Abstract

Memory Consistency and Program Verification

Rodrigo Ferreira

2010

Formal reasoning about concurrent programs is usually done with the assumption that the underlying memory model is sequentially consistent, i.e. the execution outcome is equivalent to an interleaving of instructions according to the program order. However, memory models in reality are weaker in order to accommodate compiler and hardware optimizations. To simplify the reasoning, many memory models provide a guarantee that data-race-free programs behave in a sequentially consistent manner, the so-called DRF-guarantee. The DRF-guarantee removes the burden of reasoning about relaxations when the program is well-synchronized. It is common belief that the current tools for program verification, such as Concurrent Separation Logic (CSL), yield DRF-programs. In principle, they can rely on the DRF-guarantee to ignore memory model issues. However, there is no rigorous evidence for that, given the fact that the work on memory models is not founded with a level of formalism adequate for program verification. It is a semantic gap between the two fields. This thesis provides seminal work towards bridging them together.

In our presentation, we formalize memory consistency models by giving a parameterized operational semantics to a concurrent programming language. Behaviors of a program under a relaxed memory model are defined as behaviors of a set of *related* programs under the *sequentially consistent* model. This semantics is parameterized in the sense that different memory models can be obtained by using different relations between programs. We present one particular relation, called *command subsumption*, that we believe accounts for many memory models and sequential optimizations. We then show that the derived relaxed memory model provides the DRF-guarantee, using, as intermediate, an auxiliary

mixed-step semantics that is compatible with the subsumption relation. Finally, we bridge the soundness of CSL, and variations, to our relaxed semantics. We establish the soundness following a standard semantic approach. This shows that, indeed, programs verified with CSL are race-free and their execution in the relaxed memory model exhibits the same set of behaviors as in the interleaved semantics.

Memory Consistency and Program Verification

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Rodrigo Ferreira

Dissertation Director: Zhong Shao

April 2010

Copyright © 2010 by Rodrigo Ferreira

All rights reserved.

Contents

List of Figures

Chapter 1

Introduction

For many years, optimizations of sequential code — by both compiler and architecture — have been the major source of performance improvement of computing systems. Compiler transformations, superscalar pipelines, and memory caches are some of the artifacts used to achieve that. However, these optimizations were designed to preserve the sequential semantics of the code. When placed in a concurrent context, many of them violate the interleaved semantics [51, 3] commonly assumed when reasoning about shared-memory programs.

One example of that is Dekker's mutual exclusion algorithm [24] as seen below:

Initially $x \neq y$ and $[x] = [y] = 0$

$$\begin{array}{lcl} {[x]} := 1; & & {[y]} := 1; \\ v_1 := [y]; & \parallel & v_2 := [x]; \\ \textbf{if } v_1 = 0 \textbf{ then } \textit{critical section} & & \textbf{if } v_2 = 0 \textbf{ then } \textit{critical section} \end{array}$$

for which the proof of correctness (see for instance Yu and Shao [79]) is invalidated if outcome $v_1 = v_2 = 0$ is allowed. Surprisingly, this behavior might happen if, for instance, the compiler decides to reorders the assignment statements on both sides; leading to an execution where both threads can enter the critical sections. Many other synchronization algorithms are susceptible to failure in a similar fashion, this is a well-known problem

[13, 3].

The semantics of multithreaded languages must rely on a *memory consistency model* to rigorously define how threads interact through a shared memory system. The memory model serves as a contract between the programmer and the compiler/architecture designer, and is the reflex of a trade-off between programmability and performance. Its thorough understanding is essential to correctly program a concurrent system.

The intuitive interleaving semantics, many times assumed by people not familiar with memory consistency issues, is known as *sequential consistency* (SC). It was defined originally by Lamport [51] and is, unfortunately, too costly to be adopted as it disables many optimizations widely used for sequential systems [3]. Memory models then often relax the read/write ordering and visibility among threads to create room for optimizations, hence they are referred to as *relaxed* or *weak*. As a side effect, they increase the reasoning complexity required to understand program behavior, usually to a level that cannot be managed beyond the scope of small code fragments.

To simplify the reasoning, memory models often provide a guarantee that DRF-programs behave according to the interleaved semantics; the so-called *DRF-guarantee*. A *DRF-program* is a program absent of data-races for all interleaved executions; where a *data-race* is defined as two simultaneous concurrent accesses to the same variable, at least one of them modifying the variable, and at least one of them being unsynchronized.

In order to write a DRF-program, the programmer must enforce that shared-memory communication is always performed using the specific synchronization primitives provided by the language/architecture. At the high-level, these primitives usually guarantee some form of mutual exclusion, e.g. mutexes. At the low-level — such as in our Dekker's example where we are implementing mutual exclusion — these primitives will come in the form of strong operations/barriers that enforce ordering and atomicity of memory accesses, e.g. compare-and-swap (CAS) instruction.

In our presentation, throughout this thesis, we provide just one primitive for shared memory access, the atomic block (**atomic** c). By wrapping a command c inside an atomic block the programmer has the guarantee that it will execute atomically. Furthermore, an atomic block also acts as a barrier, enforcing that all operations executed prior to it must be finished before it starts, and all operations following it must wait for it to complete its execution before they can start.

Back to our example, in order to implement Dekker's algorithm, the programmer must wrap *conflicting accesses*, i.e. accesses that may involve a race, inside an atomic block, as can be seen below:

Initially $x \neq y$ and $[x] = [y] = 0$		
atomic $[x] := 1$;		**atomic** $[y] := 1$;
atomic $v_1 := [y]$;	$\|$	**atomic** $v_2 := [x]$;
if $v_1 = 0$ **then** *critical section*		**if** $v_2 = 0$ **then** *critical section*

By doing this, the program becomes DRF, which means that its execution in a relaxed memory model appears to be sequential consistent, if, of course, we assume that the given model provides the DRF-guarantee.

In fact, we use atomic blocks because they can generalize many concrete synchronization primitives at both high- and low-levels, given that c can be arbitrary. For instance, a non-reentrant lock operation on a mutex could be implemented as

$$\textbf{atomic}\ (\textbf{while}\ [l] \neq 0\ \textbf{do skip};\ [l] := 1)$$

and a low-level CAS instruction can be implemented as

$$\textbf{atomic}\ (\textbf{if}\ [l] = v_1\ \textbf{then}\ ([l] := v_2; r := 1)\ \textbf{else}\ r := 0)$$

1.1 Memory Models and Language Semantics

Until recently, the problem of understanding and describing the semantics supported by shared-memory programs has been addressed mostly by the computer architecture community. It has been ignored by the programming language community. However, a lot of interest was brought to this matter due to the conception of the new Java Memory Model (JMM) [54].

The JMM brought to attention the fact that — besides the DRF-guarantee — high-level memory models must preserve the soundness of type safe languages in the *presence of races*. Since Java's type system is not powerful enough to rule out data-races, arbitrary side-effects from them could violate the type safety of the language and be exploited to bypass the sand-boxing provided by the runtime system. The majority of the time and effort taken to define the JMM was spent drawing the line between optimization and safety. Java also has a universal portability requirement that greatly complicates its memory model given that it must maintain the performance compromise for many different architectures, including obsolete ones.

Some consensus was achieved and the JMM was officially released [47], but the topic is still controversial [11, 21]. Nevertheless, it is undoubtful that important advances were made in the field. Based on the experiences learned from the JMM, the C++ memory model [12] was developed. However, it is much simpler. The C++ language is not type-safe, therefore, it is not unreasonable to assign arbitrary behaviors for raceful programs.

In our setting, we are interested in studying the connection between *program verification* and relaxed memory models. Therefore, we approach the problem from a different perspective than the JMM. The behavior of raceful programs do not matter from the point of view of program verification. This derives from the fact that verification already guarantees the DRF property, or, in other words, once a program has been verified it is consequently race-free. This holds for Concurrent Separation Logic (CSL) [58, 18] which is the most promising tool for concurrent program verification at the moment. It also holds for many of its variations [14, 30, 71]. Therefore, we can take race-freedom for granted[1].

Our concern is a different one. We have noticed that there is a *semantic gap* between concurrent program verification and the semantics of relaxed memory models. The soundness of CSL has been established in many different ways [18, 17, 30, 71, 42]. However, invariably, it was established with regard to a dynamic semantics that was either explicitly sequentially consistent, or grainless, i.e. too abstract to reflect the effects of any relaxed memory model. This means that — although there is a common belief that CSL-based verification extends to relaxed memory models — there is no rigorous evidence of that. On the other hand, the majority of the literature on memory models describe them informally. In the best case, they are described axiomatically as is the case for the JMM. These descriptions impose constraints to the set of memory events that might happen during execution, but the connection with language semantics is vague. They are not well-suited for applying the standard formal reasoning techniques.

Formal reasoning is typically done with regard to an operational semantics. But, unfortunately, the operational approaches to relaxed memory models described in the literature [15, 16, 46] are very preliminary. We believe the inexistence of a proper *operational semantics* is the main obstacle to close the semantic gap.

1.2 Contributions

In this thesis, we formalize memory consistency models by giving a parameterized operational semantics to a concurrent programming language. Behaviors of a program under a relaxed memory model are defined as behaviors of a set of *related* programs under the *sequentially consistent* model. That can be observed in the sample prototype rule depicted below

$$\frac{(c_1, c_1'') \in \Lambda \quad \langle c_1'', \sigma \rangle \longmapsto \langle c_1', \sigma' \rangle}{[\Lambda]\ \langle c_1 \parallel c_2, \sigma \rangle \longmapsto \langle c_1' \parallel c_2, \sigma' \rangle}$$

[1]In fact, we establish this DRF property formally as part of CSL's soundness in Chapter 6.

where before performing a sequential step, the code c_1 is replaced by c_1'' though the relation Λ. This semantics is parameterized in the sense that different memory models can be obtained by using different relations between programs. The relation Λ is a parameter that can be seen as a program transformer, such as a compiler or superscalar pipeline, reordering/rewriting the code.

The parameterized semantics becomes more interesting once we instantiate Λ with one particular relation, called *command subsumption*. Command subsumption is a relation between a program and its transformed version such that synchronization operations and the sequential semantics of non-synchronized code segments are preserved. We believe it accounts for many memory models and sequential optimizations.

Once we have established this relaxed semantics, by instantiating the parameterized semantics with the command subsumption, we show that it has the DRF-guarantee. In order to construct the proof, we use, as intermediate, an auxiliary mixed-step semantics that is compatible with the subsumption relation.

Finally, we bridge the soundness of concurrent separation logic (CSL), and variations, to our relaxed semantics. We establish the soundness following a standard semantic approach. This shows that programs verified with CSL are race-free and their execution in the relaxed memory model exhibits the same set of behaviors as in the interleaved semantics. We highlight our main contributions:

1. We define an interesting interleaved contextual semantics for concurrency (Sec. 3.7), in which atomic blocks have weak atomicity semantics (i.e. atomic blocks are not executed in a single shot an might interleave with unprotected, sequential, code). Also, our semantics does not impose restrictions over the nesting of atomic blocks and parallel compositions. Moreover, it is furnished with explicit checks for race conditions based on sequential footprints, which provides a natural way to define data-race freedom;

2. We introduce a novel, simple and general, structural operational semantics for re-

laxed memory models (Sec. 3.8). It captures the elementary view that memory models are the result of program transformations by compiler or hardware. It is parameterized in order to capture different memory models;

3. We define one particular instantiation of the semantics, based on the idea of command subsumption (Sec. 3.9). Command subsumption is an asymmetric equivalence relation between two commands that captures many sequential optimizations. It is defined semantically for nonsynchronized portions of the code and it preserves the relative displacement of synchronization code. We believe that command subsumption should also serve as basis for checking the correctness of sequential compiler transformations;

4. We prove the DRF-guarantee of our relaxed semantics (Chapter 5), showing that all program transformations performed according to the subsumption relation preserve the semantics or DRF-programs. This is performed using an intermediate mixed-step semantics that is compatible with subsumption and makes the proof a lot simpler;

5. We work out a version of CSL for our concurrent language (Chapter 6), and we prove its soundness with regard to the relaxed semantics using a standard semantic approach. Since the dynamic semantics has weak atomicity characteristics, the proof is unique when compared to the literature. Similar proofs were developed for two extensions CSL with partial permissions (Chapter 7); and SAGL, the Separated Assume-Guarantee Logic (Chapter 8), which is an adaptation of our original work [30] combining assume-guarantee reasoning and separation logic;

6. We formalize most of the technical content of this thesis inside the Coq proof assistant (Appendix B);

In a nutshell, we provide seminal work bridging together two related but nearly unconnected fields: memory consistency models and concurrent program verification.

1.3 Thesis Outline

In Chapter 2, we present the big picture of this work, intuitively, as a informal prelude to the subsequent, more technical, content of the thesis. In Chapter 3, we introduce the syntax and semantics of our concurrent programming language, which includes the relaxed semantics. In Chapter 4, we provide a set of examples to illustrate and connect our relaxed semantics to concrete features of memory models. In Chapter 5, we prove the DRF-guarantee of our relaxed semantics. In Chapter 6, we present a version of CSL for our concurrent language and we prove its soundness with regard to the relaxed semantics. In Chapter 7, we extend CSL to handle shared read-only accesses by incorporating partial permissions to the logic. In Chapter 8, we present a version of SAGL for our concurrent language, allowing shared memory to be accessed following the rely-guarantee methodology. In Chapter 9, we discuss two important extensions of our work: cross-atomic rewriting and partial barriers which are part of future work. Finally, in Chapter 10, we present some of the related work and our conclusion. Appendix A contains all the notations used through out this document. Appendix B provides a guide to the structure of the Coq proofs.

Chapter 2

Preliminaries

In this chapter, we provide an informal and intuitive introduction to the technical content to be presented in the subsequent chapters.

2.1 Overview

This thesis provides seminal work bridging together two related, but nearly unconnected, fields: *memory consistency* and *program verification*. Most of the work on concurrent program verification neglects the fact that shared-memory multiprocessors have relaxed memory consistency models; they typically adopt an interleaving semantics to parallelism. On the other hand, the work of memory consistency models is not founded with a level of formalism adequate for formal reasoning. This is a semantic gap between the two fields in which we base our work.

Relaxed memory models (RMMs) exist because we need to define, formally, the behavior of concurrent programs in the presence of sequential optimizations. These optimizations, if applied naively, produce undesirable side-effects in the execution. One such examples is the Dekker's algorithm implementation presented in Chapter 1, where a trivial sequential transformation violates mutual exclusion. Furthermore, optimizations may induce an execution that does not behave according to sequential consistency (SC), i.e. the result cannot be obtained through the sequential interleaving of operations. Such behaviors are not intuitive.

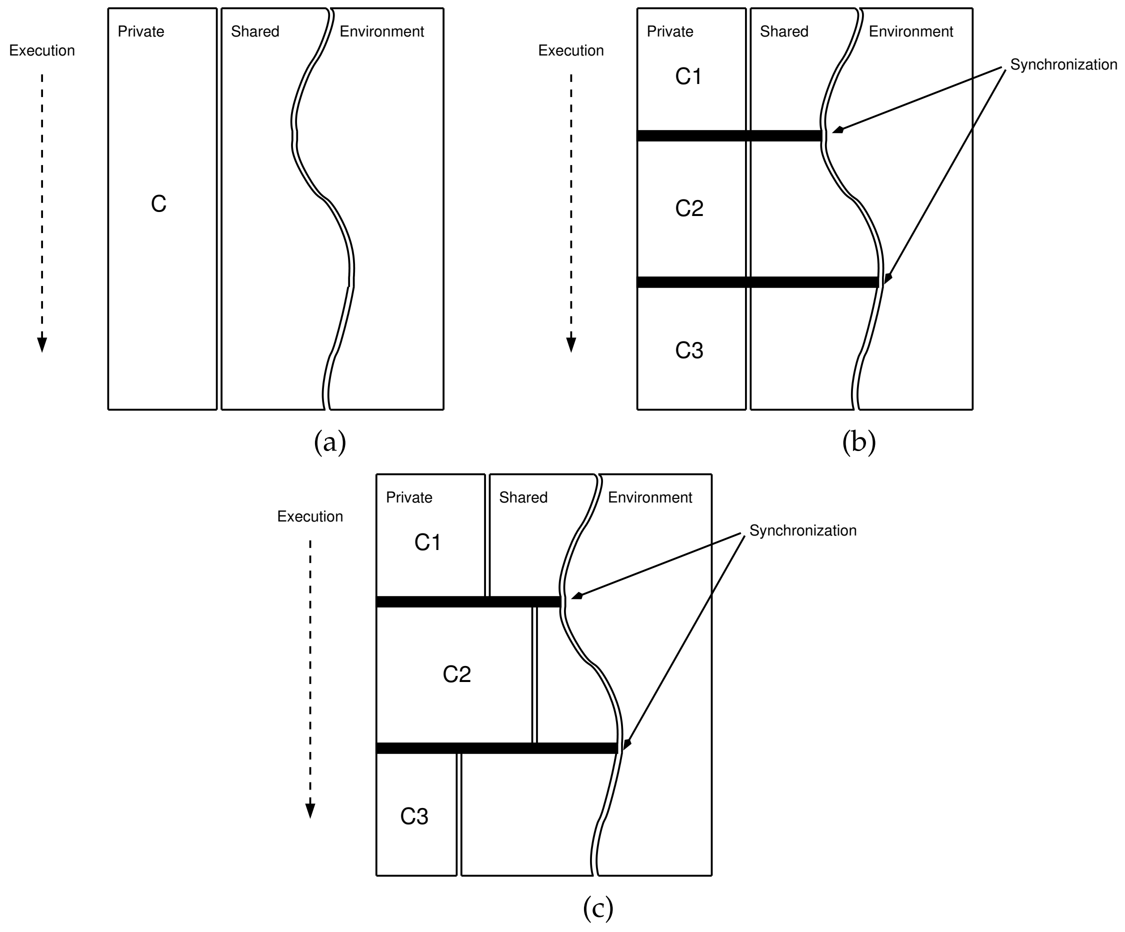

Figure 2.1: DRF-programs: (a) data accesses to private memory; (b) shared-memory accesses using synchronization; (c) memory ownership transfer during synchronization

What we learned in the past, from the literature on memory models, is that we need to divide operations into two kinds: data operations and synchronization operations. The optimizer must be aware of this segregation, and take proper cake when the optimization involves synchronization. Moreover, in order to abstract the effects of the optimizer, one must observe the data-race-free (DRF) guarantee of the RMM. The DRF-guarantee ensures that DRF-programs behave under the RMM just like they would under SC. Where a DRF-program is a program absent of data-races when executed according to SC. And a data-race is defined as two simultaneous accesses to the same memory location, at least one of them being a write, and either one of them is not synchronized.

