\mathbb{S}_H - \mathbb{S}_G \subseteq \mathbb{S}_F .$$
Since $\mathbb{S}_E \subseteq \mathbb{S}_G$ and $(\mathbb{S}_F - \mathbb{S}_E) \subseteq \mathbb{F}$, we know
$$\mathbb{S}_H - \mathbb{S}_G \subseteq \mathbb{S}_F - \mathbb{S}_E \subseteq \mathbb{F} .$$
Since $\mathbb{S}_F \subseteq \mathbb{S}_H$ and $(\mathbb{S}_G - \mathbb{S}_E) \cap \mathbb{F} = \emptyset$, we know
$$\mathbb{S}_F - \mathbb{S}_E \subseteq (\mathbb{S}_H - \mathbb{S}_E) \cap \mathbb{F} = ((\mathbb{S}_H - \mathbb{S}_G) \cup (\mathbb{S}_G - \mathbb{S}_E)) \cap \mathbb{F} = \mathbb{S}_H - \mathbb{S}_G .$$
Also we know $\text{dom}(f_H) = \mathbb{S}_H$. Since $f_F\{\mathbb{S}_F\} \subseteq S_F$, we know
$$f_F\{\mathbb{S}_H - \mathbb{S}_G\} \subseteq S_F = \text{cl}(S_E, \sigma') \subseteq \text{cl}(S_G, \sigma') = S_H .$$
Since $f_G\{\mathbb{S}_G\} \subseteq S_G$, we know
$$f_H\{\mathbb{S}_H\} \subseteq S_H .$$
Also we know
$$f_H = f_G \cup (f_F|_{\mathbb{S}_H - \mathbb{S}_G}) = (f_G|_{\mathbb{S}_G - \mathbb{S}_E}) \cup f_E \cup (f_F|_{\mathbb{S}_H - \mathbb{S}_G}) = (f_G|_{\mathbb{S}_G - \mathbb{S}_E}) \cup f_F .$$
Since $\text{injective}(f_G, \Sigma_G)$, $\text{dom}(f_G) = \mathbb{S}_G \subseteq \text{dom}(\Sigma_G)$ and $\text{forward}(\Sigma_G, \Sigma)$, we know
$$\text{injective}(f_G, \Sigma) .$$
Since $(\mathbb{S}_G - \mathbb{S}_E) \cap \mathbb{F} = \emptyset$, $f_G\{\mathbb{S}_G - \mathbb{F}\} \cap F = \emptyset$, $(\mathbb{S}_H - \mathbb{S}_G) \subseteq \mathbb{F}$ and $f_F\{\mathbb{S}_F \cap \mathbb{F}\} \subseteq F$, we know
$$f_G\{\mathbb{S}_G - \mathbb{S}_E\} \cap F = \emptyset , \qquad f_F\{\mathbb{S}_H - \mathbb{S}_G\} \subseteq F .$$
Since $f_E \subseteq f_F$ and $\text{injective}(f_F, \Sigma)$, we know
$$\text{injective}(f_H, \Sigma) .$$
Since $\mathbb{S}_F \subseteq \text{dom}(\Sigma)$, $\mathbb{S}_G \subseteq \text{dom}(\Sigma_G)$ and $\text{forward}(\Sigma_G, \Sigma)$, we know
$$\mathbb{S}_H \subseteq \text{dom}(\Sigma) .$$
From (1), by $\text{wd}(tl)$, we know
$$\sigma \xlongequal{\mathfrak{U} - \delta.ws} \sigma', \qquad \text{forward}(\sigma, \sigma') .$$

Since $\sigma_G \overset{\mathfrak{U}-\delta_0.ws}{=\!=\!=\!=} \sigma$ and $\mathsf{forward}(\sigma_G, \sigma)$, we know

$$\sigma_G \overset{\mathfrak{U}-(\delta_0\cup\delta).ws}{=\!=\!=\!=\!=\!=} \sigma', \qquad \mathsf{forward}(\sigma_G, \sigma') .$$

From (3), we know

$$(\delta_0 \cup \delta).ws \subseteq \llbracket F \rrbracket \cup f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S}_E \rrbracket \rangle\!\rangle .$$

Since $f_G\{\mathbb{S}_G - \mathbb{S}_E\} \cap F = \emptyset$ and $\mathsf{injective}(f_G, \Sigma)$, we know

$$f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S}_G - \mathbb{S}_E \rrbracket \rangle\!\rangle \ \cap\ (\llbracket F \rrbracket \cup f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S}_E \rrbracket \rangle\!\rangle) \ =\ \emptyset .$$

Thus

$$f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S}_G - \mathbb{S}_E \rrbracket \rangle\!\rangle \subseteq \mathfrak{U} - (\delta_0 \cup \delta).ws .$$

As a result, since $f_H = (f_G|_{\mathbb{S}_G - \mathbb{S}_E}) \cup f_F$, $\mathsf{forward}(\Sigma_G, \Sigma)$, $f_G\langle\!\langle \mathsf{locs}(\Sigma_G) \rangle\!\rangle \subseteq \mathsf{locs}(\sigma_G)$ and $f_F\langle\!\langle \mathsf{locs}(\Sigma) \rangle\!\rangle \subseteq \mathsf{locs}(\sigma')$, we know

$$\begin{aligned}
& f_H\langle\!\langle \mathsf{locs}(\Sigma) \rangle\!\rangle \\
&= f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S}_G - \mathbb{S}_E \rrbracket \rangle\!\rangle \ \cup\ f_F\langle\!\langle \mathsf{locs}(\Sigma) \rangle\!\rangle \\
&\subseteq (f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S}_G - \mathbb{S}_E \rrbracket \rangle\!\rangle \cap f_G\langle\!\langle \mathsf{locs}(\Sigma_G) \cap \llbracket \mathbb{S}_G - \mathbb{S}_E \rrbracket \rangle\!\rangle) \ \cup\ \mathsf{locs}(\sigma') \\
&\subseteq ((\mathfrak{U} - (\delta_0 \cup \delta).ws) \cap \mathsf{locs}(\sigma_G)) \ \cup\ \mathsf{locs}(\sigma') \\
&\subseteq \mathsf{locs}(\sigma')
\end{aligned}$$

Thus we have proved $\mathsf{wf}(\mu_H, \Sigma, \sigma')$. Since $f_H = f_G \cup (f_F|_{\mathbb{S}_H - \mathbb{S}_G})$, $f_G\{\mathbb{S}_G \cap \mathbb{F}\} \subseteq F$ and $f_F\{\mathbb{S}_H - \mathbb{S}_G\} \subseteq F$, we know

$$\begin{aligned}
& f_H\{\mathbb{S}_H \cap \mathbb{F}\} \\
&\subseteq f_G\{\mathbb{S}_G \cap \mathbb{F}\} \cup f_F\{(\mathbb{S}_H - \mathbb{S}_G) \cap \mathbb{F}\} \\
&\subseteq F
\end{aligned}$$

Also, since $f_G\{\mathbb{S}_G - \mathbb{F}\} \cap F = \emptyset$ and $(\mathbb{S}_H - \mathbb{S}_G) \subseteq \mathbb{F}$, we know

$$\begin{aligned}
& f_H\{\mathbb{S}_H - \mathbb{F}\} \cap F \\
&\subseteq (f_G\{\mathbb{S}_G - \mathbb{F}\} \cup f_F\{(\mathbb{S}_H - \mathbb{S}_G) - \mathbb{F}\}) \cap F \\
&= \emptyset
\end{aligned}$$

Thus we have proved $\mathsf{wf}(\mu_H, (\Sigma, \mathbb{F}), (\sigma', F))$. Since $\mathsf{Inv}(f_G, \Sigma_G, \sigma_G)$ and $\mathsf{forward}(\Sigma_G, \Sigma)$, we know: for any b, n, b' and n', if $(b, n) \in \mathsf{locs}(\Sigma)$ and $(f_G|_{\mathbb{S}_G - \mathbb{S}_E})\langle (b, n) \rangle = (b', n')$, we know

$$(b', n') \in \mathsf{locs}(\sigma_G) \ , \qquad \Sigma_G(b)(n) \overset{f_G}{\hookrightarrow} \sigma_G(b')(n') .$$

Since $\mathsf{blocks}(\Delta_0.ws) \subseteq \mathbb{F} \cup \mathbb{S}_E$ and $(\mathbb{S}_G - \mathbb{S}_E) \cap \mathbb{F} = \emptyset$, we know

$$\llbracket \mathbb{S}_G - \mathbb{S}_E \rrbracket \subseteq \mathfrak{U} - \Delta_0.ws .$$

Since $\Sigma_G \overset{\mathfrak{U}-\Delta_0.ws}{=\!=\!=\!=} \Sigma$ and $b \in \mathbb{S}_G - \mathbb{S}_E$, we know

$$\Sigma_G(b)(n) = \Sigma(b)(n) .$$

Since $f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S}_G - \mathbb{S}_E \rrbracket \rangle\!\rangle \subseteq \mathfrak{U} - (\delta_0 \cup \delta).ws$ and $\sigma_G \overset{\mathfrak{U}-(\delta_0\cup\delta).ws}{=\!=\!=\!=\!=\!=} \sigma'$, we know

$$(b', n') \in \mathsf{locs}(\sigma') \ , \qquad \sigma_G(b')(n') = \sigma'(b')(n') .$$

Thus we know

$$\Sigma(b)(n) \overset{f_G}{\hookrightarrow} \sigma'(b')(n') .$$

Since $f_H = f_G \cup (f_F|_{\mathbb{S}_H - \mathbb{S}_G}) = (f_G|_{\mathbb{S}_G - \mathbb{S}_E}) \cup f_F$ and $\mathsf{Inv}(f_F, \Sigma, \sigma')$, we know

$$\mathsf{Inv}(f_H, \Sigma, \sigma') .$$

Thus we have proved $\mathsf{EvolveG}(\mu_G, \mu_H, \Sigma, \sigma', \mathbb{F}, F)$. Also, since $\mathbb{S}_F - \mathbb{S}_E = \mathbb{S}_H - \mathbb{S}_G$ and $(\mathbb{S}_G - \mathbb{S}_E) \cap \mathbb{F} = \emptyset$, we know

$$(\mathbb{S}_H - \mathbb{S}_F) \cap \mathbb{F} = \emptyset .$$

Below we consider the case when $(\mathbb{F}, (\Bbbk', \Sigma), \mathsf{emp}, \bullet) \preccurlyeq^{j}_{\mu_F} (F, (\kappa'', \sigma'), \mathsf{emp}, \bullet)$. We prove

$$(\mathbb{F}, (\Bbbk', \Sigma), \mathsf{emp}, \bullet) \preccurlyeq^{j}_{\mu_H} (F, (\kappa'', \sigma'), \mathsf{emp}, \bullet) \tag{E.2}$$

Proof: by co-induction. For any $\sigma_J, \Sigma_J, \mu_J$, suppose $\mu_J = (\mathbb{S}_J, S_J, f_J)$ and

$$\Sigma \overset{\llbracket \mathbb{F} - \mathbb{S}_H \rrbracket}{=\!=\!=\!=} \Sigma_J,\ \sigma' \overset{\llbracket F \rrbracket - f_H\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S}_H \rrbracket \rangle}{=\!=\!=\!=\!=\!=\!=\!=} \sigma_J,\ \mathsf{forward}(\Sigma, \Sigma_J),\ \mathsf{forward}(\sigma', \sigma_J),\ \mathsf{EvolveR}(\mu_H, \mu_J, \Sigma_J, \sigma_J, \mathbb{F}, F).$$

Since $(\mathbb{S}_H - \mathbb{S}_F) \cap \mathbb{F} = \emptyset$ and $\Sigma \overset{\llbracket \mathbb{F} - \mathbb{S}_H \rrbracket}{=\!=\!=\!=} \Sigma_J$, we know

$$\Sigma \overset{\llbracket \mathbb{F} - \mathbb{S}_F \rrbracket}{=\!=\!=\!=} \Sigma_J.$$

Since $f_H\{\mathbb{S}_H - \mathbb{F}\} \cap F = \emptyset$, we know

$$f_H\{\mathbb{S}_H - \mathbb{S}_F\} \cap F = \emptyset\ .$$

Since $f_F \subseteq f_H$ and $\mathbb{S}_F \subseteq \mathsf{dom}(f_F)$, we know

$$(f_H\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S}_H \rrbracket \rangle\!\rangle - f_F\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mathbb{S}_F \rrbracket \rangle\!\rangle) \cap \llbracket F \rrbracket = \emptyset\ .$$