As shown in Fig. 2.1, in a DRF program each thread must necessarily be either executing data operations in its own private memory or it must be performing a shared-memory operation using a synchronization. The C's in the figure represent purely sequential portions of the code; while the black ribbons represent synchronization. During synchronization, a thread may transfer some memory to or from its own private heap, such as when a thread dequeues or enqueues a node in a concurrent queue. Note that the separation between the private memories and shared memory depicted here are merely logical and not enforced by the hardware.

From the point of view of program verification, we are interested in Concurrent Separation Logic (CSL) [58] and its variations. CSL is the most popular and promising logic for concurrency verification nowadays. Its soundness has been established with regard to a sequential consistent semantics in various different ways. Furthermore, it is common belief that programs verified using CSL are DRF.

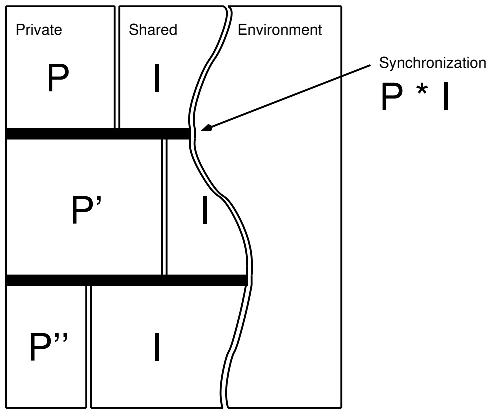

Figure 2.2: CSL verified programs

The intuition behind CSL is shown in Fig. 2.2. In CSL, each thread has its own private heap which is described by an assertion P. The shared memory has an invariant I. Separation between private heaps and shared memory is enforced using the separating conjunction from separation logic. Communication is performed at synchronization points where the thread temporarily gains exclusive access to the shared memory, performs some operation, and releases the shared memory after restoring its invariant. Even having to restore the invariant, ownership transfer might happen during synchronization. This is because the same invariant might describe different shared memories allowing them to have different sizes (e.g. if the invariant is a linked-list of arbitrary size, it will hold for a given linked list before and after a node is inserted). Note that it is not hard to see that following CSL, race freedom is an intuitive property.

The main importance of this work is the formal connection between relaxed memory models and program verification. We provide a soundness proof of CSL with regard to a relaxed semantics, which is done in the following manner:

1. We define a *parameterized* semantics, a simple semantics to derive different RMMs;
2. We define *subsumption*, a relation between a program and its transformed version such that synchronization operations and the sequential semantics of non-synchronized code segments are preserved;
3. We combine the parameterized semantics with subsumption in order to obtain our *relaxed* semantics, which we use as target for the soundness proof of CSL;

Naturally, the soundness proof relies on the DRF property of CSL-verified programs and the DRF-guarantee of our relaxed semantics.

2.2 Parameterized Semantics

The idea behind our parameterized semantics is very simple. We incorporate an optimizer into the operational semantics. At every step, we allow transformations to occur. That can be seen in Fig. 2.3.

In fact, our parameterized semantics considers the SC execution of all programs related through Λ. Naturally, we obtain a given memory model by instantiating Λ with its optimizer. This semantics if very general and can support virtually any transformation.

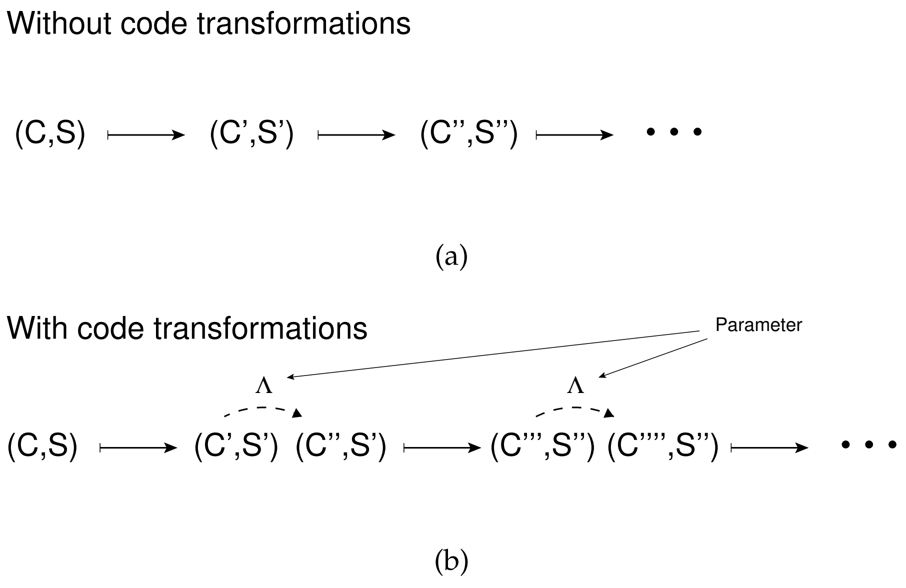

Figure 2.3: Operational semantics: (a) traditional; (b) parameterized

2.3 Command Subsumption

The intuition behind command subsumption is that we can freely optimize sequential code, because, for DRF programs, they will be accessing private memory and, therefore, the intermediate states of computation do not affect the execution of other threads. It must, though, maintain the surrounding synchronization operations. This idea is depicted in Fig. 2.4, and it is far more general than describing memory ordering of operations, specific hardware features, or compiler optimizations. Also, subsumption captures program equivalence semantically. We only look at the program states in the beginning and end of the sequential blocks, we do not look at the actual code that was executed.

Command subsumption is defined with 4 requirements in mind (depicted in Fig. 2.5):

1. It must preserve sequential safety: the target program only crashes if the source program also crashes;

2. It must preserve sequential behavior: the target program executes starting from an initial state and reaches a synchronization, or the end of the program, at a given final state, only if the source program, executing from the same initial state, reaches the

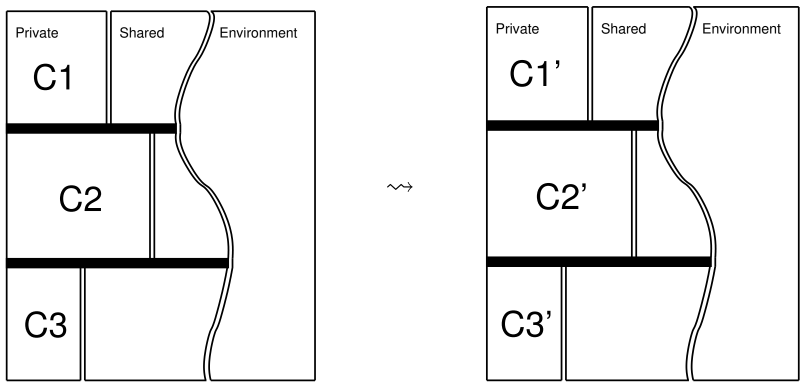

Figure 2.4: Command subsumption: it preserves the synchronization operations but allow great freedom in the rewriting of sequential code

same synchronization, at the same final state (and subsequent code must be related through subsumption);

3. It must preserve sequential termination: the target program only diverges if the source program also diverges;

4. It must preserve concurrent non-interference: the target program only executes with a given footprint if the source program executes in a footprint that is not smaller. A footprint is basically the set of memory locations that a program accesses;

2.4 Relaxed Semantics

By instantiating the parameterized semantics with the command subsumption relation, we obtain a relaxed semantics. This relaxed semantics accounts for all the sequential optimizations supported by subsumption. As can be observed from the figures presented earlier in this chapter, subsumption preserves sequential semantics of unsynchronized portions of the code, while maintaining the synchronization points. Furthermore, it does that without creating environmental interference. We hope that, by now, the reader get the intuition that: (a) if a program is verified with CSL it is DRF; and, (b) if a program is

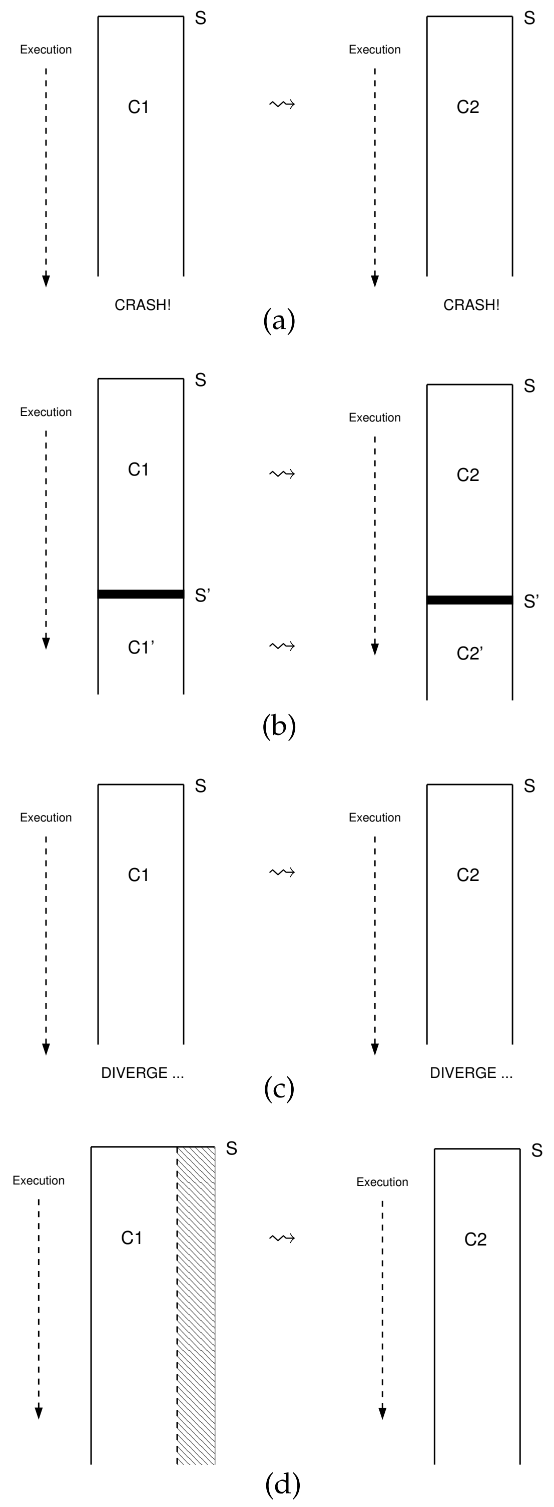

Figure 2.5: Subsumption requirements: (a) safety preservation; (b) sequential equivalence; (c) termination preservation; (d) footprint non-increase

DRF, the relaxed semantics will only produce behaviors that are achievable by a sequential interleaving.

The formal definitions of subsumption and the relaxed semantics are presented in Chapter 3. Examples of subsumption and the relaxed semantics are presented in Chapter 4. The proof of the DRF-guarantee for the relaxed semantics is presented in Chapter 5. The soundness of CSL, which includes the DRF-property of CSL-verified programs, is presented in Chapter 6.

Chapter 3

Language Syntax and Semantics

In this chapter, we present the programming language used through out this thesis. It is an imperative language composed of typical features like memory assignment, allocation and deallocation, and control structures. The language has also concurrency primitives. Here we present the language syntax and its structural operational semantics. The semantics is presented in four distinct parts:

1. A footprint semantics for the sequential core of the language;
2. A concurrent semantics for interleaved execution;
3. A parameterized semantics that can be instantiated to obtain a given memory model;
4. A relaxed semantics as an instantiation of the parameterized semantics based on the notion of program subsumption;

3.1 Syntax

The core syntax of the language is presented in Fig. 3.1.

A command c can be either and action (a), an empty command (**skip**), a control flow command (**if** b **then** c_1 **else** c_2 or **while** b **do** c), or a concurrency command ($c_1 \parallel c_2$ or **atomic** c). Actions modify either program variables or memory locations. The empty

$$
\begin{array}{rrcl}
(\textit{Command}) & c & ::= & a \mid c_1;c_2 \mid \textbf{skip} \\
 & & & \mid \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \mid \textbf{while } b \textbf{ do } c \\
 & & & \mid c_1 \,\|\, c_2 \mid \textbf{atomic } c \\
(\textit{Action}) & a & ::= & \nu := e \mid \nu := \textbf{cons}(e_1, \ldots, e_n) \mid \textbf{dispose}(e) \\
(\textit{Lvalue}) & \nu & ::= & v \mid [e] \\
(\textit{Expression}) & e & ::= & i \mid \nu \mid e_1 + e_2 \mid -e \\
(\textit{Condition}) & b & ::= & z \mid b_1 \wedge b_2 \mid \neg b \mid e_1 = e_2 \mid e_1 < e_2 \\
(\textit{Variable}) & v & \in & \{\mathtt{a}, \ldots, \mathtt{z}, \mathtt{A}, \ldots, \mathtt{Z}, \mathtt{0}, \ldots, \mathtt{9}\}^+ \\
(\textit{Integer}) & i & \in & \{\ldots, -2, -1, 0, 1, 2, \ldots\} \\
(\textit{Boolean}) & z & \in & \{\textit{true}, \textit{false}\}
\end{array}
$$

Figure 3.1: Syntax

command **skip** is a no-operation. The conditional command **if** b **then** c_1 **else** c_2 tests whether b holds; if it holds c_1 is executed, otherwise c_2 is executed. The loop command executes c while condition b holds, testing the condition before each iteration. The parallel composition command $c_1 \,\|\, c_2$ will perform the execution of both c_1 and c_2 as parallel threads; communication or interference between c_1 and c_2 may happen through shared variables or memory. The atomic command **atomic** c executes c in a single-shot; it is used to group operations accessing a shared resource preventing other threads from seeing it in a inconsistent state. The atomic command can be viewed as a synchronization block in high-level languages. But it can be viewed as an atomic operation available at the low-level, such as a memory write, a memory barrier, or a compare-and-swap (CAS) instruction. For instance, we can simulate a low-level compare-and-swap (CAS) operation:

$$\textbf{atomic } (v := [l]; \textbf{if } v = x \textbf{ then } [l] := y \textbf{ else skip}; y := v)$$

Higher-level synchronization primitives such as semaphores and mutexes can be implemented using this primitive construct.

An action a can be either an assignment ($\nu := e$), a memory allocation operation ($\nu := \textbf{cons}(e_1, \ldots, e_n)$), or a memory disposal operation ($\textbf{dispose}(e)$). An assignment $\nu := e$ evaluates the expression e and assigns the resulting integer to the location (either a variable or a memory address) obtained from the evaluation of the l-value ν. A memory allocation operation $\nu := \mathbf{cons}(e_1, \ldots, e_n)$ allocates a n-length consecutive block of fresh memory, assigning to each location i in the block (ranging from 1 to n) the integer value obtained from evaluating the associated expressions e_i; the starting address for the new block is assigned to the location obtained from the evaluation of the l-value $lval$. A memory disposal operation **dispose**(e) evaluates the expression e obtaining a memory location to be disposed, making it available to future allocation. (Note that disposal only disposes a single memory location, naturally, multiple dispose operations can be used to dispose a block of memory).

A l-value ν can be either a program variable (v) or a memory dereference ($[e]$).

An expression e can be either an integer constant (i), an l-value (ν), a binary arithmetic addition operator ($e_1 + e_2$), or an unary arithmetic negation operator ($-e$).

A condition b can be either a boolean constant (z), a binary boolean conjunction operator ($b_1 \wedge b_2$), an unary boolean negation operator ($\neg b$), or a binary arithmetic comparison operator for equality ($e_1 = e_2$) or less-than comparison ($e_1 < e_2$).

Both expressions and conditions can read program variables and the memory, but only actions can modify them. As a consequence, the evaluation order of expressions, conditions, and l-values in a given action does not matter and thus need not to be specified.

A variable v is a case-sensitive non-empty sequence of letters and decimal digits. `i`, `count`, `fly2`, `4mat` are examples of program variables.

3.2 Syntactic Sugar

In all examples presented in this thesis we make use of some syntactic sugar which is shown in Fig. 3.2. In this table we have the extended language syntax for the construct (on the left hand side) and the associated translation into the core syntax as presented in Fig. 3.1 (on the right hand side).

Syntax	Expansion
if b **then** c	**if** b **then** c **else skip**
repeat c **until** b	c; **while** $\neg b$ **do** c
for $\nu := e_1$ **to** e_2 **do** c	$\nu := e_1$; **while** $\nu < e_2 + 1$ **do** $(c; \nu := \nu + 1)$
dispose(e, n)	**dispose**$((e+1)-1); \ldots ;$ **dispose**$((e+n)-1)$
wait b	**while** $\neg b$ **do skip**
when b **do** c	**atomic** (**while** $\neg b$ **do skip**; c)
$\langle a \rangle$	**atomic** a
$e_1 - e_2$	$e_1 + - e_2$
$b_1 \vee b_2$	$\neg(\neg b_1 \wedge \neg b_2)$
$e_1 \neq e_2$	$\neg(e_1 = e_2)$
$e_1 \leq e_2$	$e_1 < e_2 + 1$
$e_1 \geq e_2$	$\neg(e_1 < e_2)$
$e_1 > e_2$	$\neg(e_1 < e_2 + 1)$

Figure 3.2: Syntactic sugar

3.3 Runtime Objects and Footprints

In order to present the formal language semantics, we need some additional structures to maintain the runtime execution environment. These structures are presented in Fig. 3.3.

$$\begin{array}{rrcl}
(\mathit{Config}) & \kappa & ::= & \langle T, \sigma \rangle \mid \mathsf{abort} \mid \mathsf{race} \\
(\mathit{State}) & \sigma & \in & \mathit{Location} \rightharpoonup_{\text{fin}} \mathit{Integer} \\
(\mathit{Location}) & \ell & \in & \mathit{Variable} + \mathit{Integer} \\
(\mathit{ThreadTree}) & T & ::= & c \mid \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c \mid \langle\!\langle T \rangle\!\rangle_{\mathbf{a}} c
\end{array}$$

Figure 3.3: Runtime objects

A program configuration κ is either a normal configuration ($\langle T, \sigma \rangle$) or an abnormal configuration (abort or race). A normal configuration $\langle T, \sigma \rangle$ is a pair composed of a thread tree T and a state σ. An abnormal configuration is either an abort or a race which are special configurations used to represent respectively, memory violation and a race condition.

A state σ is a finite map from location to integers. A location can be either a program variable or a memory address. We use a single map to represent both the variable store and the memory heap. In other words, we treat variables as resources [61].

A thread tree T is an intermediate representation of a command. It can be either a command *per se* (c), a tree node composed of two subtrees and a command ($\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$) representing the ongoing execution of a parallel composition, or a tree node composed of one subtree and a command ($\langle\!\langle T \rangle\!\rangle_{\mathbf{a}} c$) representing an ongoing execution of an atomic block. In the last two cases, the command c can be seen as a subsequent program continuation.

We also define special classes of thread trees which are called n-atomic thread trees. An n-atomic thread tree is a tree where there is exactly n top-level ongoing executions of atomic blocks.

Definition 3.1. A thread tree T is n-atomic, if, and only if, either

- $n = 0$ and $T = c$
- or, $n = 1$ and $T = \langle\!\langle T' \rangle\!\rangle_{\mathbf{a}} c$
- or, $n = n_1 + n_2$ and $T = \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$, where T_1 is n_1-atomic and T_2 is n_2-atomic

Example 3.2. Here some sample thread trees and their n-atomic class:

- c , $\textbf{atomic}\ c_1 \,\|\, c_2$, and $\langle\!\langle \textbf{atomic}\ c_2, \textbf{atomic}\ c_2 \rangle\!\rangle_{\mathbf{p}} c$, are all 0-atomic
- $\langle\!\langle t \rangle\!\rangle_{\mathbf{a}} c$, $\langle\!\langle \langle\!\langle t \rangle\!\rangle_{\mathbf{a}} c_1, \textbf{atomic}\ c_2 \rangle\!\rangle_{\mathbf{p}} c$, and $\langle\!\langle \langle\!\langle \langle\!\langle t_1 \rangle\!\rangle_{\mathbf{a}} c_1, \langle\!\langle t_2 \rangle\!\rangle_{\mathbf{a}} c_2 \rangle\!\rangle_{\mathbf{p}} c \rangle\!\rangle_{\mathbf{a}} c'$, are all 1-atomic
- $\langle\!\langle \langle\!\langle t_1 \rangle\!\rangle_{\mathbf{a}} c_1, \langle\!\langle t_2 \rangle\!\rangle_{\mathbf{a}} c_2 \rangle\!\rangle_{\mathbf{p}} c$ is 2-atomic

A very special runtime structure is a *footprint*, in which is required for the presentation of our sequential semantics (further in Sec. 3.6). A footprint δ represents a memory footprint of a command, which is the set of memory locations that it accesses. Furthermore, a footprint qualifies the type of access to each location, which can be either a read or write. We encode a memory footprint as a pair (rs, ws) where both rs and ws are a set of locations, as can be seen in Fig. 3.4.

In this encoding, a rs represents the set of locations to which a read operation is performed, and a ws represents the set of locations to which a write operation is performed.

$$\begin{array}{rrcl} (\mathit{Footprint}) & \delta & ::= & (rs, ws) \\ (\mathit{LocSet}) & rs, ws & \subseteq & \mathit{Location} \end{array}$$

Figure 3.4: Footprint

This is very convenient when constructing the footprint of a program from its text. Naturally, the set of read-only locations is obtained by the subtraction $rs \setminus ws$; and the set of read-or-write locations is obtained by the union of rs and ws. Following the same logic, in principle, we could obtain the set of write-only locations by the subtraction $ws \setminus rs$, and set of read-and-write locations by the intersection of rs and ws.

However, we take a more conservative interpretation of the footprint encoding. For the purpose of non-interference, it is not very useful to know that a given location is write-only, as a write-only location is simply as harmful as a read-and-write location. Therefore, in our interpretation of the encoding, ws should be viewed as the set of read-and-write locations, and there is no way to tell whether a location is write-only. This is a subtle difference in the interpretation, which is clearly reflected in the sub-footprint relation, e.g. $(\{\ell\}, \{\ell\}) \subseteq (\varnothing, \{\ell\})$. The definition of the sub-footprint relation is shown in Fig. 3.5 along with the other footprint related definitions.

$$\begin{aligned} emp &\stackrel{\text{def}}{=} (\varnothing, \varnothing) \\ \delta_1 \cup \delta_2 &\stackrel{\text{def}}{=} (\delta_1.rs \cup \delta_2.rs, \delta_1.ws \cup \delta_2.ws) \\ \delta_1 \subseteq \delta_2 &\stackrel{\text{def}}{=} (\delta_1.rs \subseteq \delta_2.rs \cup \delta_2.ws) \wedge (\delta_1.ws \subseteq \delta_2.ws) \\ \delta_1 \equiv \delta_2 &\stackrel{\text{def}}{=} \delta_1 \subseteq \delta_2 \wedge \delta_2 \subseteq \delta_1 \\ \delta_1 \subset \delta_2 &\stackrel{\text{def}}{=} \delta_1 \subseteq \delta_2 \wedge \delta_1 \not\subseteq \delta_2 \\ \delta_1 \overset{\rightharpoonup}{\smile} \delta_2 &\stackrel{\text{def}}{=} \delta_1.ws \cap (\delta_2.rs \cup \delta_2.ws) = \varnothing \\ \delta_1 \smile \delta_2 &\stackrel{\text{def}}{=} \delta_1 \overset{\rightharpoonup}{\smile} \delta_2 \wedge \delta_2 \overset{\rightharpoonup}{\smile} \delta_1 \end{aligned}$$

Figure 3.5: Auxiliary footprint definitions

The empty footprint emp is, naturally, defined as a pair of empty sets. The union of two footprints $\delta_1 \cup \delta_2$ is defined as the union of their respective rs's and ws's. The sub-footprint relation between two footprints $\delta_1 \subseteq \delta_2$ enforces that the δ_1 is "smaller" than

δ_2, i.e. the set of locations must be smaller or equal and some of the accesses can change from read-and-write to read-only. We define also footprint equivalence $\delta_1 \equiv \delta_2$ in terms of the sub-footprint relation, which is clearly distinct from structural equality $\delta_1 = \delta_2$. The proper sub-footprint relation $\delta_1 \subset \delta_2$ is also defined in terms of sub-footprint relation. The unidirectional non-interference between two footprints $\delta_1 \overset{\rightharpoonup}{\smile} \delta_2$ ensures that the set of read-and-write locations of δ_1 does not overlap with the set of read-or-write locations of δ_2; this ensures that any code with footprint δ_1 does not affect any code with footprint δ_2. The bidirectional non-interference between two footprints $\delta_1 \smile \delta_2$, naturally, ensures the unidirectional non-interference in both ways.

3.4 Program Contexts

In this thesis, we present the various operational semantics in a contextual manner. Therefore we define some useful command and thread tree contexts, as shown in Fig. 3.6.

$$\begin{array}{rrcl}
(\textit{SequContext}) & \mathbf{S} & ::= & \bullet \mid \mathbf{S}; c \\
(\textit{ThrdTreeContext}) & \mathbf{T} & ::= & \bullet \mid \langle\!\langle \mathbf{T}, T \rangle\!\rangle_{\mathbf{p}} c \mid \langle\!\langle T, \mathbf{T} \rangle\!\rangle_{\mathbf{p}} c \mid \langle\!\langle \mathbf{T} \rangle\!\rangle_{\mathbf{a}} c
\end{array}$$

Figure 3.6: Contexts

A command context — specified in general by $\mathcal{C}$ — is a command with a "hole". Some examples of command contexts are: $\bullet$, $\bullet; c$, $c; \bullet$, **if** b **then** $\bullet$, etc. We use the notation $\mathcal{C}[\, c\,]$ to obtain a command by replacing the hole in context $\mathcal{C}$ by the command c. For instance, $(c_1; \bullet)[\, c_2; c_3\,] = c_1; (c_2; c_3)$. We also overload this notation $\mathcal{C}_1[\, \mathcal{C}_2\,]$ to obtain a new context by replacing the hole in the given context $\mathcal{C}_1$ by another context $\mathcal{C}_2$. For example, $(c_1; \bullet)[\, \bullet; c_3\,] = c_1; (\bullet; c_3)$.

A sequential context $\mathbf{S}$ is a command context used to capture commands that start with some redex in the hole and may have a non-empty sequence of commands succeeding it.

Analogous to a command context, a thread tree context $\mathbf{T}$ is simply a thread tree with a hole. We also use the notations $\mathbf{T}[\, T\,]$ and $\mathbf{T}_1[\, \mathbf{T}_2\,]$ to replace the hole by a thread tree or a another thread tree context, respectively.

We define special classes of thread tree contexts which are call n-atomic thread three contexts. A n-atomic thread tree context is a context where produces an environment of exactly n ongoing executions of an atomic blocks. 0-atomic contexts are those where no atomic block is being executed.

Definition 3.3. A thread tree context $\mathbf{T}$ is n-atomic, if, and only if, either

- $n = 0$ and $\mathbf{T} = \bullet$
- or, $n = n_1 + n_2$ and $\mathbf{T} = \langle\!\langle \mathbf{T}', T \rangle\!\rangle_{\mathbf{p}} c$, where $\mathbf{T}'$ is n_1-atomic and T is n_2-atomic
- or, $n = n_1 + n_2$ and $\mathbf{T} = \langle\!\langle T, \mathbf{T}' \rangle\!\rangle_{\mathbf{p}} c$, where $\mathbf{T}'$ is n_1-atomic and T is n_2-atomic
- or, $\mathbf{T} = \langle\!\langle \mathbf{T}' \rangle\!\rangle_{\mathbf{a}} c$ where $\mathbf{T}'$ is also n-atomic

3.5 Semantics of Expressions and Actions

Before we present the sequential semantics, we need to present the semantics for the building blocks of commands: expressions and actions. We also present a function for computing the footprint of a given expression or action.

In Fig. 3.7, we present the semantics of l-values, expressions, conditions, and actions, respectively. We overload the notation $[\![-]\!]$ to represent the semantics of each of those syntactic categories.

We represent the semantics of l-values, conditions and expressions as a partial function from states to either a location, an integer value, or a boolean value, respectively. By being a function, we know it is deterministic, i.e. given the same expression and the same state, we always get the same result (e.g. $\forall e, \sigma.\ [\![e]\!]_\sigma = [\![e]\!]_\sigma$). By being a partial function, we can leave the result undefined whenever the expression access a location not present in the domain of the given state. For instance, if σ is empty, then $[\![\texttt{i}+1]\!]_\sigma$ is *undefined*; but, if we consider σ being $\{\texttt{i} \rightsquigarrow 5\}$, then $[\![\texttt{i}+1]\!]_\sigma$ is 6. Note that there is an abuse of notation in Fig. 3.7. Given that we are dealing with partial functions, when we write $[\![e_1]\!]_\sigma + [\![e_2]\!]_\sigma$, we

$$
\begin{aligned}
[\![\nu]\!] &\in \mathit{State} \rightharpoonup \mathit{Location} \\
[\![v]\!]_\sigma &\overset{\text{def}}{=} v \\
[\![[e]]\!]_\sigma &\overset{\text{def}}{=} [\![e]\!]_\sigma \\[1em]
[\![e]\!] &\in \mathit{State} \rightharpoonup \mathit{Integer} \\
[\![i]\!]_\sigma &\overset{\text{def}}{=} i \\
[\![\nu]\!]_\sigma &\overset{\text{def}}{=} \begin{cases} \sigma(\ell) & \text{when } [\![\nu]\!]_\sigma = \ell \text{ and } \ell \in \mathsf{dom}(\sigma) \\ \mathit{undefined} & \text{otherwise} \end{cases} \\
[\![e_1 + e_2]\!]_\sigma &\overset{\text{def}}{=} [\![e_1]\!]_\sigma + [\![e_2]\!]_\sigma \\
[\![-e]\!]_\sigma &\overset{\text{def}}{=} -[\![e]\!]_\sigma \\[1em]
[\![b]\!] &\in \mathit{State} \rightharpoonup \mathit{Boolean} \\
[\![z]\!]_\sigma &\overset{\text{def}}{=} z \\
[\![b_1 \wedge b_2]\!]_\sigma &\overset{\text{def}}{=} [\![b_1]\!]_\sigma \wedge [\![b_2]\!]_\sigma \\
[\![\neg b]\!]_\sigma &\overset{\text{def}}{=} \neg [\![b]\!]_\sigma \\
[\![e_1 = e_2]\!]_\sigma &\overset{\text{def}}{=} [\![e_1]\!]_\sigma = [\![e_2]\!]_\sigma \\
[\![e_1 < e_2]\!]_\sigma &\overset{\text{def}}{=} [\![e_1]\!]_\sigma < [\![e_2]\!]_\sigma \\[1em]
[\![a]\!] &\subseteq \mathit{State} \times \mathit{State} \\
[\![\nu := e]\!] &\overset{\text{def}}{=} \{(\sigma, \sigma\{\ell \rightsquigarrow i\}) \mid \ell \in \mathsf{dom}(\sigma) \wedge [\![\nu]\!]_\sigma = \ell \wedge [\![e]\!]_\sigma = i\} \\
[\![\nu := \mathbf{cons}(e_1, \ldots, e_n)]\!] &\overset{\text{def}}{=} \{(\sigma, \sigma\{\ell \rightsquigarrow n'+1, n'+1 \rightsquigarrow i_1, \ldots, n'+n \rightsquigarrow i_n\}) \\
&\qquad \mid \ell \in \mathsf{dom}(\sigma) \wedge n'+1 \notin \mathsf{dom}(\sigma) \wedge \ldots \wedge n'+n \notin \mathsf{dom}(\sigma) \\
&\qquad\quad \wedge [\![\nu]\!]_\sigma = \ell \wedge [\![e_1]\!]_\sigma = i_1 \wedge \ldots \wedge [\![e_n]\!]_\sigma = i_n\} \\
[\![\mathbf{dispose}(e)]\!] &\overset{\text{def}}{=} \{(\sigma, \sigma \backslash \{i\}) \mid i \in \mathsf{dom}(\sigma) \wedge [\![e]\!]_\sigma = i)\}
\end{aligned}
$$