Since $\sigma' \xlongequal{\llbracket F \rrbracket - f_H\langle\!\langle \mathsf{locs}(\Sigma)\cap\llbracket \mathbb{S}_H \rrbracket \rangle\!\rangle} \sigma_J$, we know

$$\sigma' \xlongequal{\llbracket F \rrbracket - f_F\langle\!\langle \mathsf{locs}(\Sigma)\cap\llbracket \mathbb{S}_F \rrbracket \rangle\!\rangle} \sigma_J\ .$$

Let $\mu_I = (\mathbb{S}_I, S_I, f_I)$, where $\mathbb{S}_I = \mathsf{cl}(\mathbb{S}_F, \Sigma_J)$, $S_I = \mathsf{cl}(S_F, \sigma_J)$ and $f_I = f_J|_{\mathbb{S}_I}$. Since $\mathbb{S}_F \subseteq \mathbb{S}_H$, $S_F \subseteq S_H$, $\mathbb{S}_J = \mathsf{cl}(\mathbb{S}_H, \Sigma_J)$ and $S_J = \mathsf{cl}(S_H, \sigma_J)$, we know

$$\mathbb{S}_I \subseteq \mathbb{S}_J, \qquad S_I \subseteq S_J.$$

Since $\mathsf{dom}(f_J) = \mathbb{S}_J$, we know

$$\mathsf{dom}(f_I) = \mathbb{S}_I.$$

Since $f_F \subseteq f_H$, $f_H \subseteq f_J$ and $\mathbb{S}_F \subseteq \mathbb{S}_I$, we know

$$f_F \subseteq f_I \subseteq f_J.$$

Since $\mathsf{Inv}(f_J, \Sigma_J, \sigma_J)$, $\mathsf{dom}(f_F) = \mathbb{S}_F$ and $f_F\{\mathbb{S}_F\} \subseteq S_F$, by Lemma 61, we know $f_J\{\mathbb{S}_I\} \subseteq S_I$. Thus

$$f_I\{\mathbb{S}_I\} \subseteq S_I.$$

Since $\mathsf{wf}(\mu_J, (\Sigma_J, \mathbb{F}), (\sigma_J, F))$, we know

$$\mathsf{wf}(\mu_I, (\Sigma_J, \mathbb{F}), (\sigma_J, F))\ .$$

Since $\mathsf{Inv}(f_J, \Sigma_J, \sigma_J)$ and $\mathbb{S}_I = \mathsf{cl}(\mathbb{S}_I, \Sigma_J)$, by Lemma 62, we know

$$\mathsf{Inv}(f_I, \Sigma_J, \sigma_J)\ .$$

Since $\mathbb{S}_I \subseteq \mathbb{S}_J$, $(\mathbb{S}_J - \mathbb{S}_H) \cap \mathbb{F} = \emptyset$ and $(\mathbb{S}_H - \mathbb{S}_F) \cap \mathbb{F} = \emptyset$, we know

$$(\mathbb{S}_I - \mathbb{S}_F) \cap \mathbb{F} = \emptyset\ .$$

Thus we know

$$\mathsf{EvolveR}(\mu_F, \mu_I, \Sigma_J, \sigma_J, \mathbb{F}, F).$$

From $(\mathbb{F}, (\Bbbk', \Sigma), \mathsf{emp}, \bullet) \preccurlyeq^{j}_{\mu_F} (F, (\kappa'', \sigma'), \mathsf{emp}, \bullet)$, we know there exists j' such that

$$(\mathbb{F}, (\Bbbk', \Sigma_J), \mathsf{emp}, \circ) \preccurlyeq^{j'}_{\mu_I} (F, (\kappa'', \sigma_J), \mathsf{emp}, \circ)\ .$$

Since $(\mathbb{S}_I - \mathbb{S}_F) \cap \mathbb{F} = \emptyset$, $(\mathbb{S}_J - \mathbb{S}_H) \cap \mathbb{F} = \emptyset$ and $(\mathbb{S}_H - \mathbb{S}_F) \cap \mathbb{F} = \emptyset$, we know

$$(\mathbb{S}_J - \mathbb{S}_I) \cap \mathbb{F} = \emptyset\ .$$

Thus by the co-induction hypothesis, we know

$$(\mathbb{F}, (\Bbbk', \Sigma_J), \mathsf{emp}, \circ) \preccurlyeq^{j'}_{\mu_J} (F, (\kappa'', \sigma_J), \mathsf{emp}, \circ)\ .$$

Thus we have proved (E.2). Thus we have done for this clause. □

Thus we are done. □

Lemma 61. If $f_1 \subseteq f_2$, $\mathsf{dom}(f_1) = \mathbb{S}_1$, $f_1\{\mathbb{S}_1\} \subseteq S_1$, $\mathbb{S}_2 = \mathsf{cl}(\mathbb{S}_1, \Sigma) \subseteq \mathsf{dom}(f_2)$, $S_2 = \mathsf{cl}(S_1, \sigma)$ and $\mathsf{Inv}(f_2, \Sigma, \sigma)$, then $f_2\{\mathbb{S}_2\} \subseteq S_2$.

Proof. We prove: for any k, $f_2\{\mathsf{cl}_k(\mathbb{S}_1, \Sigma)\} \subseteq \mathsf{cl}_k(S_1, \sigma)$. By induction over k.

- Base case: $k = 0$. From $f_1 \subseteq f_2$, $\mathsf{dom}(f_1) = \mathbb{S}_1$ and $f_1\{\mathbb{S}_1\} \subseteq S_1$, we are done.
- Inductive step: $k = n + 1$. For any b' and b'_1 such that $b' \in \mathsf{cl}_{n+1}(\mathbb{S}_1, \Sigma)$ and $f_2(b') = (b'_1, \_)$, we know there exist b, n and n' such that

$$b \in \mathsf{cl}_n(\mathbb{S}_1, \Sigma)\ , \qquad \Sigma(b)(n) = (b', n')\ \ .$$

Since $\mathsf{cl}_n(\mathbb{S}_1, \Sigma) \subseteq \mathsf{dom}(f_2)$, we know there exist b_1 and n_1 such that

$$f_2\langle(b, n)\rangle = (b_1, n_1)\ .$$

By the induction hypothesis, we know $f_2\{\mathsf{cl}_n(\mathbb{S}_1, \Sigma)\} \subseteq \mathsf{cl}_n(S_1, \sigma)$. Thus

$$b_1 \in \mathsf{cl}_n(S_1, \sigma)\ .$$

From $\mathsf{Inv}(f_2, \Sigma, \sigma)$, we know

$$f_2\langle(b', n')\rangle = \sigma(b_1)(n_1)\ .$$

Thus $b'_1 \in \mathsf{cl}_{n+1}(S_1, \sigma)$. Thus $f_2\{\mathsf{cl}_{n+1}(\mathbb{S}_1, \Sigma)\} \subseteq \mathsf{cl}_{n+1}(S_1, \sigma)$.

Thus we are done. □

Lemma 62. If $f_1 \subseteq f_2$, $\mathsf{dom}(f_1) = \mathbb{S}_1$, $\mathbb{S}_1 = \mathsf{cl}(\mathbb{S}_1, \Sigma)$, and $\mathsf{Inv}(f_2, \Sigma, \sigma)$, then $\mathsf{Inv}(f_1, \Sigma, \sigma)$.

Proof. For any b, n, b' and n' such that $(b, n) \in \mathsf{locs}(\Sigma)$ and $f_1\langle(b, n)\rangle = (b', n')$, since $f_1 \subseteq f_2$, we know $f_2\langle(b, n)\rangle = (b', n')$. Since $\mathsf{Inv}(f_2, \Sigma, \sigma)$, we know

$$(b', n') \in \mathsf{locs}(\sigma), \qquad \Sigma(b)(n) \overset{f_2}{\hookrightarrow} \sigma(b')(n') .$$

Suppose $\Sigma(b)(n) = v_1$ and $\sigma(b')(n') = v_2$. Thus

$$(v_1 \notin \mathfrak{U}) \wedge (v_1 = v_2) \;\vee\; v_1 \in \mathfrak{U} \wedge v_2 \in \mathfrak{U} \wedge f_2\langle v_1\rangle = v_2 .$$

If $v_1 \in \mathfrak{U}$, suppose $v_1 = (b_1, n_1)$. Since $b \in \mathsf{dom}(f_1) = \mathbb{S}_1$ and $\mathbb{S}_1 = \mathsf{cl}(\mathbb{S}_1, \Sigma)$, we know $b_1 \in \mathbb{S}_1$. Thus $f_1\langle v_1\rangle = v_2$. Thus we know

$$\Sigma(b)(n) \overset{f_1}{\hookrightarrow} \sigma(b')(n') .$$

Thus we are done. □

Lemma 63. If $\mu_E = (\mathbb{S}_E, S_E, f_E)$, $\mu_G = (\mathbb{S}_G, S_G, f_G)$, $\mathbb{S}_E \subseteq \mathbb{S}_G$, $f_E \subseteq f_G$, $\mathbb{S}_E \subseteq \mathsf{dom}(f_E)$, $f_G\{\mathbb{S}_G - \mathbb{F}\} \cap F = \emptyset$, $(\mathbb{S}_G - \mathbb{S}_E) \cap \mathbb{F} = \emptyset$ and $\mathsf{FPmatch}(\mu_E, (\Delta, \Sigma), (\delta, F))$, then $\mathsf{FPmatch}(\mu_G, (\Delta, \Sigma), (\delta, F))$.

Proof. From $\mathsf{FPmatch}(\mu_E, (\Delta, \Sigma), (\delta, F))$, we know

$$\begin{gathered}\delta.rs \cup \delta.ws \;\subseteq\; \lfloor F \rfloor \cup f_E\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_E \rfloor \rangle\!\rangle, \\ \delta.rs \cap f_E\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_E \rfloor \rangle\!\rangle \;\subseteq\; f_E\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta.rs \rangle\!\rangle, \\ \delta.ws \cap f_E\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_E \rfloor \rangle\!\rangle \;\subseteq\; f_E\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta.ws \rangle\!\rangle .\end{gathered}$$

Since $\mathbb{S}_E \subseteq \mathbb{S}_G$ and $f_E \subseteq f_G$, we know

$$\delta.rs \cup \delta.ws \;\subseteq\; \lfloor F \rfloor \cup f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_G \rfloor \rangle\!\rangle .$$

Since $(\mathbb{S}_G - \mathbb{S}_E) \cap \mathbb{F} = \emptyset$, we know

$$f_G\{\mathbb{S}_G - \mathbb{S}_E\} \subseteq f_G\{\mathbb{S}_G - \mathbb{F}\} .$$

Since $f_G\{\mathbb{S}_G - \mathbb{F}\} \cap F = \emptyset$, we know

$$f_G\{\mathbb{S}_G - \mathbb{S}_E\} \cap F = \emptyset .$$

Thus

$$f_G\langle\!\langle (\mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_G \rfloor) - (\mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_E \rfloor) \rangle\!\rangle \cap \lfloor F \rfloor = \emptyset .$$

From $f_E \subseteq f_G$ and $\mathbb{S}_E \subseteq \mathsf{dom}(f_E)$, we know

$$(f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_G \rfloor \rangle\!\rangle - f_E\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_E \rfloor \rangle\!\rangle) \cap \lfloor F \rfloor = \emptyset .$$

Thus

$$\begin{aligned}&\delta.rs \cap f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_G \rfloor \rangle\!\rangle \\ &= (\delta.rs \cap f_E\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_E \rfloor \rangle\!\rangle) \;\cup\; (\delta.rs \cap \lfloor F \rfloor \cap (f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_G \rfloor \rangle\!\rangle - f_E\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_E \rfloor \rangle\!\rangle)) \\ &\subseteq f_E\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta.rs \rangle\!\rangle \subseteq f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta.rs \rangle\!\rangle\end{aligned}$$

Similarly we can also prove

$$\delta.ws \cap f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \lfloor \mathbb{S}_G \rfloor \rangle\!\rangle \;\subseteq\; f_G\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta.ws \rangle\!\rangle$$

Thus we are done. □

Lemma 64. If $\mathbb{S}_G = \mathsf{cl}(\mathbb{S}_G, \Sigma_G)$, $\mathbb{S}_H = \mathsf{cl}(\mathbb{S}_G, \Sigma)$, $\mathbb{S}_F = \mathsf{cl}(\mathbb{S}_E, \Sigma)$, $(\mathbb{S}_G - \mathbb{S}_E) \cap \mathbb{F} = \emptyset$, $\Sigma_G \overset{\mathfrak{U} - \Delta_0.ws}{=\!=\!=} \Sigma$ and $\mathsf{blocks}(\Delta_0.ws) \subseteq \mathbb{F} \cup \mathbb{S}_E$, then $\mathbb{S}_H - \mathbb{S}_G \subseteq \mathbb{S}_F$.

Proof. For any $b \in \mathbb{S}_H - \mathbb{S}_G$, since $\mathbb{S}_H = \mathsf{cl}(\mathbb{S}_G, \Sigma)$ and $\mathbb{S}_G = \mathsf{cl}(\mathbb{S}_G, \Sigma_G)$, we know there exist $b_0, b_1, \ldots, b_k, n_0, n_1, \ldots, n_k, n, m$ such that

$$\begin{gathered}b_0 \in \mathbb{S}_G, \qquad \forall i \in [0..k).\, \Sigma(b_i)(n_i) = (b_{i+1}, n_{i+1}), \qquad \Sigma(b_k)(n_k) = (b, n), \\ 0 \le m \le k, \qquad \Sigma(b_m)(n_m) \ne \Sigma_G(b_m)(n_m), \qquad \forall i \in [0..m).\, \Sigma(b_i)(n_i) = \Sigma_G(b_i)(n_i) .\end{gathered}$$

Since $\Sigma_G \xLongequal{\mathfrak{U}-\Delta_0.ws} \Sigma$ and $\mathsf{blocks}(\Delta_0.ws) \subseteq \mathbb{F} \cup \mathbb{S}_E$, we know

$$b_m \in \mathbb{F} \cup \mathbb{S}_E\ .$$

Also we know $b_m \in \mathsf{cl}(\mathbb{S}_G, \Sigma_G) = \mathbb{S}_G$. Since $(\mathbb{S}_G - \mathbb{S}_E) \cap \mathbb{F} = \emptyset$, we know

$$b_m \in \mathbb{S}_E\ .$$

Thus $b \in \mathsf{cl}(\mathbb{S}_E, \Sigma) = \mathbb{S}_F$. Thus we are done. □

E.4 Proof of Flip of Simulation (Lemma 50)

To prove Lemma 50, we first define a relation $\simeq$ between source and target programs. We also define a relation $\simeq^i$ which is parameterized with an index i. This index describes the number of steps at the target that corresponds to zero steps of the source.

Definition 65 (Whole-Program Existential Relation)**.**
Let $\hat{\mathbb{P}} = \mathbf{let}\,\Gamma\,\mathbf{in}\ \mathsf{f}_1|\ldots|\mathsf{f}_m, \hat{P} = \mathbf{let}\,\Pi\,\mathbf{in}\ \mathsf{f}_1|\ldots|\mathsf{f}_m, \Gamma = \{(sl_1, ge_1, \gamma_1), \ldots, (sl_m, ge_m, \gamma_m)\}$ and $\Pi = \{(tl_1, ge'_1, \pi_1), \ldots, (tl_m, ge'_m, \pi_m)\}$.
$\hat{P} \simeq_\varphi \hat{\mathbb{P}}$ iff $\forall i \in \{1, \ldots, m\}.\ \varphi\langle\!\langle ge_i \rangle\!\rangle = ge'_i$, and

for any $\Sigma, \sigma, \mu, \widehat{\mathbb{W}}, \widehat{W}$, if $\mathsf{initM}(\varphi, \bigcup ge_i, \Sigma, \sigma)$, $\mu = (\mathsf{dom}(\Sigma), \mathsf{dom}(\sigma), \varphi)$, $(\hat{P}, \sigma) :\xLongrightarrow{load} \widehat{W}$, and $(\hat{\mathbb{P}}, \Sigma) :\xLongrightarrow{load} \widehat{\mathbb{W}}$, then $\widehat{\mathbb{W}} \simeq_\mu^{(\mathsf{emp},\mathsf{emp})} \widehat{W}$.

Here we define $\widehat{\mathbb{W}} \simeq_\mu^{(\Delta_0,\delta_0)} \widehat{W}$ as the largest relation such that whenever $\widehat{\mathbb{W}} \simeq_\mu^{(\Delta_0,\delta_0)} \widehat{W}$, then the following are true:

1. $\widehat{\mathbb{W}}.\mathsf{t} = \widehat{W}.\mathsf{t}$, and $\widehat{\mathbb{W}}.\mathbb{d} = \widehat{W}.\mathbb{d}$, and $\mathsf{wf}(\mu, \widehat{\mathbb{W}}.\Sigma, \widehat{W}.\sigma)$;
2. one of the following holds:
   a. there exist $\widehat{\mathbb{W}}'$, Δ, $\widehat{W}'$ and δ such that
      i. $\widehat{\mathbb{W}} \underset{\Delta}{\overset{\tau}{:\Rightarrow}}{}^{+} \widehat{\mathbb{W}}'$, and $\widehat{W} \underset{\delta}{\overset{\tau}{:\Rightarrow}}{}^{+} \widehat{W}'$, and
      ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta, ((\widehat{W}.T)(\widehat{W}.\mathsf{t})).F))$, and
      iii. $\widehat{\mathbb{W}}' \simeq_\mu^{(\Delta_0\cup\Delta,\,\delta_0\cup\delta)} \widehat{W}'$;
   b. or, there exist $\widehat{\mathbb{W}}'$, Δ, $\widehat{W}'$, δ, μ', $\mathbb{T}'$, T', $\mathbb{d}'$, o, Σ' and σ' such that
      i. $\widehat{\mathbb{W}} \underset{\Delta}{\overset{\tau}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\widehat{W} \underset{\delta}{\overset{\tau}{:\Rightarrow}}{}^{*} \widehat{W}'$, and
      ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta, ((\widehat{W}.T)(\widehat{W}.\mathsf{t})).F))$, and
      iii. for any $\mathsf{t}' \in \mathsf{dom}(\mathbb{T}')$, we have
         A. $\widehat{\mathbb{W}}' \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (\mathbb{T}', \mathsf{t}', \mathbb{d}', \Sigma')$, and $\widehat{W}' \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (T', \mathsf{t}', \mathbb{d}', \sigma')$, and $o \neq \tau$, and
         B. $(\mathbb{T}', \mathsf{t}', \mathbb{d}', \Sigma') \simeq_{\mu'}^{(\mathsf{emp},\mathsf{emp})} (T', \mathsf{t}', \mathbb{d}', \sigma')$;
   c. or, there exist $\widehat{\mathbb{W}}'$, Δ, $\widehat{W}'$ and δ such that
      i. $\widehat{\mathbb{W}} \underset{\Delta}{\overset{\tau}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\widehat{\mathbb{W}}' \underset{\mathsf{emp}}{\overset{\tau}{:\Longrightarrow}}$ **done**, and $\widehat{W} \underset{\delta}{\overset{\tau}{:\Rightarrow}}{}^{*} \widehat{W}'$, and $\widehat{W}' \underset{\mathsf{emp}}{\overset{\tau}{:\Longrightarrow}}$ **done**, and
      ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta, ((\widehat{W}.T)(\widehat{W}.\mathsf{t})).F))$.

With index. Also we define $\widehat{\mathbb{W}} \simeq_\mu^{(i,(\Delta_0,\delta_0))} \widehat{W}$ as the largest relation such that whenever $\widehat{\mathbb{W}} \simeq_\mu^{(i,(\Delta_0,\delta_0))} \widehat{W}$, then the following are true:

1. $\widehat{\mathbb{W}}.\mathsf{t} = \widehat{W}.\mathsf{t}$, and $\widehat{\mathbb{W}}.\mathbb{d} = \widehat{W}.\mathbb{d}$, and $\mathsf{wf}(\mu, \widehat{\mathbb{W}}.\Sigma, \widehat{W}.\sigma)$;
2. one of the following holds:
   a. there exist j, $\widehat{\mathbb{W}}'$, Δ, $\widehat{W}'$ and δ such that
      i. $\widehat{\mathbb{W}} \underset{\Delta}{\overset{\tau}{:\Rightarrow}}{}^{+} \widehat{\mathbb{W}}'$, and $\widehat{W} \underset{\delta}{\overset{\tau}{:\Rightarrow}}{}^{i+1} \widehat{W}'$, and
      ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta, ((\widehat{W}.T)(\widehat{W}.\mathsf{t})).F))$, and
      iii. $\widehat{\mathbb{W}}' \simeq_\mu^{(j,(\Delta_0\cup\Delta,\,\delta_0\cup\delta))} \widehat{W}'$;
   b. or, there exist j, $\widehat{\mathbb{W}}'$, Δ, $\widehat{W}'$, δ, μ', $\mathbb{T}'$, T', $\mathbb{d}'$, o, Σ' and σ' such that
      i. $\widehat{\mathbb{W}} \underset{\Delta}{\overset{\tau}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\widehat{W} \underset{\delta}{\overset{\tau}{:\Rightarrow}}{}^{i} \widehat{W}'$, and
      ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta, ((\widehat{W}.T)(\widehat{W}.\mathsf{t})).F))$, and
      iii. for any $\mathsf{t}' \in \mathsf{dom}(\mathbb{T}')$, we have
         A. $\widehat{\mathbb{W}}' \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (\mathbb{T}', \mathsf{t}', \mathbb{d}', \Sigma')$, and $\widehat{W}' \underset{\mathsf{emp}}{\overset{o}{:\Longrightarrow}} (T', \mathsf{t}', \mathbb{d}', \sigma')$, and $o \neq \tau$, and

B. $(\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma') \simeq_{\mu'}^{(j,(\mathsf{emp},\mathsf{emp}))} (T', \mathtt{t}', \mathbb{d}', \sigma')$;

c. or, there exist $\widehat{\mathbb{W}}'$, Δ, $\widehat{W}'$ and δ such that

i. $\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\widehat{\mathbb{W}}' \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **done**, and $\widehat{W} \overset{\tau}{\underset{\delta}{:\Rightarrow}}{}^{i} \widehat{W}'$, and $\widehat{W}' \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **done**, and

ii. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$.

Lemma 66. If $(\widehat{\mathbb{W}}, \Delta_0) \preccurlyeq_{\mu}^{i} (\widehat{W}, \delta_0)$ and $\mathsf{Safe}(\widehat{\mathbb{W}})$, then $\widehat{\mathbb{W}} \simeq_{\mu}^{(\Delta_0,\delta_0)} \widehat{W}$.

Lemma 67. If $\widehat{\mathbb{W}} \simeq_{\mu}^{(\Delta_0,\delta_0)} \widehat{W}$, then there exists i such that $\widehat{\mathbb{W}} \simeq_{\mu}^{(i,(\Delta_0,\delta_0))} \widehat{W}$.

Lemma 68. If $\widehat{\mathbb{W}} \simeq_{\mu}^{(i,(\Delta_0,\delta_0))} \widehat{W}$ and $\forall \mathtt{t}.\ \det((\widehat{W}.T)(\mathtt{t}).tl)$, then $(\widehat{W}, \delta_0) \leqslant_{\mu}^{i} (\widehat{\mathbb{W}}, \Delta_0)$.