Figure 3.7: Semantics of l-values, expressions, conditions, and actions

actually mean

$$
\begin{cases} i_1 + i_2 & \text{when } [\![e_1]\!] = i_1 \text{ and } [\![e_2]\!] = i_2 \\ \mathit{undefined} & \text{when } [\![e_1]\!] \text{ or } [\![e_2]\!] \text{ is } \mathit{undefined} \end{cases}
$$

On the other hand, we represent the semantics of actions as a relation of states. This is due to the fact that memory allocation is non-deterministic, therefore, we cannot use a partial function to express this. The other two actions, assignment and memory disposal, are deterministic. For instance, assuming σ being $\{\mathtt{x} \rightsquigarrow 42\}$, all the pairs in the infinite sequence $(\sigma, \{\mathtt{x} \rightsquigarrow 1, 1 \rightsquigarrow 43\})$, $(\sigma, \{\mathtt{x} \rightsquigarrow 2, 2 \rightsquigarrow 43\})$, $(\sigma, \{\mathtt{x} \rightsquigarrow 3, 3 \rightsquigarrow 43\})$, …, belong to $[\![\mathtt{x} := \mathbf{cons}(\mathtt{x}+1)]\!]$. The memory location of the newly allocated block is always a positive integer. Also, since the state is a finite partial map, there is always a positive memory location where a n-length block can be allocated. The only case where memory allocation fails, i.e. $\not\exists\sigma'.(\sigma,\sigma') \in [\![\nu := \mathbf{cons}(e_1,\ldots,e_n)]\!]$, is when the l-value fails to be evaluated for the given state; in other words $[\![\nu]\!]_\sigma$ is undefined. The semantics assignment $[\![\nu := e]\!]$ is straightforward. For a given state σ, either there is at most one σ' such that $(\sigma,\sigma') \in [\![\nu := e]\!]$. If there is no such σ' then the assignment failed. For instance, for σ being $\{\mathtt{x} \rightsquigarrow 42\}$, the assignment $\mathtt{x} := \mathtt{y}+1$ fails, i.e. $\not\exists\sigma'.\ (\sigma,\sigma') \in [\![\mathtt{x} := \mathtt{y}+1]\!]$. Similarly, assignments $\mathtt{y} := \mathtt{x}+1$ and $[\mathtt{x}] := \mathtt{x}+1$ also fail for that given state. The semantics of memory disposal $[\![\mathbf{dispose}(e)]\!]$ is also straightforward. It removes the memory location obtained by evaluating expression e from the domain of a given state σ. It fails if $[\![e]\!]_\sigma$ is undefined, or if $[\![e]\!]_\sigma \notin \mathsf{dom}(\sigma)$ (i.e. deallocation of a non-allocated memory cell).

$$
\begin{array}{lcl}
\mathsf{L}^\nu & \in & \mathit{State} \to \mathit{LocSet} \\
\mathsf{L}^\nu_\sigma & \stackrel{\text{def}}{=} & \begin{cases} \{\ell\} & \text{when } [\![\nu]\!]_\sigma = \ell \\ \varnothing & \text{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{lcl}
\Delta^\nu & \in & \mathit{State} \to \mathit{Footprint} \\
\Delta^\nu_\sigma & \stackrel{\text{def}}{=} & \begin{cases} \mathit{emp} & \text{when } \nu = v \\ \Delta^e_\sigma & \text{when } \nu = [e] \end{cases}
\end{array}
$$

$$
\begin{array}{lcl}
\Delta^e & \in & \mathit{State} \to \mathit{Footprint} \\
\Delta^e_\sigma & \stackrel{\text{def}}{=} & \begin{cases} \mathit{emp} & \text{when } e = i \\ \Delta^\nu_\sigma \cup (\mathsf{L}^\nu_\sigma, \varnothing) & \text{when } e = \nu \\ \Delta^{e_1}_\sigma \cup \Delta^{e_2}_\sigma & \text{when } e = e_1 + e_2 \\ \Delta^{e'}_\sigma & \text{when } e = -e' \end{cases}
\end{array}
$$

$$
\begin{array}{lcl}
\Delta^b & \in & \mathit{State} \to \mathit{Footprint} \\
\Delta^b_\sigma & \stackrel{\text{def}}{=} & \begin{cases} \mathit{emp} & \text{when } b = z \\ \Delta^{b_1}_\sigma \cup \Delta^{b_2}_\sigma & \text{when } b = b_1 \wedge b_2 \\ \Delta^{b'}_\sigma & \text{when } b = \neg b' \\ \Delta^{e_1}_\sigma \cup \Delta^{e_2}_\sigma & \text{when } b = e_1 = e_2 \\ \Delta^{e_1}_\sigma \cup \Delta^{e_2}_\sigma & \text{when } b = e_1 < e_2 \end{cases}
\end{array}
$$

$$
\begin{array}{lcl}
\Delta^a & \in & \mathit{State} \to \mathit{Footprint} \\
\Delta^a_\sigma & \stackrel{\text{def}}{=} & \begin{cases} \Delta^\nu_\sigma \cup \Delta^e_\sigma \cup (\varnothing, \mathsf{L}^\nu_\sigma) & \text{when } a = \nu := e \\ \Delta^\nu_\sigma \cup \Delta^{e_1}_\sigma \cup \ldots \cup \Delta^{e_n}_\sigma \cup (\varnothing, \mathsf{L}^\nu_\sigma) & \text{when } a = \nu := \mathbf{cons}(e_1,\ldots,e_n) \\ \Delta^e_\sigma \cup (\varnothing, \mathsf{L}^{[e]}_\sigma) & \text{when } a = \mathbf{dispose}(e) \end{cases}
\end{array}
$$

Figure 3.8: Footprint of l-values, expressions, conditions, and actions

In Fig. 3.8 we present a set of definitions used to calculate the footprint of l-values, expressions, conditions and actions.

First, we define L^ν as a function from states to locations sets. This function simply tests if $[\![\nu]\!]$ for the given state σ is defined. If it is defined, then it returns a singleton location set containing the corresponding location. Otherwise, it returns an empty set. The L^ν function is an auxiliary function that simply calculates the location accessed through a defined l-value, which in fact will eventually become part of either rs or ws depending on whether the ν was used as an l-value or as an expression, respectively.

The remaining of the definitions, Δ^ν, Δ^e, Δ^b, and Δ^a, are straightforward footprint calculations, done mostly by taking the union of the footprints of subexpressions. There are some remarks thought:

1. We do not include the location corresponding to an l-value, L^ν, in the computation of its footprint Δ^ν. In fact, we include it later in the ws of a given action or in the rs of a given expression, depending on whether it was used as an l-value or as an expression, respectively.

2. We do not include freshly allocated memory cells in the calculation of the footprint of memory allocation. Since memory allocation is non-deterministic, such behavior cannot be defined as a function. These cells are treated specially in the semantics, as it will become clear in the next section.

We also define a function to extract the largest footprint compatible with a state.

Definition 3.4. The function $\nabla(\sigma)$ is defined as $(\emptyset, \mathsf{dom}(\sigma))$

3.6 Sequential Semantics

We are now ready to present a structural operational semantics for the sequential core of the language. The semantics is shown in Fig. 3.9. It is presented in a contextual fashion, using for each rule a sequential context $\mathbf{S}$ as defined in Fig. 3.6. The sequential context $\mathbf{S}$ defines the places where the execution of primitive commands occur.

$$
\begin{array}{rcll}
\langle \mathbf{S}[\,a\,],\sigma\rangle & \longrightarrow & \mathsf{abort} & \text{if } \nexists\sigma'.\ (\sigma,\sigma')\in[\![a]\!] \\
\langle \mathbf{S}[\,a\,],\sigma\rangle & \longrightarrow & \langle \mathbf{S}[\,\mathbf{skip}\,],\sigma'\rangle & \text{if } (\sigma,\sigma')\in[\![a]\!] \\
\langle \mathbf{S}[\,\mathbf{skip}; c\,],\sigma\rangle & \longrightarrow & \langle \mathbf{S}[\,c\,],\sigma\rangle & \text{always} \\
\langle \mathbf{S}[\,\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\,],\sigma\rangle & \longrightarrow & \mathsf{abort} & \text{if } \nexists z.\ [\![b]\!]_\sigma = z \\
\langle \mathbf{S}[\,\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\,],\sigma\rangle & \longrightarrow & \langle \mathbf{S}[\,c_1\,],\sigma\rangle & \text{if } [\![b]\!]_\sigma = \mathit{true} \\
\langle \mathbf{S}[\,\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\,],\sigma\rangle & \longrightarrow & \langle \mathbf{S}[\,c_2\,],\sigma\rangle & \text{if } [\![b]\!]_\sigma = \mathit{false} \\
\langle \mathbf{S}[\,\mathbf{while}\ b\ \mathbf{do}\ c\,],\sigma\rangle & \longrightarrow & \multicolumn{2}{l}{\langle \mathbf{S}[\,\mathbf{if}\ b\ \mathbf{then}\ (c; \mathbf{while}\ b\ \mathbf{do}\ c)\ \mathbf{else\ skip}\,],\sigma\rangle} \\
 & & & \text{always}
\end{array}
$$

Figure 3.9: Sequential semantics

This semantics is fairly standard, and its understanding should be straightforward. There are four types of execution rules for this semantics. The first two rules define the execution of actions. An action a executes by transitioning from state σ to state σ' and long as the pair (σ, σ') belongs to the relation defined by $[\![a]\!]$. In this case, **skip** is also replaces a in the command text, achieving the effect of removing a from the text. Otherwise, if no such transition is possible, then the execution aborts by stepping into an abort configuration. The third rule defines the execution for a **skip**, which in practice on removes it from the text. The fourth, fifth and sixth rules define the execution of conditional commands. A conditional command **if** b **then** c_1 **else** c_2 executes by testing whether the condition b holds in the current state σ. This is done by testing whether the partial function $[\![b]\!]$ yields *true* or *false*, choosing, respectively either c_1 or c_2. If $[\![b]\!]$ is undefined the execution steps into an abort configuration. Finally, the last rule defines the execution of a while loop simply by performing a syntactic translation. It uses a conditional command to test the condition b and decide whether the loop body must execute one followed by the loop itself, or it it should cease execution through **skip**.

Note that in this semantics there are configurations for which the stepping is not defined, such as for those that start with an atomic block or a parallel composition. We would like to stress that this simply means that the sequential execution does not proceed from that point on. It does not mean being "stuck" as usually happens in traditional definitions of sequential operational semantics. In our semantics, an unbehaved execution is represented explicitly as stepping into an abort configuration.

Example 3.5. Here some examples of execution according to the sequential semantics:

- $\langle \mathtt{x} := \mathtt{y}, \{\mathtt{x} \rightsquigarrow 0, \mathtt{y} \rightsquigarrow 5\}\rangle \longrightarrow \langle \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 5, \mathtt{y} \rightsquigarrow 5\}\rangle$
- $\langle \mathtt{x} := \mathtt{y}, \{\mathtt{x} \rightsquigarrow 0\}\rangle \longrightarrow \mathsf{abort}$
- $\langle \mathtt{x} := \mathtt{y}, \{\mathtt{y} \rightsquigarrow 5\}\rangle \longrightarrow \mathsf{abort}$
- $\langle \mathtt{x} := [\mathtt{y}], \{\mathtt{x} \rightsquigarrow 0, \mathtt{y} \rightsquigarrow 5, 5 \rightsquigarrow 3\}\rangle \longrightarrow \langle \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 3, \mathtt{y} \rightsquigarrow 5, 5 \rightsquigarrow 3\}\rangle$
- $\langle \mathtt{x} := [\mathtt{y}], \{\mathtt{x} \rightsquigarrow 0, \mathtt{y} \rightsquigarrow 5\}\rangle \longrightarrow \mathsf{abort}$
- $\langle \mathtt{x} := [\mathtt{y}], \{\mathtt{x} \rightsquigarrow 0, 5 \rightsquigarrow 3\}\rangle \longrightarrow \mathsf{abort}$
- $\langle [\mathtt{x}] := \mathtt{y}, \{\mathtt{x} \rightsquigarrow 5, \mathtt{y} \rightsquigarrow 3, 5 \rightsquigarrow 0\}\rangle \longrightarrow \langle \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 5, \mathtt{y} \rightsquigarrow 3, 5 \rightsquigarrow 3\}\rangle$
- $\langle [\mathtt{x}] := \mathtt{y}, \{\mathtt{x} \rightsquigarrow 33, \mathtt{y} \rightsquigarrow 3, 5 \rightsquigarrow 0\}\rangle \longrightarrow \mathsf{abort}$
- $\langle [\mathtt{x}] := \mathtt{y}, \{\mathtt{y} \rightsquigarrow 3, 33 \rightsquigarrow 0\}\rangle \longrightarrow \mathsf{abort}$
- $\langle \mathtt{x} := \mathtt{y} + \mathtt{z}, \{\mathtt{x} \rightsquigarrow 5, \mathtt{y} \rightsquigarrow 3, \mathtt{z} \rightsquigarrow 4\}\rangle \longrightarrow \langle \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 7, \mathtt{y} \rightsquigarrow 3, \mathtt{z} \rightsquigarrow 4\}\rangle$
- $\langle \mathtt{x} := \mathbf{cons}(5), \{\mathtt{x} \rightsquigarrow 0\}\rangle \longrightarrow \langle \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 33, 33 \rightsquigarrow 5\}\rangle$
- $\langle \mathtt{x} := \mathbf{cons}(5), \{\mathtt{x} \rightsquigarrow 0\}\rangle \longrightarrow \langle \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 44, 44 \rightsquigarrow 5\}\rangle$
- $\langle \mathtt{x} := \mathbf{cons}(5), \emptyset\rangle \longrightarrow \mathsf{abort}$
- $\langle \mathtt{x} := \mathbf{cons}(5, 6), \{\mathtt{x} \rightsquigarrow 0\}\rangle \longrightarrow \langle \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 44, 44 \rightsquigarrow 5, 45 \rightsquigarrow 6\}\rangle$
- $\langle \mathbf{dispose}(\mathtt{x}), \{\mathtt{x} \rightsquigarrow 44, 44 \rightsquigarrow 5\}\rangle \longrightarrow \langle \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 44\}\rangle$
- $\langle \mathbf{dispose}(\mathtt{x}), \{\mathtt{x} \rightsquigarrow 44\}\rangle \longrightarrow \mathsf{abort}$
- $\langle \mathtt{y} := \mathtt{x}; \mathtt{z} := \mathtt{y}, \{\mathtt{x} \rightsquigarrow 5, \mathtt{y} \rightsquigarrow 3, \mathtt{z} \rightsquigarrow 4\}\rangle \longrightarrow \langle \mathbf{skip}; \mathtt{z} := \mathtt{y}, \{\mathtt{x} \rightsquigarrow 5, \mathtt{y} \rightsquigarrow 5, \mathtt{z} \rightsquigarrow 4\}\rangle$
- $\langle \mathbf{skip}; \mathtt{z} := \mathtt{y}, \{\mathtt{x} \rightsquigarrow 5, \mathtt{y} \rightsquigarrow 5, \mathtt{z} \rightsquigarrow 4\}\rangle \longrightarrow \langle \mathtt{z} := \mathtt{y}, \{\mathtt{x} \rightsquigarrow 5, \mathtt{y} \rightsquigarrow 5, \mathtt{z} \rightsquigarrow 4\}\rangle$

- $\langle \textbf{if}\ \mathtt{y} < 0\ \textbf{then}\ \mathtt{x} := \mathtt{y}\ \textbf{else}\ \mathtt{x} := -\mathtt{y}, \{\mathtt{y} \rightsquigarrow 5\}\rangle \longrightarrow \langle \mathtt{x} := -\mathtt{y}, \{\mathtt{y} \rightsquigarrow 5\}\rangle$

- $\langle \textbf{if}\ \mathtt{y} < 0\ \textbf{then}\ \mathtt{x} := \mathtt{y}\ \textbf{else}\ \mathtt{x} := -\mathtt{y}, \{\mathtt{x} \rightsquigarrow 0\}\rangle \longrightarrow \mathsf{abort}$