Then, for Lemma 24, from **let** Γ **in** $\mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m \preccurlyeq_{ge,\varphi}$ **let** Π **in** $\mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m$, we know

$$\forall i \in \{1, \ldots, m\}.\ \mathsf{dom}(sl_i.\mathsf{InitCore}(\gamma_i)) = \mathsf{dom}(tl_i.\mathsf{InitCore}(\pi_i))\ .$$

For any $\mathbb{T}$, T, $\mathtt{t}$, $\mathbb{d}$, Σ, σ, μ, $\mathbb{S}$ and S, if $(\hat{\mathbb{P}}, \Sigma) \overset{load}{:\Longrightarrow} (\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma)$, $(\hat{P}, \sigma) \overset{load}{:\Longrightarrow} (T, \mathtt{t}, \mathbb{d}, \sigma)$, $ge \subseteq \mathsf{locs}(\Sigma)$, $\mathsf{dom}(\Sigma) = \mathbb{S} \subseteq \mathsf{dom}(\varphi)$, $\mathsf{cl}(\mathbb{S}, \Sigma) = \mathbb{S}$, $\mathsf{locs}(\sigma) = \varphi\langle\!\langle \mathsf{locs}(\Sigma) \rangle\!\rangle$, $\mathsf{Inv}(\varphi, \Sigma, \sigma)$ and $\mu = (\mathbb{S}, \varphi\{\mathbb{S}\}, \varphi|_{\mathbb{S}})$, from **let** Γ **in** $\mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m \preccurlyeq_{ge,\varphi}$ **let** Π **in** $\mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m$, we know there exists $i \in \mathtt{index}$ such that

$$((\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma), \mathsf{emp}) \preccurlyeq_{\mu}^{i} ((T, \mathtt{t}, \mathbb{d}, \sigma), \mathsf{emp})\ .$$

By Lemmas 66, 67 and 68, we know there exists j such that $((T, \mathtt{t}, \mathbb{d}, \sigma), \mathsf{emp}) \leqslant_{\mu}^{j} ((\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma), \mathsf{emp})$. Thus we are done.

Proof of Lemma 66. We prove the following strengthened result:

$$\text{If } \widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta_0'}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}_0,\ \widehat{W} \overset{\tau}{\underset{\delta_0'}{:\Rightarrow}}{}^{*} \widehat{W}_0 \text{ and } (\widehat{\mathbb{W}}_0, \Delta_0 \cup \Delta_0') \preccurlyeq_{\mu}^{i} (\widehat{W}_0, \delta_0 \cup \delta_0'), \text{ then } \widehat{\mathbb{W}} \simeq_{\mu}^{(\Delta_0,\delta_0)} \widehat{W}.$$

By co-induction. Then, by induction over i (using the well-founded ordering of $\mathtt{index}$).

1. $i = 0$.

From $\mathsf{Safe}(\widehat{\mathbb{W}})$, we know $\neg(\widehat{\mathbb{W}}_0 \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **abort**), we know there exists $\widehat{\mathbb{W}}'$ and Δ' such that

$$\widehat{\mathbb{W}}_0 \overset{\tau}{\underset{\Delta'}{:\Rightarrow}} \widehat{\mathbb{W}}',\ \text{ or }\ \exists o \neq \tau.\ \widehat{\mathbb{W}}_0 \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} \widehat{\mathbb{W}}',\ \text{ or }\ \widehat{\mathbb{W}}_0 \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}} \textbf{done}$$

a. If $\widehat{\mathbb{W}}_0 \overset{\tau}{\underset{\Delta'}{:\Rightarrow}} \widehat{\mathbb{W}}'$, from $(\widehat{\mathbb{W}}_0, \Delta_0 \cup \Delta_0') \preccurlyeq_{\mu}^{i} (\widehat{W}_0, \delta_0 \cup \delta_0')$, we know there exist $\widehat{W}'$, δ' and j such that

i. $\widehat{W}_0 \overset{\tau}{\underset{\delta'}{:\Rightarrow}}{}^{+} \widehat{W}'$, and

ii. $\mathsf{FPmatch}(\Delta_0 \cup \Delta_0' \cup \Delta', \delta_0 \cup \delta_0' \cup \delta', \mu, (\widehat{\mathbb{W}}_0.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, and

iii. $(\widehat{\mathbb{W}}', \Delta_0 \cup \Delta_0' \cup \Delta') \preccurlyeq_{\mu}^{j} (\widehat{W}', \delta_0 \cup \delta_0' \cup \delta')$.

Thus we know

$$\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta_0' \cup \Delta'}{:\Longrightarrow}}{}^{+} \widehat{\mathbb{W}}', \quad \text{and} \quad \widehat{W} \overset{\tau}{\underset{\delta_0' \cup \delta'}{:\Longrightarrow}}{}^{+} \widehat{W}'\ .$$

From $\mathsf{FPmatch}(\Delta_0 \cup \Delta_0' \cup \Delta', \delta_0 \cup \delta_0' \cup \delta', \mu, (\widehat{\mathbb{W}}_0.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, by Lemma 57, we know

$$\mathsf{FPmatch}(\Delta_0 \cup \Delta_0' \cup \Delta', \delta_0 \cup \delta_0' \cup \delta', \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))\ .$$

Since $\widehat{\mathbb{W}}' \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}{}^{*} \widehat{\mathbb{W}}'$, $\widehat{W}' \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}{}^{*} \widehat{W}'$ and $(\widehat{\mathbb{W}}', \Delta_0 \cup \Delta_0' \cup \Delta') \preccurlyeq_{\mu}^{j} (\widehat{W}', \delta_0 \cup \delta_0' \cup \delta')$, by the co-induction hypothesis, we know

$$\widehat{\mathbb{W}}' \simeq_{\mu}^{(\Delta_0 \cup \Delta_0' \cup \Delta',\, \delta_0 \cup \delta_0' \cup \delta')} \widehat{W}'\ .$$

Thus we are done.

b. If $o \neq \tau$ and $\widehat{\mathbb{W}}_0 \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} \widehat{\mathbb{W}}'$, by the operational semantics, we know there exist $\mathbb{T}'$, $\mathbb{d}'$ and Σ' such that for any $\mathtt{t}' \in \mathsf{dom}(\mathbb{T}')$, we have

$$\widehat{\mathbb{W}}_0 \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} (\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma')\ .$$

From $(\widehat{\mathbb{W}}_0, \Delta_0 \cup \Delta_0') \preccurlyeq_{\mu}^{i} (\widehat{W}_0, \delta_0 \cup \delta_0')$, we know there exist $\widehat{W}'$, δ', μ', T', σ' and j such that for any $\mathtt{t}' \in \mathsf{dom}(\mathbb{T}')$, we have

i. $\widehat{W}_0 \overset{\tau}{\underset{\delta'}{:\Rightarrow}}{}^{*} \widehat{W}'$, and $\widehat{W}' \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} (T', \mathtt{t}', \mathbb{d}', \sigma')$, and

ii. $\mathsf{FPmatch}(\Delta_0 \cup \Delta_0', \delta_0 \cup \delta_0' \cup \delta', \mu, (\widehat{\mathbb{W}}_0.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, and

iii. $((\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma'), \mathsf{emp}) \preccurlyeq_{\mu'}^{j} ((T', \mathtt{t}', \mathbb{d}', \sigma'), \mathsf{emp})$.

Since $((\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma'), \mathsf{emp}) \preccurlyeq^{j}_{\mu'} ((T', \mathtt{t}', \mathbb{d}', \sigma'), \mathsf{emp})$, by the co-induction hypothesis, we know

$$(\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma') \simeq^{(\mathsf{emp},\mathsf{emp})}_{\mu'} (T', \mathtt{t}', \mathbb{d}', \sigma') \ .$$

Thus we are done.

c. If $\widehat{\mathbb{W}}_0 :\overset{\tau}{\underset{\mathsf{emp}}{\Longrightarrow}} \textbf{done}$, from $(\widehat{\mathbb{W}}_0, \Delta_0 \cup \Delta_0') \preccurlyeq^{i}_{\mu} (\widehat{W}_0, \delta_0 \cup \delta_0')$, we know there exist $\widehat{W}'$ and δ' such that

i. $\widehat{W}_0 :\overset{\tau}{\underset{\delta'}{\Longrightarrow}}{}^{*} \widehat{W}'$, and $\widehat{W}' :\overset{\tau}{\underset{\mathsf{emp}}{\Longrightarrow}} \textbf{done}$, and

ii. $\mathsf{FPmatch}(\Delta_0 \cup \Delta_0', \delta_0 \cup \delta_0' \cup \delta', \mu, (\widehat{\mathbb{W}}_0.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$.

Thus we are done.

2. $i > 0$.

From $\mathsf{Safe}(\widehat{\mathbb{W}})$, we know $\neg(\widehat{\mathbb{W}}_0 :\overset{\tau}{\underset{\mathsf{emp}}{\Longrightarrow}} \textbf{abort})$, we know there exists $\widehat{\mathbb{W}}'$ and Δ' such that

$$\widehat{\mathbb{W}}_0 :\overset{\tau}{\underset{\Delta'}{\Longrightarrow}} \widehat{\mathbb{W}}', \text{ or } \exists o \neq \tau.\ \widehat{\mathbb{W}}_0 :\overset{o}{\underset{\mathsf{emp}}{\Longrightarrow}} \widehat{\mathbb{W}}', \text{ or } \widehat{\mathbb{W}}_0 :\overset{\tau}{\underset{\mathsf{emp}}{\Longrightarrow}} \textbf{done}$$

a. If $\widehat{\mathbb{W}}_0 :\overset{\tau}{\underset{\Delta'}{\Longrightarrow}} \widehat{\mathbb{W}}'$, from $(\widehat{\mathbb{W}}_0, \Delta_0 \cup \Delta_0') \preccurlyeq^{i}_{\mu} (\widehat{W}_0, \delta_0 \cup \delta_0')$, we know one of the following holds:

i. there exists j such that

A. $j < i$, and

B. $(\widehat{\mathbb{W}}', \Delta_0 \cup \Delta_0' \cup \Delta') \preccurlyeq^{j}_{\mu} (\widehat{W}_0, \delta_0 \cup \delta_0')$;

ii. or, there exist $\widehat{W}'$, δ' and j such that

A. $\widehat{W}_0 :\overset{\tau}{\underset{\delta'}{\Longrightarrow}}{}^{+} \widehat{W}'$, and

B. $\mathsf{FPmatch}(\Delta_0 \cup \Delta_0' \cup \Delta', \delta_0 \cup \delta_0' \cup \delta', \mu, (\widehat{\mathbb{W}}_0.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, and

C. $(\widehat{\mathbb{W}}', \Delta_0 \cup \Delta_0' \cup \Delta') \preccurlyeq^{j}_{\mu} (\widehat{W}', \delta_0 \cup \delta_0' \cup \delta')$.

For case (i), we know

$$\widehat{\mathbb{W}} :\overset{\tau}{\underset{\Delta_0' \cup \Delta'}{\Longrightarrow}}{}^{*} \widehat{\mathbb{W}}', \quad \text{and} \quad \widehat{W} :\overset{\tau}{\underset{\delta_0' \cup \delta'}{\Longrightarrow}}{}^{*} \widehat{W}' \ .$$

Since $j < i$, by the induction hypothesis, we know

$$\widehat{\mathbb{W}} \simeq^{(\Delta_0, \delta_0)}_{\mu} \widehat{W} \ .$$

For case (ii), the proof is similar to the above 1(a).

b. If $\exists o \neq \tau.\ \widehat{\mathbb{W}}_0 :\overset{o}{\underset{\mathsf{emp}}{\Longrightarrow}} \widehat{\mathbb{W}}''$, the proof is similar to the above 1(b).

c. If $\widehat{\mathbb{W}}_0 :\overset{\tau}{\underset{\mathsf{emp}}{\Longrightarrow}} \textbf{done}$, the proof is similar to the above 1(c).