- $\langle \textbf{while}\ \mathtt{x} < 10\ \textbf{do}\ \mathtt{x} := \mathtt{x}+1, \{\mathtt{x} \rightsquigarrow 0\}\rangle \longrightarrow$

  $\langle \textbf{if}\ \mathtt{x} < 10\ \textbf{then}\ (\mathtt{x} := \mathtt{x}+1; \textbf{while}\ \mathtt{x} < 10\ \textbf{do}\ \mathtt{x} := \mathtt{x}+1)\ \textbf{else skip}, \{\mathtt{x} \rightsquigarrow 0\}\rangle$

- $\langle \textbf{while}\ \mathtt{x} < 10\ \textbf{do}\ \mathtt{x} := \mathtt{x}+1, \emptyset\rangle \longrightarrow$

  $\langle \textbf{if}\ \mathtt{x} < 10\ \textbf{then}\ (\mathtt{x} := \mathtt{x}+1; \textbf{while}\ \mathtt{x} < 10\ \textbf{do}\ \mathtt{x} := \mathtt{x}+1)\ \textbf{else skip}, \emptyset\rangle$

$$
\begin{array}{rcll}
\langle \mathbf{S}[\,a\,], \sigma\rangle & \xrightarrow[\Delta^a_\sigma]{} & \mathsf{abort} & \text{if } \nexists\sigma'.\ (\sigma,\sigma') \in [\![a]\!] \\
\langle \mathbf{S}[\,a\,], \sigma\rangle & \xrightarrow[\Delta^a_\sigma \cup \delta]{} & \langle \mathbf{S}[\,\textbf{skip}\,], \sigma'\rangle & \text{if } (\sigma,\sigma') \in [\![a]\!] \wedge \delta = (\emptyset, \mathsf{dom}(\sigma') \backslash \mathsf{dom}(\sigma)) \\
\langle \mathbf{S}[\,\textbf{skip}; c\,], \sigma\rangle & \xrightarrow[emp]{} & \langle \mathbf{S}[\,c\,], \sigma\rangle & \text{always} \\
\langle \mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\,], \sigma\rangle & \xrightarrow[\Delta^b_\sigma]{} & \mathsf{abort} & \text{if } \nexists z.\ [\![b]\!]_\sigma = z \\
\langle \mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\,], \sigma\rangle & \xrightarrow[\Delta^b_\sigma]{} & \langle \mathbf{S}[\,c_1\,], \sigma\rangle & \text{if } [\![b]\!]_\sigma = \mathit{true} \\
\langle \mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\,], \sigma\rangle & \xrightarrow[\Delta^b_\sigma]{} & \langle \mathbf{S}[\,c_2\,], \sigma\rangle & \text{if } [\![b]\!]_\sigma = \mathit{false} \\
\langle \mathbf{S}[\,\textbf{while}\ b\ \textbf{do}\ c\,], \sigma\rangle & \xrightarrow[emp]{} & \multicolumn{2}{l}{\langle \mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ (c; \textbf{while}\ b\ \textbf{do}\ c)\ \textbf{else skip}\,], \sigma\rangle} \\
 & & & \text{always}
\end{array}
$$

Figure 3.10: Sequential semantics with footprints

In Fig. 3.10 we present a sequential semantics equivalent to the one shown in Fig. 3.9 adorned with footprints. For each rule in Fig. 3.9 there is a similar corresponding rule in Fig. 3.10. The Δ functions used to calculate the footprints are defined in Fig. 3.8. The only non-trivial footprint calculation is for the second rule where the set of locations present in σ' and not present in σ are included in the read-write set. This is due to the presence of memory allocation for which the newly allocated cells were not included in the read-write set by the function Δ^a, given that there is some non-determinism associated with their computation by $[\![a]\!]$.

Example 3.6. Here the same set of examples from Example 3.5 adorned with footprints:

- $\langle \mathrm{x} := \mathrm{y}, \{\mathrm{x} \rightsquigarrow 0, \mathrm{y} \rightsquigarrow 5\}\rangle \underset{(\{\mathtt{y}\},\{\mathtt{x}\})}{\longrightarrow} \langle \mathbf{skip}, \{\mathrm{x} \rightsquigarrow 5, \mathrm{y} \rightsquigarrow 5\}\rangle$
- $\langle \mathrm{x} := \mathrm{y}, \{\mathrm{x} \rightsquigarrow 0\}\rangle \underset{(\{\mathtt{y}\},\{\mathtt{x}\})}{\longrightarrow} \mathsf{abort}$
- $\langle \mathrm{x} := \mathrm{y}, \{\mathrm{y} \rightsquigarrow 5\}\rangle \underset{(\{\mathtt{y}\},\{\mathtt{x}\})}{\longrightarrow} \mathsf{abort}$
- $\langle \mathrm{x} := [\mathrm{y}], \{\mathrm{x} \rightsquigarrow 0, \mathrm{y} \rightsquigarrow 5, 5 \rightsquigarrow 3\}\rangle \underset{(\{\mathtt{y},5\},\{\mathtt{x}\})}{\longrightarrow} \langle \mathbf{skip}, \{\mathrm{x} \rightsquigarrow 3, \mathrm{y} \rightsquigarrow 5, 5 \rightsquigarrow 3\}\rangle$
- $\langle \mathrm{x} := [\mathrm{y}], \{\mathrm{x} \rightsquigarrow 0, \mathrm{y} \rightsquigarrow 5\}\rangle \underset{(\{\mathtt{y},5\},\{\mathtt{x}\})}{\longrightarrow} \mathsf{abort}$
- $\langle \mathrm{x} := [\mathrm{y}], \{\mathrm{x} \rightsquigarrow 0, 5 \rightsquigarrow 3\}\rangle \underset{(\{\mathtt{y}\},\{\mathtt{x}\})}{\longrightarrow} \mathsf{abort}$
- $\langle [\mathrm{x}] := \mathrm{y}, \{\mathrm{x} \rightsquigarrow 5, \mathrm{y} \rightsquigarrow 3, 5 \rightsquigarrow 0\}\rangle \underset{(\{\mathtt{y},\mathtt{x}\},\{5\})}{\longrightarrow} \langle \mathbf{skip}, \{\mathrm{x} \rightsquigarrow 5, \mathrm{y} \rightsquigarrow 3, 5 \rightsquigarrow 3\}\rangle$
- $\langle [\mathrm{x}] := \mathrm{y}, \{\mathrm{x} \rightsquigarrow 33, \mathrm{y} \rightsquigarrow 3, 5 \rightsquigarrow 0\}\rangle \underset{(\{\mathtt{y},\mathtt{x}\},\{33\})}{\longrightarrow} \mathsf{abort}$
- $\langle [\mathrm{x}] := \mathrm{y}, \{\mathrm{y} \rightsquigarrow 3, 33 \rightsquigarrow 0\}\rangle \underset{(\{\mathtt{y},\mathtt{x}\},\varnothing)}{\longrightarrow} \mathsf{abort}$
- $\langle \mathrm{x} := \mathrm{y}+\mathrm{z}, \{\mathrm{x} \rightsquigarrow 5, \mathrm{y} \rightsquigarrow 3, \mathrm{z} \rightsquigarrow 4\}\rangle \underset{(\{\mathtt{y},\mathtt{z}\},\{\mathtt{x}\})}{\longrightarrow} \langle \mathbf{skip}, \{\mathrm{x} \rightsquigarrow 7, \mathrm{y} \rightsquigarrow 3, \mathrm{z} \rightsquigarrow 4\}\rangle$
- $\langle \mathrm{x} := \mathbf{cons}(5), \{\mathrm{x} \rightsquigarrow 0\}\rangle \underset{(\varnothing,\{\mathtt{x},33\})}{\longrightarrow} \langle \mathbf{skip}, \{\mathrm{x} \rightsquigarrow 33, 33 \rightsquigarrow 5\}\rangle$
- $\langle \mathrm{x} := \mathbf{cons}(5), \{\mathrm{x} \rightsquigarrow 0\}\rangle \underset{(\varnothing,\{\mathtt{x},44\})}{\longrightarrow} \langle \mathbf{skip}, \{\mathrm{x} \rightsquigarrow 44, 44 \rightsquigarrow 5\}\rangle$
- $\langle \mathrm{x} := \mathbf{cons}(5), \varnothing\rangle \underset{(\varnothing,\{\mathtt{x}\})}{\longrightarrow} \mathsf{abort}$
- $\langle \mathrm{x} := \mathbf{cons}(5, 6), \{\mathrm{x} \rightsquigarrow 0\}\rangle \underset{(\varnothing,\{\mathtt{x},44,45\})}{\longrightarrow} \langle \mathbf{skip}, \{\mathrm{x} \rightsquigarrow 44, 44 \rightsquigarrow 5, 45 \rightsquigarrow 6\}\rangle$
- $\langle \mathbf{dispose}(\mathrm{x}), \{\mathrm{x} \rightsquigarrow 44, 44 \rightsquigarrow 5\}\rangle \underset{(\{\mathtt{x}\},\{44\})}{\longrightarrow} \langle \mathbf{skip}, \{\mathrm{x} \rightsquigarrow 44\}\rangle$
- $\langle \mathbf{dispose}(\mathrm{x}), \{\mathrm{x} \rightsquigarrow 44\}\rangle \underset{(\{\mathtt{x}\},\{44\})}{\longrightarrow} \mathsf{abort}$
- $\langle \mathrm{y} := \mathrm{x}; \mathrm{z} := \mathrm{y}, \{\mathrm{x} \rightsquigarrow 5, \mathrm{y} \rightsquigarrow 3, \mathrm{z} \rightsquigarrow 4\}\rangle \underset{(\{\mathtt{x}\},\{\mathtt{y}\})}{\longrightarrow} \langle \mathbf{skip}; \mathrm{z} := \mathrm{y}, \{\mathrm{x} \rightsquigarrow 5, \mathrm{y} \rightsquigarrow 5, \mathrm{z} \rightsquigarrow 4\}\rangle$
- $\langle \mathbf{skip}; \mathrm{z} := \mathrm{y}, \{\mathrm{x} \rightsquigarrow 5, \mathrm{y} \rightsquigarrow 5, \mathrm{z} \rightsquigarrow 4\}\rangle \underset{(\varnothing,\varnothing)}{\longrightarrow} \langle \mathrm{z} := \mathrm{y}, \{\mathrm{x} \rightsquigarrow 5, \mathrm{y} \rightsquigarrow 5, \mathrm{z} \rightsquigarrow 4\}\rangle$
- $\langle \mathbf{if}\ \mathrm{y} < 0\ \mathbf{then}\ \mathrm{x} := \mathrm{y}\ \mathbf{else}\ \mathrm{x} := -\mathrm{y}, \{\mathrm{y} \rightsquigarrow 5\}\rangle \underset{(\{y\},\varnothing)}{\longrightarrow} \langle \mathrm{x} := -\mathrm{y}, \{\mathrm{y} \rightsquigarrow 5\}\rangle$
- $\langle \mathbf{if}\ \mathrm{y} < 0\ \mathbf{then}\ \mathrm{x} := \mathrm{y}\ \mathbf{else}\ \mathrm{x} := -\mathrm{y}, \{\mathrm{x} \rightsquigarrow 0\}\rangle \underset{(\{y\},\varnothing)}{\longrightarrow} \mathsf{abort}$

- $\langle \textbf{while}\ \mathtt{x}<10\ \textbf{do}\ \mathtt{x}:=\mathtt{x}+1, \{\mathtt{x}\rightsquigarrow 0\}\rangle \underset{(\varnothing,\varnothing)}{\longrightarrow}$
  $\langle \textbf{if}\ \mathtt{x}<10\ \textbf{then}\ (\mathtt{x}:=\mathtt{x}+1; \textbf{while}\ \mathtt{x}<10\ \textbf{do}\ \mathtt{x}:=\mathtt{x}+1)\ \textbf{else skip}, \{\mathtt{x}\rightsquigarrow 0\}\rangle$

- $\langle \textbf{while}\ \mathtt{x}<10\ \textbf{do}\ \mathtt{x}:=\mathtt{x}+1, \varnothing\rangle \underset{(\varnothing,\varnothing)}{\longrightarrow}$
  $\langle \textbf{if}\ \mathtt{x}<10\ \textbf{then}\ (\mathtt{x}:=\mathtt{x}+1; \textbf{while}\ \mathtt{x}<10\ \textbf{do}\ \mathtt{x}:=\mathtt{x}+1)\ \textbf{else skip}, \varnothing\rangle$

Following the adorned sequential semantics, we can define a multi-step by taking the union of the footprints of each individual step.

Definition 3.7. A sequential multi-step with footprint $\underset{\delta}{\longrightarrow}^n$ is defined as:

$$\frac{}{\kappa \underset{emp}{\longrightarrow}^0 \kappa} \qquad \frac{\kappa \underset{\delta_1}{\longrightarrow} \kappa'' \quad \kappa'' \underset{\delta_2}{\longrightarrow}^n \kappa'}{\kappa \underset{\delta_1\cup\delta_2}{\longrightarrow}^{n+1} \kappa'}$$

The reflexive transitive closure is defined as: $\kappa \underset{\delta}{\longrightarrow}^* \kappa' \iff \exists n.\ \kappa \underset{\delta}{\longrightarrow}^n \kappa'$.

Example 3.8. Here a similar set of examples from Example 3.27 adorned with footprints:

- $\langle \mathtt{x}:=[\mathtt{x}], \{\mathtt{x}\rightsquigarrow 33, 33\rightsquigarrow 44, 44\rightsquigarrow 7\}\rangle \underset{(\varnothing,\varnothing)}{\longrightarrow}^0 \langle \mathtt{x}:=[\mathtt{x}], \{\mathtt{x}\rightsquigarrow 33, 33\rightsquigarrow 44, 44\rightsquigarrow 7\}\rangle$

- $\langle \mathtt{x}:=[\mathtt{x}]; \mathtt{x}:=[\mathtt{x}], \{\mathtt{x}\rightsquigarrow 33, 33\rightsquigarrow 44, 44\rightsquigarrow 7\}\rangle \underset{(\{\mathtt{x},33\},\{\mathtt{x}\})}{\longrightarrow}^* \langle \mathtt{x}:=[\mathtt{x}], \{\mathtt{x}\rightsquigarrow 44, 33\rightsquigarrow 44, 44\rightsquigarrow 7\}\rangle$

- $\langle \mathtt{x}:=[\mathtt{x}]; \mathtt{x}:=[\mathtt{x}], \{\mathtt{x}\rightsquigarrow 33, 33\rightsquigarrow 44, 44\rightsquigarrow 7\}\rangle \underset{(\{\mathtt{x},33,44\},\{\mathtt{x}\})}{\longrightarrow}^* \langle \textbf{skip}, \{\mathtt{x}\rightsquigarrow 7, 33\rightsquigarrow 44, 44\rightsquigarrow 7\}\rangle$

- $\langle \mathtt{x}:=[\mathtt{x}]; \mathtt{x}:=[\mathtt{x}+1], \{\mathtt{x}\rightsquigarrow 33, 33\rightsquigarrow 44, 44\rightsquigarrow 7\}\rangle \underset{(\{\mathtt{x},33,45\},\{\mathtt{x}\})}{\longrightarrow}^* \mathsf{abort}$

- $\langle \textbf{while}\ \mathtt{x}<10\ \textbf{do}\ \mathtt{x}:=\mathtt{x}+1, \{\mathtt{x}\rightsquigarrow 0\}\rangle \underset{(\{\mathtt{x}\},\{\mathtt{x}\})}{\longrightarrow}^* \langle \textbf{while}\ \mathtt{x}<10\ \textbf{do}\ \mathtt{x}:=\mathtt{x}+1, \{\mathtt{x}\rightsquigarrow 2\}\rangle$

- $\langle \textbf{while}\ \mathtt{x}<10\ \textbf{do}\ \mathtt{x}:=\mathtt{x}+1, \{\mathtt{x}\rightsquigarrow 0\}\rangle \underset{(\{\mathtt{x}\},\{\mathtt{x}\})}{\longrightarrow}^* \langle \textbf{skip}, \{\mathtt{x}\rightsquigarrow 10\}\rangle$

- $\langle \mathtt{x}:=\textbf{cons}(0); [\mathtt{x}+1]:=3, \{\mathtt{x}\rightsquigarrow 5, 33\rightsquigarrow 0\}\rangle \underset{(\{\mathtt{x}\},\{\mathtt{x},22,23\})}{\longrightarrow}^* \mathsf{abort}$

- $\langle \mathtt{x}:=\textbf{cons}(0); [\mathtt{x}+1]:=3, \{\mathtt{x}\rightsquigarrow 5, 33\rightsquigarrow 0\}\rangle \underset{(\{\mathtt{x}\},\{\mathtt{x},32,33\})}{\longrightarrow}^* \langle \textbf{skip}, \{\mathtt{x}\rightsquigarrow 32, 32\rightsquigarrow 0, 33\rightsquigarrow 3\}\rangle$

- $\langle \textbf{while}\ \mathtt{x}\neq 10\ \textbf{do}\ \mathtt{x}:=\textbf{cons}(\mathtt{x}), \{\mathtt{x}\rightsquigarrow 0\}\rangle \underset{(\{\mathtt{x}\},\{\mathtt{x},10,33,44\})}{\longrightarrow}^*$
  $\langle \textbf{skip}, \{\mathtt{x}\rightsquigarrow 10, 10\rightsquigarrow 44, 33\rightsquigarrow 0, 44\rightsquigarrow 33\}\rangle$

Now, it should be trivial to establish the equivalence between the plain sequential semantics of Fig. 3.9 and the adorned sequential semantics of Fig. 3.10.