Thus we are done. □

Proof of Lemma 67. We prove $\widehat{\mathbb{W}} \simeq^{(i,(\Delta_0, \delta_0))}_{\mu} \widehat{W}$ holds if $\widehat{\mathbb{W}}.\mathtt{t} = \widehat{W}.\mathtt{t}$, and $\widehat{\mathbb{W}}.\mathbb{d} = \widehat{W}.\mathbb{d}$, and $\mathsf{wf}(\mu, \widehat{\mathbb{W}}.\Sigma)$ and one of the following holds:

1. there exist $\widehat{\mathbb{W}}'$, Δ, $\widehat{W}'$ and δ such that

a. $\widehat{\mathbb{W}} :\overset{\tau}{\underset{\Delta}{\Longrightarrow}}{}^{+} \widehat{\mathbb{W}}'$, and $\widehat{W} :\overset{\tau}{\underset{\delta}{\Longrightarrow}}{}^{i+1} \widehat{W}'$, and

b. $\mathsf{FPmatch}(\Delta_0 \cup \Delta, \delta_0 \cup \delta, \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, and

c. $\widehat{\mathbb{W}}' \simeq^{(\Delta_0 \cup \Delta, \delta_0 \cup \delta)}_{\mu} \widehat{W}'$;

2. or, there exist $\widehat{\mathbb{W}}'$, Δ, $\widehat{W}'$, δ, μ', $\mathbb{T}'$, T', $\mathbb{d}'$, o, Σ' and σ' such that

a. $\widehat{\mathbb{W}} :\overset{\tau}{\underset{\Delta}{\Longrightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\widehat{W} :\overset{\tau}{\underset{\delta}{\Longrightarrow}}{}^{i} \widehat{W}'$, and

b. $\mathsf{FPmatch}(\Delta_0 \cup \Delta, \delta_0 \cup \delta, \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, and

c. for any $\mathtt{t}' \in \mathsf{dom}(\mathbb{T}')$, we have

i. $\widehat{\mathbb{W}}' :\overset{o}{\underset{\mathsf{emp}}{\Longrightarrow}} (\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma')$, and $\widehat{W}' :\overset{o}{\underset{\mathsf{emp}}{\Longrightarrow}} (T', \mathtt{t}', \mathbb{d}', \sigma')$, and $o \neq \tau$, and

ii. $(\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma') \simeq^{(\mathsf{emp},\mathsf{emp})}_{\mu'} (T', \mathtt{t}', \mathbb{d}', \sigma')$;

3. or, there exist $\widehat{\mathbb{W}}'$, Δ, $\widehat{W}'$ and δ such that

a. $\widehat{\mathbb{W}} :\overset{\tau}{\underset{\Delta}{\Longrightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\widehat{\mathbb{W}}' :\overset{\tau}{\underset{\mathsf{emp}}{\Longrightarrow}} \textbf{done}$, and $\widehat{W} :\overset{\tau}{\underset{\delta}{\Longrightarrow}}{}^{i} \widehat{W}'$, and $\widehat{W}' :\overset{\tau}{\underset{\mathsf{emp}}{\Longrightarrow}} \textbf{done}$, and

b. $\mathsf{FPmatch}(\Delta_0 \cup \Delta, \delta_0 \cup \delta, \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$.

We prove the above by co-induction. Thus we are done. □

Proof of Lemma 68. By co-induction. We need to prove the following:

1. $\forall \widehat{W}', \delta.$ if $\widehat{W} :\overset{\tau}{\underset{\delta}{\Longrightarrow}} \widehat{W}'$, then one of the following holds:
   a. there exists j such that
      i. $j < i$, and
      ii. $(\widehat{W}', \delta_0 \cup \delta) \leqslant^{j}_{\mu} (\widehat{\mathbb{W}}, \Delta_0)$;
   b. there exist $\widehat{\mathbb{W}}'$, Δ and j such that
      i. $\widehat{\mathbb{W}} :\overset{\tau}{\underset{\Delta}{\Longrightarrow}}{}^{+} \widehat{\mathbb{W}}'$, and
      ii. $\mathsf{FPmatch}(\Delta_0 \cup \Delta, \delta_0 \cup \delta, \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, and
      iii. $(\widehat{W}', \delta_0 \cup \delta) \leqslant^{j}_{\mu} (\widehat{\mathbb{W}}', \Delta_0 \cup \Delta)$.

Proof. By inversion of $\widehat{\mathbb{W}} \simeq^{(i,(\Delta_0,\delta_0))}_{\mu} \widehat{W}$, we know one of the following holds:

(A) there exist $\widehat{\mathbb{W}}_1$, Δ, $\widehat{W}_1$, $\widehat{W}_2$, δ_1, δ_2 and i' such that
   (1) $\widehat{\mathbb{W}} :\overset{\tau}{\underset{\Delta}{\Longrightarrow}}{}^{+} \widehat{\mathbb{W}}_1$, and $\widehat{W} :\overset{\tau}{\underset{\delta_1}{\Longrightarrow}} \widehat{W}_1$, and $\widehat{W}_1 :\overset{\tau}{\underset{\delta_2}{\Longrightarrow}}{}^{i} \widehat{W}_2$, and
   (2) $\mathsf{FPmatch}(\Delta_0 \cup \Delta, \delta_0 \cup \delta_1 \cup \delta_2, \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, and
   (3) $\widehat{\mathbb{W}}_1 \simeq^{(i',(\Delta_0\cup\Delta,\,\delta_0\cup\delta_1\cup\delta_2))}_{\mu} \widehat{W}_2$.

$$\begin{tikzcd}
\widehat{\mathbb{W}} \arrow[r] \arrow[d, no head] & {} \arrow[r] & {} \arrow[r] & \widehat{\mathbb{W}}_1 \arrow[d, dashed, no head] \arrow[dl, dashed, no head] \arrow[dll, dashed, no head] \\
\widehat{W} \arrow[r] & {} \arrow[r] & {} \arrow[r] & \widehat{W}_2
\end{tikzcd}$$

First, from (3), by the co-induction hypothesis, we know

$$(\widehat{W}_2, \delta_0 \cup \delta_1 \cup \delta_2) \leqslant^{i'}_{\mu} (\widehat{\mathbb{W}}_1, \Delta_0 \cup \Delta) \tag{E.3}$$

Then we prove:

$$\forall \widehat{W}_0, n_0, n_1, \delta', \delta''.\ (\widehat{W}_1 :\overset{\tau}{\underset{\delta'}{\Longrightarrow}}{}^{n_0} \widehat{W}_0) \wedge (\widehat{W}_0 :\overset{\tau}{\underset{\delta''}{\Longrightarrow}}{}^{n_1} \widehat{W}_2) \implies (\widehat{W}_0, \delta_0 \cup \delta_1 \cup \delta') \leqslant^{i'+n_1}_{\mu} (\widehat{\mathbb{W}}_1, \Delta_0 \cup \Delta) \tag{E.4}$$

By induction over n_1.

i. $n_1 = 0$. Thus $\widehat{W}_0 = \widehat{W}_2$. Since $\forall \mathtt{t}.\ \mathsf{det}((\widehat{W}.T)(\mathtt{t}).tl)$, we know $\delta' = \delta_2$. By (E.3), we are done.

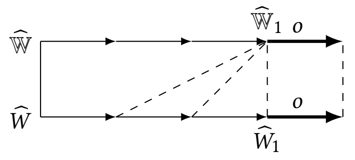
ii. $n_1 = m + 1$. Thus there exist $\widehat{W}'_0$, δ'_0 and δ'_1 such that $\delta'' = \delta'_0 \cup \delta'_1$, $\widehat{W}_0 :\overset{\tau}{\underset{\delta'_0}{\Longrightarrow}} \widehat{W}'_0$ and $\widehat{W}'_0 :\overset{\tau}{\underset{\delta'_1}{\Longrightarrow}}{}^{m} \widehat{W}_2$. By the induction hypothesis, we know
$$(\widehat{W}'_0, \delta_0 \cup \delta_1 \cup \delta' \cup \delta'_0) \leqslant^{i'+m}_{\mu} (\widehat{\mathbb{W}}_1, \Delta_0 \cup \Delta)\ .$$
To prove $(\widehat{W}_0, \delta_0 \cup \delta_1 \cup \delta') \leqslant^{i'+m+1}_{\mu} (\widehat{\mathbb{W}}_1, \Delta_0 \cup \Delta)$, by co-induction. Since $\forall \mathtt{t}.\ \mathsf{det}((\widehat{W}.T)(\mathtt{t}).tl)$, we can finish the proof.

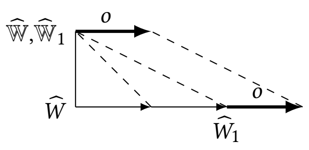
Thus we are done of (E.4). Thus, we know (E.5) holds.

$$(\widehat{W}_1, \delta_0 \cup \delta_1) \leqslant^{i'+i}_{\mu} (\widehat{\mathbb{W}}_1, \Delta_0 \cup \Delta) \tag{E.5}$$

Finally, since $\forall \mathtt{t}.\ \mathsf{det}((\widehat{W}.T)(\mathtt{t}).tl)$, we know $\widehat{W}_1 = \widehat{W}'$ and $\delta_1 = \delta$. From (2), we know
$$\mathsf{FPmatch}(\Delta_0 \cup \Delta, \delta_0 \cup \delta_1, \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F)).$$
From (1) and (E.5), we are done.

(B) there exist i', $\widehat{\mathbb{W}}_1$, Δ, $\widehat{W}_1$, δ_1, μ', $\mathbb{T}'$, T', $\mathbb{d}'$, o, Σ' and σ' such that
   (1) $\widehat{\mathbb{W}} :\overset{\tau}{\underset{\Delta}{\Longrightarrow}}{}^{*} \widehat{\mathbb{W}}_1$, and $\widehat{W} :\overset{\tau}{\underset{\delta_1}{\Longrightarrow}}{}^{i} \widehat{W}_1$, and
   (2) $\mathsf{FPmatch}(\Delta_0 \cup \Delta, \delta_0 \cup \delta_1, \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, and
   (3) for any $\mathtt{t}' \in \mathsf{dom}(\mathbb{T}')$, we have
      A. $\widehat{\mathbb{W}}_1 :\overset{o}{\underset{\mathsf{emp}}{\Longrightarrow}} (\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma')$, and $\widehat{W}_1 :\overset{o}{\underset{\mathsf{emp}}{\Longrightarrow}} (T', \mathtt{t}', \mathbb{d}', \sigma')$, and $o \neq \tau$, and
      B. $(\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma') \simeq^{(i',(\mathsf{emp},\mathsf{emp}))}_{\mu'} (T', \mathtt{t}', \mathbb{d}', \sigma')$.



or


First, from (3), by the co-induction hypothesis, we know

$$((T', \mathtt{t}', \mathbb{d}', \sigma'), \mathtt{emp}) \leqslant^{i'}_{\mu'} ((\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma'), \mathtt{emp})\ . \tag{E.6}$$

Since $\forall \mathtt{t}.\ \mathsf{det}((\widehat{W}.T)(\mathtt{t}).tl)$, we know

$$(\widehat{W}_1, \delta_0 \cup \delta_1) \leqslant^{0}_{\mu} (\widehat{\mathbb{W}}_1, \Delta_0 \cup \Delta)\ . \tag{E.7}$$

Also $i > 0$ and there exists δ_1' such that $\delta_1 = \delta \cup \delta_1'$,

$$\widehat{W} \overset{\tau}{\underset{\delta}{:\Rightarrow}} \widehat{W}' \qquad \text{and} \qquad \widehat{W}' \overset{\tau}{\underset{\delta_1'}{:\Rightarrow}}{}^{i-1}\, \widehat{W}_1\ .$$

Similarly to the above case (A), we can prove (E.4). Then we know

$$(\widehat{W}', \delta_0 \cup \delta) \leqslant^{i-1}_{\mu} (\widehat{\mathbb{W}}_1, \Delta_0 \cup \Delta) \tag{E.8}$$

Since $\forall \mathtt{t}.\ \mathsf{det}((\widehat{W}.T)(\mathtt{t}).tl)$, from (1) and (2), we are done.

(C) there exist $\widehat{\mathbb{W}}_1$, Δ, $\widehat{W}_1$ and δ_1 such that

(1) $\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^* \widehat{\mathbb{W}}_1$, and $\widehat{\mathbb{W}}_1 \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **done**, and $\widehat{W} \overset{\tau}{\underset{\delta_1}{:\Rightarrow}}{}^i \widehat{W}_1$, and $\widehat{W}_1 \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **done**, and

(2) $\mathsf{FPmatch}(\Delta_0 \cup \Delta, \delta_0 \cup \delta_1, \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$.

Since $\forall \mathtt{t}.\ \mathsf{det}((\widehat{W}.T)(\mathtt{t}).tl)$, we know

$$(\widehat{W}_1, \delta_0 \cup \delta_1) \leqslant^{0}_{\mu} (\widehat{\mathbb{W}}_1, \Delta_0 \cup \Delta)\ . \tag{E.9}$$

Also $i > 0$ and there exists δ_1' such that $\delta_1 = \delta \cup \delta_1'$,

$$\widehat{W} \overset{\tau}{\underset{\delta}{:\Rightarrow}} \widehat{W}' \qquad \text{and} \qquad \widehat{W}' \overset{\tau}{\underset{\delta_1'}{:\Rightarrow}}{}^{i-1}\, \widehat{W}_1\ .$$

Similarly to the above case, we can prove (E.4). Then we know

$$(\widehat{W}', \delta_0 \cup \delta) \leqslant^{i-1}_{\mu} (\widehat{\mathbb{W}}_1, \Delta_0 \cup \Delta) \tag{E.10}$$

Since $\forall \mathtt{t}.\ \mathsf{det}((\widehat{W}.T)(\mathtt{t}).tl)$, from (1) and (2), we are done.

Thus we are done. □

2. $\forall T', \mathbb{d}', \sigma', o.$ if $o \neq \tau$ and $\exists \mathtt{t}'.\ \widehat{W} \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} (T', \mathtt{t}', \mathbb{d}', \sigma')$, then there exist $\widehat{\mathbb{W}}', \Delta, \mu', \mathbb{T}', \Sigma'$ and j such that for any $\mathtt{t}' \in \mathsf{dom}(T')$, we have
   a. $\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^* \widehat{\mathbb{W}}'$, and $\widehat{\mathbb{W}}' \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} (\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma')$, and
   b. $\mathsf{FPmatch}(\Delta_0 \cup \Delta, \delta_0, \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, and
   c. $((T', \mathtt{t}', \mathbb{d}', \sigma'), \mathtt{emp}) \leqslant^{j}_{\mu'} ((\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma'), \mathtt{emp})$.

Proof. By inversion of $\widehat{\mathbb{W}} \simeq^{(i,(\Delta_0,\delta_0))}_{\mu} \widehat{W}$, since $\forall \mathtt{t}.\ \mathsf{det}((\widehat{W}.T)(\mathtt{t}).tl)$, we know there exist i', $\widehat{\mathbb{W}}_1$, Δ, μ', $\mathbb{T}'$, T', $\mathbb{d}'$, o, Σ' and σ' such that

(1) $\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^* \widehat{\mathbb{W}}_1$, and

(2) $\mathsf{FPmatch}(\Delta_0 \cup \Delta, \delta_0, \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, and

(3) for any $\mathtt{t}' \in \mathsf{dom}(\mathbb{T}')$, we have
   i. $\widehat{\mathbb{W}}_1 \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} (\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma')$, and $\widehat{W} \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} (T', \mathtt{t}', \mathbb{d}', \sigma')$, and $o \neq \tau$, and
   ii. $(\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma') \simeq^{(i',(\mathsf{emp},\mathsf{emp}))}_{\mu'} (T', \mathtt{t}', \mathbb{d}', \sigma')$.

From (3), by the co-induction hypothesis, we know

$$((T', \mathtt{t}', \mathbb{d}', \sigma'), \mathtt{emp}) \leqslant^{i'}_{\mu'} ((\mathbb{T}', \mathtt{t}', \mathbb{d}', \Sigma'), \mathtt{emp})\ .$$

Thus we are done. □

3. if $\widehat{W} \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **done**, then there exist $\widehat{\mathbb{W}}'$ and Δ such that

a. $\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Longrightarrow}}{}^* \widehat{\mathbb{W}}'$, and $\widehat{\mathbb{W}}' \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **done**, and

b. $\mathsf{FPmatch}(\Delta_0 \cup \Delta, \delta_0, \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$.

Proof. By inversion of $\widehat{\mathbb{W}} \simeq_{\mu}^{(i,(\Delta_0,\delta_0))} \widehat{W}$, since $\forall \mathtt{t}.\ \mathsf{det}((\widehat{W}.T)(\mathtt{t}).tl)$, we know there exist $\widehat{\mathbb{W}}_1$ and Δ such that

(1) $\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Longrightarrow}}{}^* \widehat{\mathbb{W}}_1$, and $\widehat{\mathbb{W}}_1 \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **done**, and $\widehat{W} \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}}$ **done**, and

(2) $\mathsf{FPmatch}(\Delta_0 \cup \Delta, \delta_0, \mu, (\widehat{\mathbb{W}}.\Sigma, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$.

Thus we are done. □

Thus we have finished the proof. □

E.5 Proof of NPDRF Preservation (Lemma 51)

We only need to prove: for any $\hat{\mathbb{P}}$, $\hat{P}$, φ, Σ and σ, if $\hat{P} \leqslant_{\varphi} \hat{\mathbb{P}}$, $\mathsf{initM}(\varphi, \mathsf{GE}(\hat{\mathbb{P}}.\Gamma), \Sigma, \sigma)$, and $(\hat{P}, \sigma) \Longmapsto \mathsf{Race}$, then $(\hat{\mathbb{P}}, \sigma) \Longmapsto \mathsf{Race}$.

From $(\hat{P}, \sigma) \Longmapsto \mathsf{Race}$, we have two cases:

1. there exist $\widehat{W}$, $\mathtt{t}_1$, $\mathtt{t}_2$, δ_1, d_1 δ_2, d_2 such that $(\hat{P}, \sigma) \overset{load}{:\Longrightarrow} \widehat{W}$, $\mathtt{t}_1 \neq \mathtt{t}_2$, $\mathsf{NPpred}(\widehat{W}, \mathtt{t}_1, (\delta_1, d_1))$, $\mathsf{NPpred}(\widehat{W}, \mathtt{t}_2, (\delta_2, d_2))$ and $\neg((\delta_1, d_1) \smile (\delta_2, d_2))$.

   From $\mathsf{NPpred}(\widehat{W}, \mathtt{t}_1, (\delta_1, d_1))$ and $\mathsf{NPpred}(\widehat{W}, \mathtt{t}_2, (\delta_2, d_2))$, we know there exist T, $\mathbb{d}$, σ, tl_1, π_1, F_1, κ_1, κ_1', σ_1', tl_2, π_2, F_2, κ_2, κ_2', σ_2' such that

   $$\widehat{W} = (T, \_, \mathbb{d}, \sigma), \quad T(\mathtt{t}_1) = (tl_1, (\pi_1, F_1), \kappa_1), \quad (\pi_1, F_1) \vdash (\kappa_1, \sigma) \overset{\tau}{\underset{\delta_1}{\longmapsto}}{}^* (\kappa_1', \sigma_1'), \quad \mathbb{d}(\mathtt{t}_1) = d_1,$$
   $$T(\mathtt{t}_2) = (tl_2, (\pi_2, F_2), \kappa_2), \quad (\pi_2, F_2) \vdash (\kappa_2, \sigma) \overset{\tau}{\underset{\delta_2}{\longmapsto}}{}^* (\kappa_2', \sigma_2'), \quad \mathbb{d}(\mathtt{t}_1) = d_2$$

   By the operational semantics, we know

   $$(\hat{P}, \sigma) \overset{load}{:\Longrightarrow} (T, \mathtt{t}_1, \mathbb{d}, \sigma), \quad (\hat{P}, \sigma) \overset{load}{:\Longrightarrow} (T, \mathtt{t}_2, \mathbb{d}, \sigma), \quad d_1 = d_2 = 0,$$
   $$(T, \mathtt{t}_1, \mathbb{d}, \sigma) \overset{\tau}{\underset{\delta_1}{:\Longrightarrow}}{}^* (T\{\mathtt{t}_1 \leadsto (tl_1, (\pi_1, F_1), \kappa_1'), \mathtt{t}_1, \mathbb{d}, \sigma_1'), \quad (T, \mathtt{t}_2, \mathbb{d}, \sigma) \overset{\tau}{\underset{\delta_2}{:\Longrightarrow}}{}^* (T\{\mathtt{t}_2 \leadsto (tl_2, (\pi_2, F_2), \kappa_2'), \mathtt{t}_2, \mathbb{d}, \sigma_2')$$

   From $\hat{P} \leqslant_{\varphi} \hat{\mathbb{P}}$, we know there exist $\mathtt{f}_1, \ldots, \mathtt{f}_n$, $\Gamma = \{(sl_1, \gamma_1), \ldots, (sl_m, \gamma_m)\}$ and $\Pi = \{(tl_1, \pi_1), \ldots, (tl_m, \pi_m)\}$, such that $\hat{\mathbb{P}} =$ **let** Γ **in** $\mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m$, $\hat{P} =$ **let** Π **in** $\mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m$ and $\forall i \in \{1, \ldots, m\}.\ \mathsf{dom}(sl_i.\mathsf{InitCore}(\gamma_i)) = \mathsf{dom}(tl_i.\mathsf{InitCore}(\pi_i))$. Thus we know there exists $\mathbb{T}$ such that

   $$(\hat{\mathbb{P}}, \Sigma) \overset{load}{:\Longrightarrow} (\mathbb{T}, \mathtt{t}_1, \mathbb{d}, \Sigma), \quad (\hat{\mathbb{P}}, \Sigma) \overset{load}{:\Longrightarrow} (\mathbb{T}, \mathtt{t}_2, \mathbb{d}, \Sigma).$$

   Let $\mathbb{S} = \mathsf{dom}(\Sigma)$, $S = \varphi\{\!\{\mathbb{S}\}\!\}$ and $\mu = (\mathbb{S}, S, \varphi|_{\mathbb{S}})$. We know there exists $i_1, i_2 \in \mathtt{index}$ such that

   $$((T, \mathtt{t}_1, \mathbb{d}, \sigma), \mathsf{emp}) \leqslant_{\mu}^{i_1} ((\mathbb{T}, \mathtt{t}_1, \mathbb{d}, \Sigma), \mathsf{emp}), \quad ((T, \mathtt{t}_2, \mathbb{d}, \sigma), \mathsf{emp}) \leqslant_{\mu}^{i_2} ((\mathbb{T}, \mathtt{t}_2, \mathbb{d}, \Sigma), \mathsf{emp})\ .$$

   By Lemma 69, we know there exist $\widehat{W}_1'$, δ_1', $\widehat{\mathbb{W}}_1'$, Δ_1' $\widehat{W}_2'$, δ_2', $\widehat{\mathbb{W}}_2'$ and Δ_2' such that

   $$(T, \mathtt{t}_1, \mathbb{d}, \sigma) \overset{\tau}{\underset{\delta_1'}{:\Longrightarrow}}{}^* \widehat{W}_1', \quad (\mathbb{T}, \mathtt{t}_1, \mathbb{d}, \Sigma) \overset{\tau}{\underset{\Delta_1'}{:\Longrightarrow}}{}^* \widehat{\mathbb{W}}_1', \quad \delta_1 \subseteq \delta_1', \quad \mathsf{FPmatch}(\mu, (\Delta_1', \Sigma), (\delta_1', F_1));$$
   $$(T, \mathtt{t}_2, \mathbb{d}, \sigma) \overset{\tau}{\underset{\delta_2'}{:\Longrightarrow}}{}^* \widehat{W}_2', \quad (\mathbb{T}, \mathtt{t}_2, \mathbb{d}, \Sigma) \overset{\tau}{\underset{\Delta_2'}{:\Longrightarrow}}{}^* \widehat{\mathbb{W}}_2', \quad \delta_2 \subseteq \delta_2', \quad \mathsf{FPmatch}(\mu, (\Delta_2', \Sigma), (\delta_2', F_2)).$$

   Let $\widehat{\mathbb{W}} = (\mathbb{T}, \mathtt{t}_1, \mathbb{d}, \Sigma)$. Then

   $$\mathsf{NPpred}(\widehat{\mathbb{W}}, \mathtt{t}_1, (\Delta_1', d_1)) \text{ and } \mathsf{NPpred}(\widehat{\mathbb{W}}, \mathtt{t}_2, (\Delta_2', d_2))\ .$$