Remark 3.9. The following holds trivially:

1. $\kappa \longrightarrow \kappa' \iff \exists\delta.\ \kappa \xrightarrow[\delta]{} \kappa'$
2. $\kappa \longrightarrow^n \kappa' \iff \exists\delta.\ \kappa \xrightarrow[\delta]{}^n \kappa'$
3. $\kappa \longrightarrow^* \kappa' \iff \exists\delta.\ \kappa \xrightarrow[\delta]{}^* \kappa'$

Remark 3.10. The following holds:

1. If $\langle c, \sigma\rangle \xrightarrow[\delta]{} \mathsf{abort}$, then $\delta \not\subseteq \nabla(\sigma)$
2. If $\langle c, \sigma\rangle \xrightarrow[\delta]{} \langle c', \sigma'\rangle$, then $\delta \subseteq \nabla(\sigma) \cup \nabla(\sigma')$

Framing properties. Here are the framing properties [73] of our sequential semantics, which basically show that any safe code will work the exact same way in a larger context with the exact same footprint. Also, as expected, assignments do not modify the domain of the state.

Remark 3.11. If $(\sigma_1, \sigma_1') \in [\![a]\!]$, and $\sigma = \sigma_1 \uplus \sigma_2$, then

1. exists σ' such that $(\sigma, \sigma') \in [\![a]\!]$
2. If $(\sigma, \sigma'') \in [\![a]\!]$, then exists σ_1'' such that $\sigma'' = \sigma_1'' \uplus \sigma_2$ and $(\sigma_1, \sigma_1'') \in [\![a]\!]$

Remark 3.12. If $[\![b]\!]_{\sigma_1} = z$, and $\sigma = \sigma_1 \uplus \sigma_2$, then $[\![b]\!]_{\sigma} = z$

Remark 3.13. If $(\sigma, \sigma') \in [\![\nu := e]\!]$, then $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$

Remark 3.14. If $(\sigma_1, \sigma_1') \in [\![a]\!]$, and $\sigma = \sigma_1 \uplus \sigma_2$, then $\Delta^a_\sigma = \Delta^a_{\sigma_1}$

Remark 3.15. If $[\![b]\!]_{\sigma_1} = z$, and $\sigma = \sigma_1 \uplus \sigma_2$, then $\Delta^b_\sigma = \Delta^b_{\sigma_1}$

Lemma 3.16. If $\neg\langle c, \sigma_1\rangle \longrightarrow \mathsf{abort}$, and $\sigma = \sigma_1 \uplus \sigma_2$, then

1. $\neg\langle c, \sigma\rangle \longrightarrow \mathsf{abort}$

2. If $\langle c, \sigma\rangle \underset{\delta}{\longrightarrow} \langle c', \sigma'\rangle$, then exists σ_1' such that $\sigma' = \sigma_1' \uplus \sigma_2$ and $\langle c, \sigma_1\rangle \underset{\delta}{\longrightarrow} \langle c', \sigma_1'\rangle$

Proof. Assuming (a) $\neg\langle c, \sigma_1\rangle \longrightarrow \mathsf{abort}$ and (b) $\sigma = \sigma_1 \uplus \sigma_2$, we have 2 cases:

- If we assume (c) $\langle c, \sigma\rangle \longrightarrow \mathsf{abort}$, we need to show that we reach a contradiction. Therefore, given (a), we will show that $\langle c, \sigma_1\rangle \longrightarrow \mathsf{abort}$. From the semantics, and (c), there are 2 cases:
  - We have (d) $c = \mathbf{S}[\,a\,]$ and (e) $\nexists\sigma'.\ (\sigma, \sigma') \in [\![a]\!]$, and we need to show that $\nexists\sigma_1'.\ (\sigma_1, \sigma_1') \in [\![a]\!]$. From (e) and (b), by contradiction, using Remark 3.11 (item 1), we conclude
  - We have (d) $c = \mathbf{S}[\,\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\,]$ and (e) $\nexists z.[\![b]\!]_\sigma = z$, and we need to show that $\nexists z'.\ [\![b]\!]_{\sigma_1} = z'$. From (e) and (b), by contradiction, using Remark 3.12, we conclude

- If (c) $\langle c, \sigma\rangle \underset{\delta}{\longrightarrow} \langle c', \sigma'\rangle$, from the semantics, there are 5 cases:
  - We have (d) $c = \mathbf{S}[\,a\,]$, (e) $c' = \mathbf{S}[\,\mathbf{skip}\,]$, (f) $\delta = \Delta^a_\sigma \cup (\emptyset, \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma))$, and (g) $(\sigma, \sigma') \in [\![a]\!]$. From (d), and (a), we know there exists σ_1', such that (h) $(\sigma_1, \sigma_1') \in [\![a]\!]$. From (h), (b), and (g), using Remark 3.11 (item 2), we know there exists σ_1'' such that (i) $\sigma' = \sigma_1'' \uplus \sigma_2$ and (j) $(\sigma_1, \sigma_1'') \in [\![a]\!]$. We instantiate the goal with σ_1'', and remains to show that $\sigma' = \sigma_1'' \uplus \sigma_2$ and $\langle c, \sigma_1\rangle \underset{\delta}{\longrightarrow} \langle c', \sigma_1''\rangle$. From the semantics, and (j), we have (k) $\langle \mathbf{S}[\,a\,], \sigma_1\rangle \underset{\Delta^a_{\sigma_1} \cup \delta'}{\longrightarrow} \langle \mathbf{S}[\,\mathbf{skip}\,], \sigma_1''\rangle$, where (l) $\delta' = (\emptyset, \mathsf{dom}(\sigma_1'') \setminus \mathsf{dom}(\sigma_1))$. From (j) and (b), using Remark 3.14, we have (m) $\Delta^a_\sigma = \Delta^a_{\sigma_1}$. From (i), (b), and (l), we know that (n) $\delta' = (\emptyset, \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma))$. From (k), (d), (e), (m), and (n), we have (o) $\langle c, \sigma_1\rangle \underset{\delta}{\longrightarrow} \langle c', \sigma_1''\rangle$. From (i) and (o), we conclude
  - We have (d) $c = \mathbf{S}[\,\mathbf{skip}; c''\,]$, (e) $c' = \mathbf{S}[\,c''\,]$, (f) $\delta = \mathit{emp}$, and (g) $\sigma = \sigma'$. We instantiate the goal with σ_1, and remains to show that $\sigma' = \sigma_1 \uplus \sigma_2$ and

$\langle c, \sigma_1 \rangle \xrightarrow[\delta]{} \langle c', \sigma_1 \rangle$. From (g) and (b), we have (h) $\sigma' = \sigma_1 \uplus \sigma_2$. From the semantics, we have (i) $\langle \mathbf{S}[\,\mathbf{skip}; c''\,], \sigma_1 \rangle \xrightarrow[emp]{} \langle \mathbf{S}[\,c''\,], \sigma_1 \rangle$. From (i), (d), (e), and (f), we have (j) $\langle c, \sigma_1 \rangle \xrightarrow[\delta]{} \langle c', \sigma_1 \rangle$. From (h) and (j), we conclude

- We have (d) $c = \mathbf{S}[\,\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\,]$, (e) $c' = \mathbf{S}[\,c_1\,]$, (f) $\delta = \Delta^b_\sigma$, (g) $\sigma = \sigma'$, and (h) $[\![b]\!]_\sigma = \mathit{true}$. We instantiate the goal with σ_1, and remains to show that $\sigma' = \sigma_1 \uplus \sigma_2$ and $\langle c, \sigma_1 \rangle \xrightarrow[\delta]{} \langle c', \sigma_1 \rangle$. From (d), and (a), we know there exists z, such that (i) $[\![b]\!]_{\sigma_1} = z$. From (i) and (b), using Remark 3.12, we have (j) $[\![b]\!]_\sigma = z$. From (h), (j), and (i), we have (k) $[\![b]\!]_{\sigma_1} = \mathit{true}$. From (i) and (b), using Remark 3.15, we have (l) $\Delta^b_\sigma = \Delta^b_{\sigma_1}$. From (g) and (b), we have (m) $\sigma' = \sigma_1 \uplus \sigma_2$. From the semantics, and (k), we have (n) $\langle \mathbf{S}[\,\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\,], \sigma_1 \rangle \xrightarrow[\Delta^b_{\sigma_1}]{} \langle \mathbf{S}[\,c_1\,], \sigma_1 \rangle$. From (n), (d), (e), (f), and (l), we have (o) $\langle c, \sigma_1 \rangle \xrightarrow[\delta]{} \langle c', \sigma_1 \rangle$. From (m) and (o), we conclude

- We have (d) $c = \mathbf{S}[\,\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\,]$, (e) $c' = \mathbf{S}[\,c_2\,]$, (f) $\delta = \Delta^b_\sigma$, (g) $\sigma = \sigma'$, and (h) $[\![b]\!]_\sigma = \mathit{false}$. We instantiate the goal with σ_1, and remains to show that $\sigma' = \sigma_1 \uplus \sigma_2$ and $\langle c, \sigma_1 \rangle \xrightarrow[\delta]{} \langle c', \sigma_1 \rangle$. From (d), and (a), we know there exists z, such that (i) $[\![b]\!]_{\sigma_1} = z$. From (i) and (b), using Remark 3.12, we have (j) $[\![b]\!]_\sigma = z$. From (h), (j), and (i), we have (k) $[\![b]\!]_{\sigma_1} = \mathit{false}$. From (i) and (b), using Remark 3.15, we have (l) $\Delta^b_\sigma = \Delta^b_{\sigma_1}$. From (g) and (b), we have (m) $\sigma' = \sigma_1 \uplus \sigma_2$. From the semantics, and (k), we have (n) $\langle \mathbf{S}[\,\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\,], \sigma_1 \rangle \xrightarrow[\Delta^b_{\sigma_1}]{} \langle \mathbf{S}[\,c_2\,], \sigma_1 \rangle$. From (n), (d), (e), (f), and (l), we have (o) $\langle c, \sigma_1 \rangle \xrightarrow[\delta]{} \langle c', \sigma_1 \rangle$. From (m) and (o), we conclude

- We have (d) $c = \mathbf{S}[\,\mathbf{while}\ b\ \mathbf{do}\ c''\,]$, (e) $c' = \mathbf{S}[\,\mathbf{if}\ b\ \mathbf{then} (c''; \mathbf{while}\ b\ \mathbf{do}\ c'') \mathbf{else\ skip}\,]$, (f) $\delta = emp$, and (g) $\sigma = \sigma'$. We instantiate the goal with σ_1, and remains to show that $\sigma' = \sigma_1 \uplus \sigma_2$ and $\langle c, \sigma_1 \rangle \xrightarrow[\delta]{} \langle c', \sigma_1 \rangle$. From (g) and (b), we have (h) $\sigma' = \sigma_1 \uplus \sigma_2$. From the semantics, we have (i) $\langle \mathbf{S}[\,\mathbf{while}\ b\ \mathbf{do}\ c''\,], \sigma_1 \rangle \xrightarrow[emp]{} \langle \mathbf{S}[\,\mathbf{if}\ b\ \mathbf{then}\ (c''; \mathbf{while}\ b\ \mathbf{do}\ c'')\ \mathbf{else\ skip}\,], \sigma_1 \rangle$. From (i), (d), (e), and (f), we have (j) $\langle c, \sigma_1 \rangle \xrightarrow[\delta]{} \langle c', \sigma_1 \rangle$. From (h) and (j), we conclude $\square$

3.7 Interleaved Semantics

Fig. 3.11 defines the interleaving semantics of concurrent programs. This is also a contextual semantics, where $\mathbf{T}$ is used to specify a given thread.

$$
\begin{array}{rcll}
\langle \mathbf{T}[\,c\,], \sigma\rangle & \longmapsto & \mathsf{abort} & \text{if } \langle c, \sigma\rangle \longrightarrow \mathsf{abort} \\
\langle \mathbf{T}[\,c\,], \sigma\rangle & \longmapsto & \langle \mathbf{T}[\,c'\,], \sigma'\rangle & \text{if } \langle c, \sigma\rangle \longrightarrow \langle c', \sigma'\rangle \\
\langle \mathbf{T}[\,\mathbf{S}[\,c_1 \parallel c_2\,]\,], \sigma\rangle & \longmapsto & \langle \mathbf{T}[\langle\!\langle c_1, c_2\rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,])\,], \sigma\rangle & \text{always} \\
\langle \mathbf{T}[\langle\!\langle \mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathbf{p}} c\,], \sigma\rangle & \longmapsto & \langle \mathbf{T}[\,c\,], \sigma\rangle & \text{always} \\
\langle \mathbf{T}[\,\mathbf{S}[\,\mathbf{atomic}\; c\,]\,], \sigma\rangle & \longmapsto & \langle \mathbf{T}[\langle\!\langle c\rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\mathbf{skip}\,])\,], \sigma\rangle & \text{if } \mathbf{T} \text{ is } 0\text{-atomic} \\
\langle \mathbf{T}[\langle\!\langle \mathbf{skip}\rangle\!\rangle_{\mathbf{a}} c\,], \sigma\rangle & \longmapsto & \langle \mathbf{T}[\,c\,], \sigma\rangle & \text{always} \\
\langle \mathbf{T}[\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,]\rangle\!\rangle_{\mathbf{p}} c\,], \sigma\rangle & \longmapsto & \mathsf{race} & \text{if } \exists \delta_1, \delta_2, c_1', \sigma', \kappa. \\
 & & & \quad \langle c_1, \sigma\rangle \xrightarrow[\delta_1]{} \langle c_1', \sigma'\rangle \wedge \\
 & & & \quad \langle c_2, \sigma'\rangle \xrightarrow[\delta_2]{} \kappa \wedge \delta_1 \not\smile \delta_2 \\
\langle \mathbf{T}[\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,]\rangle\!\rangle_{\mathbf{p}} c\,], \sigma\rangle & \longmapsto & \mathsf{race} & \text{if } \exists \delta_1, \delta_2, c_2', \sigma', \kappa. \\
 & & & \quad \langle c_2, \sigma\rangle \xrightarrow[\delta_2]{} \langle c_2', \sigma'\rangle \wedge \\
 & & & \quad \langle c_1, \sigma'\rangle \xrightarrow[\delta_1]{} \kappa \wedge \delta_2 \not\smile \delta_1
\end{array}
$$

Figure 3.11: Interleaved semantics

The first two rules, define the a single step execution of a thread performing a sequential command. It follows the sequential semantics for the thread, that for the current state σ either aborts or finds a subsequent state σ'. The single thread stepping is then reflected in the interleaving semantics in a straightforward manner.

The next four rules are structural rules that syntactically modify the thread tree to reflect the following operations, respectively: thread fork, thread join, atomic begin, and atomic end. Despite of the heavy notation, these operations should be simple to follow. A special mention should be done for the atomic begin operation. For the atomic begin rule, we require a 0-atomic context differently from all other rules. This is due to the fact that in our semantics atomic blocks execute in small steps, and we would like to prevent an atomic block from starting if there is already a concurrent atomic block executing.

This ensures that atomic blocks are mutually exclusive in our interleaved semantics. And since atomic blocks can interleave with non-atomic operations, it resembles the semantics of *weak atomicity* [38]. Note also that we do not require any syntactic constraints to the nesting of atomic blocks and parallel compositions. They can nest just like any other sequential command of the language.

Remark 3.17. If T is 0- or 1-atomic, and $\langle T, \sigma \rangle \longmapsto^* \langle T', \sigma' \rangle$, then T' is 0- or 1-atomic

The last two rules of Fig. 3.11 are the ones that make this semantics peculiar or interesting. These two rules are symmetric, so we will only explain the first one. It defines a race condition by checking if there is any interference from the stepping of a thread c_1 to the subsequent step of another thread c_2, i.e. c_1 writes to at least one location read by c_2. This is considered a race condition as the execution of c_2 is likely to depend on the scheduling of c_1 prior to it. The program steps into a race configuration if such condition is established. Note, however, that the race detection in this semantics is non-deterministic. This means that even if there is a race condition according to the race rule, the program might still step using one of the other rules, ignoring this condition. Although this might seem unintuitive, this is very similar to program safety. An unsafe program does not necessarily aborts because it depends on the non-determinism of the scheduling. Nevertheless, we can define the safety of a given configuration κ simply as $\neg\kappa \longmapsto^* \mathsf{abort}$. In the same way, this semantics allows us to define the race freedom of a given configuration κ simply as $\neg\kappa \longmapsto^* \mathsf{race}$.

Another remark we would like to make is regarding the race conditions, as defined by the semantics, and memory allocation and deallocation. In order to avoid stepping into a race configuration, a program has to prevent concurrent allocation and deallocation. This is because an allocation action interleaved after a deallocation action might reuse the memory. Following this semantics, they will have overlapping write sets and be considered a race (it is curious, but this cannot happen if we consider two concurrent allocations or two concurrent deallocations). Therefore, to avoid that, a program has to enforce mutual exclusion between concurrent allocation and deallocation. This can be done by proper synchronization of the program. Or by using a simple and straightforward pattern: always wrap allocation and deallocation inside an atomic block. We will consider this last approach as our solution to avoid race conditions between memory allocation and deallocation. It is not enforced by the dynamic semantics of Fig. 3.11, but it will be enforced syntactically by the static semantics presented further in the subsequent chapters. We take this approach because we know that these memory management operations are at a higher level of abstraction, and, in fact, their implementation typically use some sort of synchronization to handle the reuse of memory. Real implementations have a bounded free list and more deterministic behavior. Our language is powerful enough to allow implementing such allocators with relying on the built-in constructs, if this becomes an issue.

Example 3.18. Here some examples of execution according to the interleaved semantics:

- $\langle (\mathtt{x} := \mathtt{y} \parallel \mathtt{y} := \mathtt{x}), \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 2\}\rangle \longmapsto \langle \langle\!\langle \mathtt{x} := \mathtt{y}, \mathtt{y} := \mathtt{x} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 2\}\rangle$
- $\langle \langle\!\langle \mathtt{x} := \mathtt{y}, \mathtt{y} := \mathtt{x} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 2\}\rangle \longmapsto \langle \langle\!\langle \mathbf{skip}, \mathtt{y} := \mathtt{x} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 2, \mathtt{y} \rightsquigarrow 2\}\rangle$
- $\langle \langle\!\langle \mathtt{x} := \mathtt{y}, \mathtt{y} := \mathtt{x} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 2\}\rangle \longmapsto \langle \langle\!\langle \mathtt{x} := \mathtt{y}, \mathbf{skip} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 1\}\rangle$
- $\langle \langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 1\}\rangle \longmapsto \langle \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 1\}\rangle$
- $\langle \langle\!\langle \mathtt{x} := \mathtt{y}, \mathtt{y} := \mathtt{x} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 2\}\rangle \longmapsto \mathsf{race}$
- $\langle \langle\!\langle \mathbf{atomic}\ \mathtt{x} := \mathtt{y}, \mathtt{y} := \mathtt{x} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 2\}\rangle \longmapsto$
  $\langle \langle\!\langle \langle\!\langle \mathtt{x} := \mathtt{y} \rangle\!\rangle_{\mathrm{a}} \mathbf{skip}, \mathtt{y} := \mathtt{x} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 2\}\rangle$
- $\langle \langle\!\langle \langle\!\langle \mathtt{x} := \mathtt{y} \rangle\!\rangle_{\mathrm{a}} \mathbf{skip}, \mathtt{y} := \mathtt{x} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 2\}\rangle \longmapsto$
  $\langle \langle\!\langle \langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathrm{a}} \mathbf{skip}, \mathtt{y} := \mathtt{x} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 2, \mathtt{y} \rightsquigarrow 2\}\rangle$
- $\langle \langle\!\langle \langle\!\langle \mathtt{x} := \mathtt{y} \rangle\!\rangle_{\mathrm{a}} \mathbf{skip}, \mathtt{y} := \mathtt{x} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 2\}\rangle \longmapsto$
  $\langle \langle\!\langle \langle\!\langle \mathtt{x} := \mathtt{y} \rangle\!\rangle_{\mathrm{a}} \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 1\}\rangle$

- $\langle\langle\!\langle\langle\!\langle \mathtt{x} := \mathtt{y}\rangle\!\rangle_{\mathbf{a}}\mathbf{skip}, \mathtt{y} := \mathtt{x}\rangle\!\rangle_{\mathbf{p}}\mathbf{skip}, \{\mathtt{x} \rightsquigarrow 1, \mathtt{y} \rightsquigarrow 2\}\rangle \longmapsto \mathsf{race}$

Following the interleaved semantics, we can define the divergence of a concurrent program.

Definition 3.19. The divergence of a configuration κ, according to the interleaved semantics, is defined as follows: $\kappa \longmapsto^{\infty} \iff \forall n.\ \exists \kappa'.\ \kappa \longmapsto^{n} \kappa'$

$$
\begin{array}{rcll}
\langle \mathbf{T}[\,c\,], \sigma\rangle & \underset{\delta}{\longmapsto} & \mathsf{abort} & \text{if } \langle c, \sigma\rangle \underset{\delta}{\longrightarrow} \mathsf{abort} \\
\langle \mathbf{T}[\,c\,], \sigma\rangle & \underset{\delta}{\longmapsto} & \langle \mathbf{T}[\,c'\,], \sigma'\rangle & \text{if } \langle c, \sigma\rangle \underset{\delta}{\longrightarrow} \langle c', \sigma'\rangle \\
\langle \mathbf{T}[\,\mathbf{S}[\,c_1 \,\|\, c_2\,]\,], \sigma\rangle & \underset{emp}{\longmapsto} & \langle \mathbf{T}[\,\langle\!\langle c_1, c_2\rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,])\,], \sigma\rangle & \text{always} \\
\langle \mathbf{T}[\,\langle\!\langle \mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathbf{p}} c\,], \sigma\rangle & \underset{emp}{\longmapsto} & \langle \mathbf{T}[\,c\,], \sigma\rangle & \text{always} \\
\langle \mathbf{T}[\,\mathbf{S}[\,\mathbf{atomic}\ c\,]\,], \sigma\rangle & \underset{emp}{\longmapsto} & \langle \mathbf{T}[\,\langle\!\langle c\rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\mathbf{skip}\,])\,], \sigma\rangle & \text{if } \mathbf{T} \text{ is 0-atomic} \\
\langle \mathbf{T}[\,\langle\!\langle \mathbf{skip}\rangle\!\rangle_{\mathbf{a}} c\,], \sigma\rangle & \underset{emp}{\longmapsto} & \langle \mathbf{T}[\,c\,], \sigma\rangle & \text{always} \\
\langle \mathbf{T}[\,\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,]\rangle\!\rangle_{\mathbf{p}} c\,], \sigma\rangle & \underset{\delta_1 \cup \delta_2}{\longmapsto} & \mathsf{race} & \text{if } \exists c_1', \sigma', \kappa. \\
 & & & \quad \langle c_1, \sigma\rangle \underset{\delta_1}{\longrightarrow} \langle c_1', \sigma'\rangle \wedge \\
 & & & \quad \langle c_2, \sigma'\rangle \underset{\delta_2}{\longrightarrow} \kappa \wedge \delta_1 \not\smile \delta_2 \\
\langle \mathbf{T}[\,\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,]\rangle\!\rangle_{\mathbf{p}} c\,], \sigma\rangle & \underset{\delta_1 \cup \delta_2}{\longmapsto} & \mathsf{race} & \text{if } \exists c_2', \sigma', \kappa. \\
 & & & \quad \langle c_2, \sigma\rangle \underset{\delta_2}{\longrightarrow} \langle c_2', \sigma'\rangle \wedge \\
 & & & \quad \langle c_1, \sigma'\rangle \underset{\delta_1}{\longrightarrow} \kappa \wedge \delta_2 \not\smile \delta_1
\end{array}
$$

Figure 3.12: Interleaved semantics with footprints

In Fig. 3.12 we present a interleaved semantics equivalent to the one shown in Fig. 3.11 adorned with footprints. Naturally, for each rule in Fig. 3.11 there is a similar corresponding rule in Fig. 3.12.

Following the adorned interleaved semantics, we can define a multi-step by taking the union of the footprints of each individual step.

Definition 3.20. A interleaved multi-step with footprint $\underset{\delta}{\longmapsto}^n$ is defined as:

$$\frac{}{\kappa \underset{emp}{\longmapsto}^0 \kappa} \qquad \frac{\kappa \underset{\delta_1}{\longmapsto} \kappa'' \quad \kappa'' \underset{\delta_2}{\longmapsto}^n \kappa'}{\kappa \underset{\delta_1 \cup \delta_2}{\longmapsto}^{n+1} \kappa'}$$

The reflexive transitive closure is defined as: $\kappa \underset{\delta}{\longmapsto}^* \kappa' \iff \exists n.\ \kappa \underset{\delta}{\longmapsto}^n \kappa'$.

Now, it should be trivial to establish the equivalence between the plain interleaved semantics of Fig. 3.11 and the adorned interleaved semantics of Fig. 3.12.

Remark 3.21. The following holds trivially:

1. $\kappa \longmapsto \kappa' \iff \exists \delta.\ \kappa \underset{\delta}{\longmapsto} \kappa'$

2. $\kappa \longmapsto^n \kappa' \iff \exists \delta.\ \kappa \underset{\delta}{\longmapsto}^n \kappa'$

3. $\kappa \longmapsto^* \kappa' \iff \exists \delta.\ \kappa \underset{\delta}{\longmapsto}^* \kappa'$

Framing properties. Here are the framing properties of our interleaved semantics, which basically show that any safe code will work the exact same way in a larger context. Note that a safe program that is race free will also be race free in a larger context.

Lemma 3.22. If $\neg\langle T, \sigma_1\rangle \longmapsto \textsf{abort}$, then for all σ and σ_2, such that $\sigma = \sigma_1 \uplus \sigma_2$, we have:

1. $\neg\langle T, \sigma\rangle \longmapsto \textsf{abort}$

2. If $\neg\langle T, \sigma_1\rangle \longmapsto \textsf{race}$, then $\neg\langle T, \sigma\rangle \longmapsto \textsf{race}$

3. If $\langle T, \sigma\rangle \underset{\delta}{\longmapsto} \langle T', \sigma'\rangle$, then exists σ_1' such that $\sigma' = \sigma_1' \uplus \sigma_2$ and $\langle T, \sigma_1\rangle \underset{\delta}{\longmapsto} \langle T', \sigma_1'\rangle$

Proof. Assuming (a) $\neg\langle T, \sigma_1\rangle \longmapsto \textsf{abort}$, and (b) $\sigma = \sigma_1 \uplus \sigma_2$, we have 3 cases:

- We need to show that (c) $\langle T, \sigma\rangle \longmapsto \textsf{abort}$ is false. We will assume (c) and reach a contradiction. From (c), and the semantics, we have (d) $T = \mathbf{T}[\,c\,]$ and (e) $\langle c, \sigma\rangle \longrightarrow \textsf{abort}$. From (e), using Lemma 3.16 (item 1), we know (f) $\langle c, \sigma_1\rangle \longrightarrow \textsf{abort}$. From (d), (f), and the semantics, we have (g) $\langle T, \sigma_1\rangle \longrightarrow \textsf{abort}$. We observe that (g) contradicts (a), and conclude

- We assume (c) $\neg\langle T, \sigma_1\rangle \longmapsto \textsf{race}$, and we need to show that (d) $\langle T, \sigma\rangle \longmapsto \textsf{race}$ is false. We will assume (d) and reach a contradiction. From (d), and the semantics, we have 2 cases:
  - We have (e) $T = \mathbf{T}[\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,]\rangle\!\rangle_{\mathrm{p}} c\,]$, (f) $\langle c_1, \sigma\rangle \underset{\delta_1}{\longrightarrow} \langle c_1', \sigma'\rangle$, (g) $\langle c_2, \sigma'\rangle \underset{\delta_2}{\longrightarrow} \kappa$, and (h) $\delta_1 \mathrel{\not\overset{\rightarrow}{\smile}} \delta_2$. From (e), (a), and the semantics, we know (i) $\neg\langle c_1, \sigma_1\rangle \longmapsto \textsf{abort}$. From (i) and (f), using Lemma 3.16 (item 2), we know there exists σ_1' such that (j) $\sigma' = \sigma_1' \uplus \sigma_2$ and (k) $\langle c_1, \sigma_1\rangle \underset{\delta_1}{\longrightarrow} \langle c_1', \sigma_1'\rangle$. We then have 2 cases:
    - $*$ We consider (l) $\neg\langle c_2, \sigma_1'\rangle \longrightarrow \textsf{abort}$. From (l), and Lemma 3.16 (item 1), we know (m) $\neg\langle c_2, \sigma'\rangle \longrightarrow \textsf{abort}$. From (m), and (g), we know (n) $\kappa = \langle c_2', \sigma''\rangle$. From (l), and (g), using Lemma 3.16 (item 2), we know there exists σ_1'' such that (o) $\sigma'' = \sigma_1'' \uplus \sigma_2$ and (p) $\langle c_2, \sigma_1'\rangle \underset{\delta_2}{\longrightarrow} \langle c_2', \sigma_1''\rangle$. From (e), (k), (p), (h), and the semantics, we know (q) $\langle T, \sigma_1\rangle \longmapsto \textsf{race}$. We observe that (q) contradicts (c), and conclude
    - $*$ We consider (l) $\langle c_2, \sigma_1'\rangle \longrightarrow \textsf{abort}$. From (l), we know there exists δ_2' such that (m) $\langle c_2, \sigma_1'\rangle \underset{\delta_2'}{\longrightarrow} \textsf{abort}$. We then have 2 cases:
      1. We consider (n) $\delta_1 \mathrel{\not\overset{\rightarrow}{\smile}} \delta_2'$. From (e), (k), (m), (n), and the semantics, we know (o) $\langle T, \sigma_1\rangle \longmapsto \textsf{race}$. We observe that (o) contradicts (c), and conclude
      2. We consider (n) $\delta_1 \overset{\rightarrow}{\smile} \delta_2'$. From (k), (m), and (n), it is not hard to see that (o) $\langle c_2, \sigma_1\rangle \underset{\delta_2'}{\longrightarrow} \textsf{abort}$. From (e), (o), and the semantics, we know (p) $\langle T, \sigma_1\rangle \longmapsto \textsf{abort}$. We observe that (p) contradicts (a), and conclude
  - We have (e) $T = \mathbf{T}[\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,]\rangle\!\rangle_{\mathrm{p}} c\,]$, (f) $\langle c_2, \sigma\rangle \underset{\delta_2}{\longrightarrow} \langle c_2', \sigma'\rangle$, (g) $\langle c_1, \sigma'\rangle \underset{\delta_1}{\longrightarrow} \kappa$, and (h) $\delta_2 \mathrel{\not\overset{\rightarrow}{\smile}} \delta_1$. The proof is symmetric to the previous case.