   Since $\neg((\delta_1, d_1) \smile (\delta_2, d_2))$, we know

   $$\neg(\delta_1 \smile \delta_2) \text{ and } \neg(d_1 = d_2 = 1)\ .$$

   Since $\delta_1 \subseteq \delta_1'$ and $\delta_2 \subseteq \delta_2'$, we know

   $$\neg(\delta_1' \smile \delta_2')\ .$$

   Since $(\hat{P}, \sigma) \overset{load}{:\Longrightarrow} \widehat{W}$, we know

   $$F_1 \cap F_2 = \emptyset\ .$$

   Since $\mathsf{wf}(\mu, \Sigma, \sigma)$, by Lemma 70, we know

   $$\neg(\Delta_1' \smile \Delta_2')\ .$$

   Thus $\neg((\Delta_1', d_1) \smile (\Delta_2', d_2))$. Thus we have

   $$(\hat{\mathbb{P}}, \Sigma) \Longmapsto \mathsf{Race}\ .$$

2. there exist $\widehat{W}$ and n such that $(\hat{P}, \sigma) :\overset{load}{\Longrightarrow} \widehat{W}$ and $\widehat{W} :\Longrightarrow^n \mathsf{Race}$.
   Suppose $\widehat{W} = (T, \mathtt{t}, \mathbb{d}, \sigma)$. From $\hat{P} \preccurlyeq_\varphi \hat{\mathbb{P}}$, we know there exist $\mathtt{f}_1, \ldots, \mathtt{f}_n$, $\Gamma = \{(sl_1, \gamma_1), \ldots, (sl_m, \gamma_m)\}$ and $\Pi = \{(tl_1, \pi_1), \ldots, (tl_m, \pi_m)\}$, such that $\hat{\mathbb{P}} = \mathbf{let}\ \Gamma\ \mathbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m$, $\hat{P} = \mathbf{let}\ \Pi\ \mathbf{in}\ \mathtt{f}_1 \mid \ldots \mid \mathtt{f}_m$ and $\forall i \in \{1, \ldots, m\}.\ \mathsf{dom}(sl_i.\mathsf{InitCore}(\gamma_i)) = \mathsf{dom}(tl_i.\mathsf{InitCore}(\pi_i))$. Thus we know there exists $\mathbb{T}$ such that
   $$(\hat{\mathbb{P}}, \Sigma) :\overset{load}{\Longrightarrow} (\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma)\ .$$
   Let $\mathbb{S} = \mathsf{dom}(\Sigma)$, $S = \mathsf{dom}(\sigma)$, $\mu = (\mathbb{S}, S, \varphi|_{\mathbb{S}})$ and $\Delta_0 = \delta_0 = \mathsf{emp}$. We know there exists $i \in \mathtt{index}$ such that
   $$((T, \mathtt{t}, \mathbb{d}, \sigma), \delta_0) \preccurlyeq^i_\mu ((\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma), \Delta_0)\ .$$
   We only need to prove $(\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma) :\Longrightarrow^+ \mathsf{Race}$. By induction over n.
   a. $n = 1$.
      From $(T, \mathtt{t}, \mathbb{d}, \sigma) :\Longrightarrow \mathsf{Race}$, we know there exist $\widehat{W}'$, o, $\mathtt{t}_1$, $\mathtt{t}_2$, δ_1, δ_2, d_1 and d_2 such that
      $$(T, \mathtt{t}, \mathbb{d}, \sigma) :\underset{\mathsf{emp}}{\overset{o}{\Longrightarrow}} \widehat{W}'\ , \quad o = e \vee o = \mathsf{sw}\ , \quad \mathtt{t}_1 \neq \mathtt{t}_2\ ,$$
      $$\mathsf{NPpred}(\widehat{W}', \mathtt{t}_1, (\delta_1, d_1))\ , \quad \mathsf{NPpred}(\widehat{W}', \mathtt{t}_2, (\delta_2, d_2))\ , \quad \neg((\delta_1, d_1) \smile (\delta_2, d_2))$$
      Suppose $\widehat{W}' = (T', \mathtt{t}', \mathbb{d}', \sigma)$. From $(T, \mathtt{t}, \mathbb{d}, \sigma) :\underset{\mathsf{emp}}{\overset{o}{\Longrightarrow}} \widehat{W}'$, $o = e \vee o = \mathsf{sw}$ and $((T, \mathtt{t}, \mathbb{d}, \sigma), \delta_0) \preccurlyeq^i_\mu ((\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma), \Delta_0)$, we know there exist Δ'', $\widehat{\mathbb{W}}''$, μ', $\mathbb{T}'$, Σ' and j such that
      $$(\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma) :\underset{\Delta''}{\overset{\tau}{\Longrightarrow}}{}^* \widehat{\mathbb{W}}''\ , \quad \widehat{\mathbb{W}}'' :\underset{\mathsf{emp}}{\overset{o}{\Longrightarrow}} (\mathbb{T}', \mathtt{t}_1, \mathbb{d}', \Sigma')\ , \quad ((T', \mathtt{t}_1, \mathbb{d}', \sigma), \mathsf{emp}) \preccurlyeq^j_{\mu'} ((\mathbb{T}', \mathtt{t}_1, \mathbb{d}', \Sigma'), \mathsf{emp})$$
      $$\widehat{\mathbb{W}}'' :\underset{\mathsf{emp}}{\overset{o}{\Longrightarrow}} (\mathbb{T}', \mathtt{t}_2, \mathbb{d}', \Sigma')\ , \quad ((T', \mathtt{t}_2, \mathbb{d}', \sigma), \mathsf{emp}) \preccurlyeq^j_{\mu'} ((\mathbb{T}', \mathtt{t}_2, \mathbb{d}', \Sigma'), \mathsf{emp})\ .$$
      From $\mathsf{NPpred}(\widehat{W}', \mathtt{t}_1, (\delta_1, d_1))$ and $\mathsf{NPpred}(\widehat{W}', \mathtt{t}_2, (\delta_2, d_2))$, we know there exist tl_1, π_1, F_1, κ'_1, κ''_1, σ''_1, tl_2, π_2, F_2, κ'_2, κ''_2, σ''_2 such that
      $$T'(\mathtt{t}_1) = (tl_1, (\pi_1, F_1), \kappa'_1), \quad (\pi_1, F_1) \vdash (\kappa'_1, \sigma) \underset{\delta_1}{\overset{\tau}{\longmapsto}}{}^* (\kappa''_1, \sigma''_1), \quad \mathbb{d}'(\mathtt{t}_1) = d_1,$$
      $$T'(\mathtt{t}_2) = (tl_2, (\pi_2, F_2), \kappa'_2), \quad (\pi_2, F_2) \vdash (\kappa'_2, \sigma) \underset{\delta_2}{\overset{\tau}{\longmapsto}}{}^* (\kappa''_2, \sigma''_2), \quad \mathbb{d}'(\mathtt{t}_1) = d_2, \quad F_1 \cap F_2 = \emptyset\ .$$
      By the operational semantics, we know
      $$(T', \mathtt{t}_1, \mathbb{d}', \sigma) :\underset{\delta_1}{\overset{\tau}{\Longrightarrow}}{}^* (T'\{\mathtt{t}_1 \leadsto (tl_1, (\pi_1, F_1), \kappa''_1), \mathtt{t}_1, \mathbb{d}', \sigma''_1), \quad (T', \mathtt{t}_2, \mathbb{d}', \sigma) :\underset{\delta_2}{\overset{\tau}{\Longrightarrow}}{}^* (T'\{\mathtt{t}_2 \leadsto (tl_2, (\pi_2, F_2), \kappa''_2), \mathtt{t}_2, \mathbb{d}', \sigma''_2)$$
      By Lemma 69, we know there exist $\widehat{W}'_1$, δ'_1, $\widehat{\mathbb{W}}'_1$, Δ'_1 $\widehat{W}'_2$, δ'_2, $\widehat{\mathbb{W}}'_2$ and Δ'_2 such that
      $$(T', \mathtt{t}_1, \mathbb{d}', \sigma) :\underset{\delta'_1}{\overset{\tau}{\Longrightarrow}}{}^* \widehat{W}'_1, \quad (\mathbb{T}', \mathtt{t}_1, \mathbb{d}', \Sigma') :\underset{\Delta'_1}{\overset{\tau}{\Longrightarrow}}{}^* \widehat{\mathbb{W}}'_1, \quad \delta_1 \subseteq \delta'_1, \quad \mathsf{FPmatch}(\mu', (\Delta'_1, \Sigma'), (\delta'_1, F_1));$$
      $$(T', \mathtt{t}_2, \mathbb{d}', \sigma) :\underset{\delta'_2}{\overset{\tau}{\Longrightarrow}}{}^* \widehat{W}'_2, \quad (\mathbb{T}', \mathtt{t}_2, \mathbb{d}', \Sigma') :\underset{\Delta'_2}{\overset{\tau}{\Longrightarrow}}{}^* \widehat{\mathbb{W}}'_2, \quad \delta_2 \subseteq \delta'_2, \quad \mathsf{FPmatch}(\mu', (\Delta'_2, \Sigma'), (\delta'_2, F_2)).$$
      Let $\widehat{\mathbb{W}}' = (\mathbb{T}', \mathtt{t}_1, \mathbb{d}', \Sigma')$. Then
      $$\mathsf{NPpred}(\widehat{\mathbb{W}}', \mathtt{t}_1, (\Delta'_1, d_1)) \text{ and } \mathsf{NPpred}(\widehat{\mathbb{W}}', \mathtt{t}_2, (\Delta'_2, d_2))\ .$$
      Since $\neg((\delta_1, d_1) \smile (\delta_2, d_2))$, we know
      $$\neg(\delta_1 \smile \delta_2) \text{ and } \neg(d_1 = d_2 = 1)\ .$$
      Since $\delta_1 \subseteq \delta'_1$ and $\delta_2 \subseteq \delta'_2$, we know
      $$\neg(\delta'_1 \smile \delta'_2)\ .$$
      Since $F_1 \cap F_2 = \emptyset$ and $\mathsf{wf}(\mu', \Sigma', \sigma)$, by Lemma 70, we know
      $$\neg(\Delta'_1 \smile \Delta'_2)\ .$$
      Thus $\neg((\Delta'_1, d_1) \smile (\Delta'_2, d_2))$. Thus we have
      $$(\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma) \Longrightarrow^+ \mathsf{Race}\ .$$
   b. $n = m + 1$. Thus we know there exist $\widehat{W}'$, o, δ such that
      $$(T, \mathtt{t}, \mathbb{d}, \sigma) \underset{\delta}{\overset{o}{:\Rightarrow}} \widehat{W}' \text{ and } \widehat{W}' :\Longrightarrow^m \mathsf{Race}\ .$$
      Since $((T, \mathtt{t}, \mathbb{d}, \sigma), \delta_0) \preccurlyeq^i_\mu ((\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma), \Delta_0)$, we know there exists Δ, $\widehat{\mathbb{W}}'$, μ', j, Δ'_0 and δ'_0 such that
      $$(\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma) \underset{\Delta}{\overset{o}{:\Rightarrow}}{}^* \widehat{\mathbb{W}}'\ , \quad (\widehat{W}', \delta'_0) \preccurlyeq^j_{\mu'} (\widehat{\mathbb{W}}', \Delta'_0)\ .$$
      By the induction hypothesis, we know
      $$\widehat{\mathbb{W}}' :\Longrightarrow^+ \mathsf{Race}\ .$$
      Thus we have
      $$(\mathbb{T}, \mathtt{t}, \mathbb{d}, \Sigma) \Longrightarrow^+ \mathsf{Race}\ .$$

Thus we are done.

Lemma 69. If $(\widehat{W}, \delta_0) \leqslant_{\mu}^{i} (\widehat{\mathbb{W}}, \Delta_0)$, $\mathsf{Safe}(\widehat{W})$ and $\widehat{W} \overset{\tau}{\underset{\delta_1}{:\Rightarrow}}{}^{n} \widehat{W}_1$, then there exist $\widehat{W}'$, δ, $\widehat{\mathbb{W}}'$ and Δ such that

1. $\widehat{W} \overset{\tau}{\underset{\delta}{:\Rightarrow}}{}^{*} \widehat{W}'$, and $\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\delta_1 \subseteq \delta$, and
2. $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$.

Proof. By induction over n.

1. $n = 0$. Thus $\delta_1 = \mathsf{emp}$. By induction over i (using the well-founded order).
   a. $\neg\exists j. j < i$.
   Since $\neg(\widehat{W} \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}} \textbf{abort})$, we know there exists $\widehat{W}''$ and δ'' such that
   $$\widehat{W} \overset{\tau}{\underset{\delta''}{:\Longrightarrow}} \widehat{W}'', \text{ or } \exists o \neq \tau.\, \widehat{W} \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} \widehat{W}'', \text{ or } \widehat{W} \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}} \textbf{done}$$
   i. If $\widehat{W} \overset{\tau}{\underset{\delta''}{:\Longrightarrow}} \widehat{W}''$, we let $\widehat{W}' = \widehat{W}''$ and $\delta = \delta''$. From $(\widehat{W}, \delta_0) \leqslant_{\mu}^{i} (\widehat{\mathbb{W}}, \Delta_0)$ and $\neg\exists j. j < i$, we know there exist $\widehat{\mathbb{W}}'$ and Δ such that
   $$\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}', \qquad \text{and} \qquad \mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F)).$$
   ii. If $\exists o \neq \tau.\, \widehat{W} \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} \widehat{W}''$, we let $\widehat{W}' = \widehat{W}$. From $(\widehat{W}, \delta_0) \leqslant_{\mu}^{i} (\widehat{\mathbb{W}}, \Delta_0)$, we know there exist $\widehat{\mathbb{W}}'$ and Δ such that
   $$\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}', \qquad \text{and} \qquad \mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F)) \ .$$
   iii. If $\widehat{W} \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}} \textbf{done}$, we let $\widehat{W}' = \widehat{W}$. From $(\widehat{W}, \delta_0) \leqslant_{\mu}^{i} (\widehat{\mathbb{W}}, \Delta_0)$, we know there exist $\widehat{\mathbb{W}}'$ and Δ such that
   $$\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}', \qquad \text{and} \qquad \mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F)) \ .$$
   b. $\exists j. j < i$.
   Since $\neg(\widehat{W} \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}} \textbf{abort})$, we know there exists $\widehat{W}''$ and δ'' such that
   $$\widehat{W} \overset{\tau}{\underset{\delta''}{:\Longrightarrow}} \widehat{W}'', \text{ or } \exists o \neq \tau.\, \widehat{W} \overset{o}{\underset{\mathsf{emp}}{:\Longrightarrow}} \widehat{W}'', \text{ or } \widehat{W} \overset{\tau}{\underset{\mathsf{emp}}{:\Longrightarrow}} \textbf{done}$$
   We consider only the case of $\widehat{W} \overset{\tau}{\underset{\delta''}{:\Longrightarrow}} \widehat{W}''$ (the other two cases are similar to the above proofs). From $(\widehat{W}, \delta_0) \leqslant_{\mu}^{i} (\widehat{\mathbb{W}}, \Delta_0)$, we know one of the following holds:
   i. there exist $\widehat{\mathbb{W}}''$, Δ'' and j such that
   $$\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta''}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}'', \quad \text{and} \quad j < i, \quad \text{and} \quad (\widehat{W}'', \delta_0 \cup \delta'') \leqslant_{\mu}^{j} (\widehat{\mathbb{W}}'', \Delta_0 \cup \Delta'') \ .$$
   By the induction hypothesis, we know there exist $\widehat{W}'$, δ, $\widehat{\mathbb{W}}'$ and Δ such that
   $\widehat{W}'' \overset{\tau}{\underset{\delta}{:\Rightarrow}}{}^{*} \widehat{W}'$ and $\widehat{\mathbb{W}}'' \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta'' \cup \Delta, \widehat{\mathbb{W}}''.\Sigma), (\delta_0 \cup \delta'' \cup \delta, ((\widehat{W}''.T)(\widehat{W}''.\mathtt{t})).F))$.
   By the operational semantics, we know $((\widehat{W}''.T)(\widehat{W}''.\mathtt{t})).F = ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F$. By Lemma 57, we know
   $\widehat{W} \overset{\tau}{\underset{\delta'' \cup \delta}{:\Longrightarrow}}{}^{*} \widehat{W}'$ and $\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta'' \cup \Delta}{:\Longrightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta'' \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta'' \cup \delta, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$.
   ii. there exist $\widehat{\mathbb{W}}'$ and Δ such that
   $$\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}', \quad \text{and} \quad \mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta'', ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F)) \ .$$
2. $n = m + 1$. Thus there exist $\widehat{W}_2$, δ_2 and δ_2' such that
   $$\widehat{W} \overset{\tau}{\underset{\delta_2}{:\Rightarrow}} \widehat{W}_2, \quad \widehat{W}_2 \overset{\tau}{\underset{\delta_2'}{:\Rightarrow}}{}^{m} \widehat{W}_1, \quad \delta_1 = \delta_2 \cup \delta_2'$$
   From $(\widehat{W}, \delta_0) \leqslant_{\mu}^{i} (\widehat{\mathbb{W}}, \Delta_0)$, we know one of the following holds:
   a. there exist $\widehat{\mathbb{W}}_2$, Δ_2 and j such that
   $$\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta_2}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}_2, \quad \text{and} \quad j < i, \quad \text{and} \quad (\widehat{W}_2, \delta_0 \cup \delta_2) \leqslant_{\mu}^{j} (\widehat{\mathbb{W}}_2, \Delta_0 \cup \Delta_2) \ .$$
   By the induction hypothesis, we know there exist $\widehat{W}'$, δ, $\widehat{\mathbb{W}}'$ and Δ such that
   $\widehat{W}_2 \overset{\tau}{\underset{\delta}{:\Rightarrow}}{}^{*} \widehat{W}'$ and $\widehat{\mathbb{W}}_2 \overset{\tau}{\underset{\Delta}{:\Rightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\delta_2' \subseteq \delta$, and $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta_2 \cup \Delta, \widehat{\mathbb{W}}_2.\Sigma), (\delta_0 \cup \delta_2 \cup \delta, ((\widehat{W}_2.T)(\widehat{W}_2.\mathtt{t})).F))$.
   By the operational semantics, we know $((\widehat{W}_2.T)(\widehat{W}_2.\mathtt{t})).F = ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F$. By Lemma 57, we know
   $\widehat{W} \overset{\tau}{\underset{\delta_2 \cup \delta}{:\Longrightarrow}}{}^{*} \widehat{W}'$ and $\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta_2 \cup \Delta}{:\Longrightarrow}}{}^{*} \widehat{\mathbb{W}}'$, and $\delta_1 \subseteq \delta_2 \cup \delta$, and $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta_2 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta_2 \cup \delta, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$.

b. there exist $\widehat{\mathbb{W}}_2$ and Δ_2 such that
$\widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta_2}{:\Longrightarrow}}{}^* \widehat{\mathbb{W}}_2$, and $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta_2, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta_2, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$ and $(\widehat{W}_2, \mathtt{emp}) \preccurlyeq^j_\mu (\widehat{\mathbb{W}}_2, \mathtt{emp})$.

By the induction hypothesis, we know there exist $\widehat{W}'$, δ, $\widehat{\mathbb{W}}'$ and Δ such that
$$\widehat{W}_2 \overset{\tau}{\underset{\delta}{:\Longrightarrow}}{}^* \widehat{W}' \text{ and } \widehat{\mathbb{W}}_2 \overset{\tau}{\underset{\Delta}{:\Longrightarrow}}{}^* \widehat{\mathbb{W}}', \text{ and } \delta_2' \subseteq \delta, \text{ and } \mathsf{FPmatch}(\mu, (\Delta, \widehat{\mathbb{W}}_2.\Sigma), (\delta, ((\widehat{W}_2.T)(\widehat{W}_2.\mathtt{t})).F)).$$

By the operational semantics, we know $((\widehat{W}_2.T)(\widehat{W}_2.\mathtt{t})).F = ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F$. By Lemma 57, we know
$$\widehat{W} \overset{\tau}{\underset{\delta_2 \cup \delta}{:\Longrightarrow}}{}^* \widehat{W}' \text{ and } \widehat{\mathbb{W}} \overset{\tau}{\underset{\Delta_2 \cup \Delta}{:\Longrightarrow}}{}^* \widehat{\mathbb{W}}', \text{ and } \delta_1 \subseteq \delta_2 \cup \delta, \text{ and } \mathsf{FPmatch}(\mu, (\Delta, \widehat{\mathbb{W}}.\Sigma), (\delta, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F)).$$

From $\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta_2, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta_2, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F))$, we know
$$\mathsf{FPmatch}(\mu, (\Delta_0 \cup \Delta_2 \cup \Delta, \widehat{\mathbb{W}}.\Sigma), (\delta_0 \cup \delta_2 \cup \delta, ((\widehat{W}.T)(\widehat{W}.\mathtt{t})).F)).$$

Thus we are done. □

Lemma 70. If $\Delta_1 \smile \Delta_2$, $\mathsf{FPmatch}(\mu, (\Delta_1, \Sigma), (\delta_1, F_1))$, $\mathsf{FPmatch}(\mu, (\Delta_2, \Sigma), (\delta_2, F_2))$, $\mathsf{injective}(\mu.f, \Sigma)$ and $F_1 \cap F_2 = \emptyset$, then $\delta_1 \smile \delta_2$.

Proof. From $\mathsf{FPmatch}(\mu, (\Delta_1, \Sigma), (\delta_1, F_1))$ and $\mathsf{FPmatch}(\mu, (\Delta_2, \Sigma), (\delta_2, F_2))$, we know

$$\begin{gathered}
\delta_1.rs \cup \delta_1.ws \subseteq \llbracket F_1 \rrbracket \cup \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle \ , \\
\delta_1.rs \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle \subseteq \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_1.rs \rangle\!\rangle \ , \quad \delta_1.ws \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle \subseteq \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_1.ws \rangle\!\rangle \ ; \\
\delta_2.rs \cup \delta_2.ws \subseteq \llbracket F_2 \rrbracket \cup \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle \ , \\
\delta_2.rs \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle \subseteq \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_2.rs \rangle\!\rangle \ , \quad \delta_2.ws \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle \subseteq \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_2.ws \rangle\!\rangle \ .
\end{gathered}$$

Thus we know

$$\begin{aligned}
\delta_1.rs &\subseteq (\llbracket F_1 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cup (\delta_1.rs \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \\
&\subseteq (\llbracket F_1 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cup (\mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_1.rs \rangle\!\rangle \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \ , \\
\delta_1.ws &\subseteq (\llbracket F_1 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cup (\delta_1.ws \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \\
&\subseteq (\llbracket F_1 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cup (\mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_1.ws \rangle\!\rangle \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \ , \\
\delta_2.rs &\subseteq (\llbracket F_2 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cup (\delta_2.rs \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \\
&\subseteq (\llbracket F_2 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cup (\mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_2.rs \rangle\!\rangle \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \ , \\
\delta_2.ws &\subseteq (\llbracket F_2 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cup (\delta_2.ws \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \\
&\subseteq (\llbracket F_2 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cup (\mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_2.ws \rangle\!\rangle \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \ .
\end{aligned}$$

We have

$$\begin{gathered}
(\llbracket F_1 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cap (\mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_2.rs \rangle\!\rangle \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) = \emptyset \ , \\
(\llbracket F_2 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cap (\mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_1.rs \rangle\!\rangle \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) = \emptyset \ , \\
(\llbracket F_1 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cap (\mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_2.ws \rangle\!\rangle \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) = \emptyset \ , \\
(\llbracket F_2 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cap (\mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_1.ws \rangle\!\rangle \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) = \emptyset \ .
\end{gathered}$$

Since $F_1 \cap F_2 = \emptyset$, we know

$$(\llbracket F_1 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) \cap (\llbracket F_2 \rrbracket - \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \llbracket \mu.\mathbb{S} \rrbracket \rangle\!\rangle) = \emptyset \ .$$

Since $\Delta_1 \smile \Delta_2$, we know

$$\Delta_1.ws \cap (\Delta_2.rs \cup \Delta_2.ws) = \emptyset \quad \text{and} \quad \Delta_1.rs \cap \Delta_2.ws = \emptyset \ .$$

Since $\mathsf{injective}(\mu.f, \Sigma)$, we know

$$\begin{gathered}
\mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_1.ws \rangle\!\rangle \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_2.rs \rangle\!\rangle = \emptyset \ , \\
\mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_1.ws \rangle\!\rangle \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_2.ws \rangle\!\rangle = \emptyset \ , \\
\mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_1.rs \rangle\!\rangle \cap \mu.f\langle\!\langle \mathsf{locs}(\Sigma) \cap \Delta_2.ws \rangle\!\rangle = \emptyset \ .
\end{gathered}$$

Thus we know

$$\delta_1.ws \cap (\delta_2.rs \cup \delta_2.ws) = \emptyset \quad \text{and} \quad \delta_1.rs \cap \delta_2.ws = \emptyset \ .$$

Thus $\delta_1 \smile \delta_2$. □