- We assume (c) $\langle T, \sigma\rangle \underset{\delta}{\longmapsto} \langle T', \sigma'\rangle$, and we need to show that then exists σ_1' such that $\sigma' = \sigma_1' \uplus \sigma_2$ and $\langle T, \sigma_1\rangle \underset{\delta}{\longmapsto} \langle T', \sigma_1'\rangle$. From (c), and the semantics, we have 5 cases:
  - We consider (d) $T = \mathbf{T}[\,c\,]$, (e) $T' = \mathbf{T}[\,c'\,]$, and (f) $\langle c, \sigma\rangle \underset{\delta}{\longrightarrow} \langle c', \sigma'\rangle$. From

(d), (a), and the semantics, we know (g) $\neg\langle c, \sigma_1\rangle \longrightarrow \textsf{abort}$. From (g) and (f), using Lemma 3.16 (item 2), we know there exists σ_1' such that (h) $\sigma' = \sigma_1' \uplus \sigma_2$ and (i) $\langle c, \sigma_1\rangle \xrightarrow[\delta]{} \langle c', \sigma_1'\rangle$. From (d), (e), (i), and the semantics, we know (j) $\langle T, \sigma_1\rangle \overset{}{\underset{\delta}{\longmapsto}} \langle T', \sigma_1'\rangle$. Instantiating the goal with σ_1', from (h) and (j), we conclude

- We consider (d) $T = \mathbf{T}[\,\mathbf{S}[\,c_1 \parallel c_2\,]\,]$, (e) $T' = \mathbf{T}[\,\langle\!\langle c_1, c_2\rangle\!\rangle_{\mathbf{p}}\mathbf{S}[\,\mathbf{skip}\,]\,]$, (f) $\delta = emp$, and (g) $\sigma = \sigma'$. From (d), (e), and the semantics, we know (h) $\langle T, \sigma_1\rangle \underset{emp}{\longmapsto} \langle T', \sigma_1\rangle$. Instantiating the goal with σ_1, from (b) and (h), we conclude
- We consider (d) $T = \mathbf{T}[\,\langle\!\langle \mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathbf{p}} c\,]$, (e) $T' = \mathbf{T}[\,c\,]$, (f) $\delta = emp$, and (g) $\sigma = \sigma'$. From (d), (e), and the semantics, we know (h) $\langle T, \sigma_1\rangle \underset{emp}{\longmapsto} \langle T', \sigma_1\rangle$. Instantiating the goal with σ_1, from (b) and (h), we conclude
- We consider (d) $T = \mathbf{T}[\,\mathbf{S}[\,\mathbf{atomic}\ c\,]\,]$, (e) $T' = \mathbf{T}[\,\langle\!\langle c\rangle\!\rangle_{\mathbf{a}}\mathbf{S}[\,\mathbf{skip}\,]\,]$, (f) $\delta = emp$, (g) $\sigma = \sigma'$, and (h) $\mathbf{T}$ is 0-atomic. From (d), (e), (h), and the semantics, we know (h) $\langle T, \sigma_1\rangle \underset{emp}{\longmapsto} \langle T', \sigma_1\rangle$. Instantiating the goal with σ_1, from (b) and (h), we conclude
- We consider (d) $T = \mathbf{T}[\,\langle\!\langle \mathbf{skip}\rangle\!\rangle_{\mathbf{a}} c\,]$, (e) $T' = \mathbf{T}[\,c\,]$, (f) $\delta = emp$, and (g) $\sigma = \sigma'$. From (d), (e), and the semantics, we know (h) $\langle T, \sigma_1\rangle \underset{emp}{\longmapsto} \langle T', \sigma_1\rangle$. Instantiating the goal with σ_1, from (b) and (h), we conclude □

3.8 Parameterized Semantics

In this section, we present our parameterized operational semantics. This semantics allows transformations to the program text to occur during execution. The actual transformations that happen are defined by a parameter of the semantics, that is why we refer to it as *parameterized*.

$$[\Lambda]\ \langle T, \sigma\rangle \longmapsto \kappa \qquad \text{if } \exists T'.\ (T, T') \in \lfloor\Lambda\rfloor \wedge \langle T', \sigma\rangle \longmapsto \kappa$$

Figure 3.13: Parameterized semantics

Figure 3.13 shows the parameterized semantics as a single rule. The stepping relation takes Λ as a parameter, which is a binary relation between commands

$$\Lambda \subseteq \mathit{Command} \times \mathit{Command}$$

however it is lifted to thread trees according to the following definition:

Definition 3.23. The binary relation between thread trees $\lfloor\Lambda\rfloor$ is defined as follows:

$$\frac{(c_1, c_2) \in \Lambda}{(c_1, c_2) \in \lfloor\Lambda\rfloor} \qquad \frac{(T_1, T_2) \in \lfloor\Lambda\rfloor \quad (T_1', T_2') \in \lfloor\Lambda\rfloor \quad (c_1, c_2) \in \Lambda}{(\langle\!\langle T_1, T_1' \rangle\!\rangle_{\mathbf{p}} c_1, \langle\!\langle T_2, T_2' \rangle\!\rangle_{\mathbf{p}} c_2) \in \lfloor\Lambda\rfloor} \qquad \frac{(T_1, T_2) \in \lfloor\Lambda\rfloor \quad (c_1, c_2) \in \Lambda}{(\langle\!\langle T_1 \rangle\!\rangle_{\mathbf{a}} c_1, \langle\!\langle T_2 \rangle\!\rangle_{\mathbf{a}} c_2) \in \lfloor\Lambda\rfloor}$$

The semantics follows the interleaved semantics presented in Fig. 3.11, except that at any given step, the current thread tree can be replaced by another thread tree related through the $\lfloor\Lambda\rfloor$ relation. $\lfloor\Lambda\rfloor$ allows replacing leaf and continuation commands of thread trees through the Λ relation. Λ is supposed to provide a set of commands that are related to the given command using some notion of equivalence. This Λ-based semantics chooses nondeterministically which thread tree will be chosen from $\lfloor\Lambda\rfloor$. Therefore, in order to reason about this semantics, one needs to consider all possible thread trees related through a given instantiation of Λ.

Following the parameterized semantics, we can define both a multi-step and the divergence of a concurrent program.

Definition 3.24. A parameterized multi-step $[\Lambda] \quad \longmapsto^n$ is defined as:

$$\frac{}{[\Lambda]\ \kappa \longmapsto^0 \kappa} \qquad \frac{[\Lambda]\ \kappa \longmapsto \kappa'' \quad [\Lambda]\ \kappa'' \longmapsto^n \kappa'}{[\Lambda]\ \kappa \longmapsto^{n+1} \kappa'}$$

The reflexive transitive closure is defined as: $[\Lambda]\ \kappa \longmapsto^* \kappa' \iff \exists n.\ [\Lambda]\ \kappa \longmapsto^n \kappa'$.

Definition 3.25. The divergence of a configuration κ, according to the parameterized semantics, is defined as follows: $[\Lambda]\ \kappa \longmapsto^\infty \iff \forall n.\ \exists \kappa'.\ [\Lambda]\ \kappa \longmapsto^n \kappa'$

Naturally, different instantiations of Λ yield different semantics. As one can see, this semantics is trivially equivalent to the interleaving semantics shown in Fig. 3.11 once Λ is instantiated with an identity relation.

Remark 3.26. The following holds trivially:

1. $[=]\ \kappa \longmapsto \kappa' \iff \kappa \longmapsto \kappa'$

2. $[=]\ \kappa \longmapsto^n \kappa' \iff \kappa \longmapsto^n \kappa'$

3. $[=]\ \kappa \longmapsto^* \kappa' \iff \kappa \longmapsto^* \kappa'$

4. $[=]\ \kappa \longmapsto^\infty \iff \kappa \longmapsto^\infty$

A more interesting relation to be used as an instantiation of Λ is presented in the following section.

3.9 Command Subsumption

In this section, we define a *command subsumption* relation. We call it subsumption — instead of equivalence — because it is an asymmetric relation. This asymmetry comes from the fact that programs are non-deterministic. If the set of outcomes of program A is the same as the set of outcomes of program B, we would say they are equivalent, and one could be used in the place of the other. However, if the set of outcomes of program B is a subset of the outcomes of program A, we can no longer replace program B by program A, but certainly we can replace program A by program B. Thus, we can see program B as a specialized (or rewritten) version of program A. In our terminology, we say that program A subsumes program B, or equivalently we say that program B is subsumed by program A.

Our subsumption relation permits rewriting of non-synchronized portions of a program — those in-between atomic operations — freely, as long as it:

1. preserves their safety and termination properties;

2. preserves or specializes their sequential semantics;

3. does not increase their footprint.

The first two requirements are necessary to define subsumption even for sequential programs. The last requirement is necessary for concurrent programs. The intuition behind it is based on the fact that concurrent programs should be well-synchronized. In other words, accesses to shared memory should be performed through atomic operations; and non-synchronized operations should be restricted to only access thread-local or read-only memory. Therefore, the effect of a thread's non-synchronized code is not visible to other threads until the next atomic block is reached. The last requirement guarantees that this property is be preserved by subsumption.

In order to define subsumption, we need two auxiliary definitions — sequential evaluation (or big-step) and sequential divergence — as show in Fig. 3.14. A sequential evaluation $\kappa \Downarrow \kappa'$, starting from configuration κ, executes a finite number of sequential steps until configuration κ' is reached where no more sequential steps can be taken. A sequential divergence $\kappa \Uparrow$, happens when there is always a configuration reachable from κ after n-steps, for all possible values of n.

$$\begin{array}{ll} \kappa \Downarrow \kappa' & \text{if } \kappa \longrightarrow^{*} \kappa' \wedge \neg(\exists \kappa''.\ \kappa' \longrightarrow \kappa'') \\ \kappa \Uparrow & \text{if } \forall n.\ \exists \kappa'.\ \kappa \longrightarrow^{n} \kappa' \end{array}$$

Figure 3.14: Evaluation semantics

Example 3.27. Here are some examples of evaluation and divergence for the sequential semantics:

- $\langle \mathtt{x} := [\mathtt{x}]; \mathtt{x} := [\mathtt{x}], \{\mathtt{x} \rightsquigarrow 33, 33 \rightsquigarrow 44, 44 \rightsquigarrow 7\}\rangle \Downarrow \langle \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 7, 33 \rightsquigarrow 44, 44 \rightsquigarrow 7\}\rangle$

- $\langle \mathbf{while}\ \mathtt{x} < 10\ \mathbf{do}\ \mathtt{x} := \mathtt{x}+1, \{\mathtt{x} \rightsquigarrow 0\}\rangle \Downarrow \langle \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 10\}\rangle$

- $\langle \mathbf{while}\ \mathtt{x} < 10\ \mathbf{do}\ \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 0\}\rangle \Uparrow$

- $\langle \mathtt{x} := \mathbf{cons}(0); [\mathtt{x}+1] := 3, \{\mathtt{x} \rightsquigarrow 5, 33 \rightsquigarrow 0\}\rangle \Downarrow \langle \mathbf{skip}, \{\mathtt{x} \rightsquigarrow 32, 32 \rightsquigarrow 0, 33 \rightsquigarrow 3\}\rangle$

- $\langle \textbf{while}\ \mathtt{x} \neq 10\ \textbf{do}\ \mathtt{x} := \textbf{cons}(\mathtt{x}), \{\mathtt{x} \rightsquigarrow 0\}\rangle \Downarrow \langle \textbf{skip}, \{\mathtt{x} \rightsquigarrow 10, 10 \rightsquigarrow 44, 33 \rightsquigarrow 0, 44 \rightsquigarrow 33\}\rangle$

- $\langle \textbf{while}\ \mathtt{x} \neq 10\ \textbf{do}\ \mathtt{x} := \textbf{cons}(\mathtt{x}), \{\mathtt{x} \rightsquigarrow 0\}\rangle \Uparrow$

Example 3.28. Here some examples of sequential evaluation in the presence of non-sequential commands:

- $\langle \textbf{atomic skip}, \varnothing\rangle \Downarrow \langle \textbf{atomic skip}, \varnothing\rangle$

- $\langle \textbf{atomic skip}; \mathtt{x} := 3, \{\mathtt{x} \rightsquigarrow 0\}\rangle \Downarrow \langle \textbf{atomic skip}; \mathtt{x} := 3, \{\mathtt{x} \rightsquigarrow 0\}\rangle$

- $\langle \mathtt{x} := 3; \textbf{atomic skip}, \{\mathtt{x} \rightsquigarrow 0\}\rangle \Downarrow \langle \textbf{atomic skip}, \{\mathtt{x} \rightsquigarrow 3\}\rangle$

- $\langle \textbf{if}\ \mathtt{x} < 2\ \textbf{then}\ \mathtt{x} := 3\ \textbf{else}\ (\textbf{atomic}\ \mathtt{y} := \mathtt{x}), \{\mathtt{x} \rightsquigarrow 0\}\rangle \Downarrow \langle \textbf{skip}, \{\mathtt{x} \rightsquigarrow 3\}\rangle$

- $\langle \textbf{if}\ \mathtt{x} < 2\ \textbf{then}\ \mathtt{x} := 3\ \textbf{else}\ (\textbf{atomic}\ \mathtt{y} := \mathtt{x}), \{\mathtt{x} \rightsquigarrow 2\}\rangle \Downarrow \langle \textbf{atomic}\ \mathtt{y} := \mathtt{x}, \{\mathtt{x} \rightsquigarrow 2\}\rangle$

- $\langle \textbf{while}\ \mathtt{x} < 10\ \textbf{do}\ (\mathtt{x} := \mathtt{x}+1; \textbf{if}\ \mathtt{x} = 8\ \textbf{then}\ (\mathtt{x} := 5 \parallel \mathtt{x} := 6)), \{\mathtt{x} \rightsquigarrow 0\}\rangle \Downarrow$
  $\langle (\mathtt{x} := 5 \parallel \mathtt{x} := 6); \textbf{while}\ \mathtt{x} < 10\ \textbf{do}\ (\mathtt{x} := \mathtt{x}+1; \textbf{if}\ \mathtt{x} = 8\ \textbf{then}\ (\mathtt{x} := 5 \parallel \mathtt{x} := 6)), \{\mathtt{x} \rightsquigarrow 8\}\rangle$

Properties of evaluation. Given the definition of the sequential semantics (from Fig. 3.9), it is not hard to see that, after a non-aborting big-step, either we reached the end of the execution, or we reached a concurrent command.

Remark 3.29. If $\kappa \Downarrow \kappa'$, where $\kappa' = \langle c, \sigma\rangle$, then either:

1. $c = \textbf{skip}$;

2. or exists $\mathbf{S}$, c_1, and c_2, such that $c = \mathbf{S}[\, c_1 \parallel c_2 \,]$;

3. or exists $\mathbf{S}$, and c', such that $c = \mathbf{S}[\, \textbf{atomic}\ c' \,]$.

Furthermore, if we reached the end of execution or a concurrent command, then we know that any big-step will be trivially empty, and no divergence is possible.

Remark 3.30. For all κ, where $\kappa = \langle c, \sigma\rangle$, and c is $\textbf{skip}$, $\mathbf{S}[\, c_1 \parallel c_2 \,]$, or $\mathbf{S}[\, \textbf{atomic}\ c' \,]$, we have:

1. $\kappa \Downarrow \kappa$;

2. for all κ', if $\kappa \Downarrow \kappa'$, then $\kappa = \kappa'$;

3. not $\kappa \Uparrow$;

Trivially, from Remark 3.29 and Remark 3.30 we can establish that after a big-step, any subsequent big-step will be empty, and no divergence is possible.

Corollary 3.31. If $\kappa \Downarrow \kappa'$, then

1. $\kappa' \Downarrow \kappa'$;

2. for all κ'', if $\kappa' \Downarrow \kappa''$, then $\kappa' = \kappa''$;

3. not $\kappa' \Uparrow$;

We can also make the following remark regarding command evaluation, which means that if a command evaluates to itself then the final state must match the initial state; as a result of an empty evaluation.

Remark 3.32. If $\langle c, \sigma\rangle \Downarrow \langle c, \sigma'\rangle$, then $\sigma = \sigma'$

Now we are ready to define the command subsumption relation $c_1 \rightsquigarrow c_2$. Note that in order to have a well-founded inductive definition of subsumption, we had to use indexing. A coinductive definition could also be used, removing the need for indexing. However, we chose to avoid coinduction given that inductive principles are more common and far better understood.

Definition 3.33. $c_1 \rightsquigarrow_0 c_2$ always holds; $c_1 \rightsquigarrow_{n+1} c_2$ holds if, and only if, the following are true:

1. If $\langle c_2, \sigma\rangle \longrightarrow^* \mathsf{abort}$, then $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$;

2. If $\langle c_2, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$, then either $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$,

   or there exists c_1' such that $\langle c_1, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and the following constraints hold:

(a) if $c_2' = \textbf{skip}$, then $c_1' = \textbf{skip}$;

(b) if $c_2' = \mathbf{S}_2[\, c_2'' \,\|\, c_2''' \,]$, there exist $\mathbf{S}_1$, c_1'' and c_1''' such that

i. $c_1' = \mathbf{S}_1[\, c_1'' \,\|\, c_1''' \,]$;

ii. $c_1'' \leadsto_n c_2''$;

iii. $c_1''' \leadsto_n c_2'''$;

iv. $\mathbf{S}_1[\,\textbf{skip}\,] \leadsto_n \mathbf{S}_2[\,\textbf{skip}\,]$;

(c) if $c_2' = \mathbf{S}_2[\,\textbf{atomic}\ c_2''\,]$, there exist $\mathbf{S}_1$ and c_1'' such that

i. $c_1' = \mathbf{S}_1[\,\textbf{atomic}\ c_1''\,]$;

ii. $c_1'' \leadsto_n c_2''$;

iii. $\mathbf{S}_1[\,\textbf{skip}\,] \leadsto_n \mathbf{S}_2[\,\textbf{skip}\,]$;

3. If $\langle c_2, \sigma \rangle \Uparrow$, then either $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$, or $\langle c_1, \sigma \rangle \Uparrow$.

4. If $\langle c_2, \sigma \rangle \xrightarrow[\delta_2]{}^* \kappa_2$, then either $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$, or there exists δ_1 and κ_1 such that $\langle c_1, \sigma \rangle \xrightarrow[\delta_1]{}^* \kappa_1$ and $\delta_2 \subseteq \delta_1$.

We define $c_1 \leadsto c_2$ as $\forall n.\ c_1 \leadsto_n c_2$. We define $T_1 \leadsto_{\mathtt{t}} T_2$ as $(T_1, T_2) \in \lfloor \leadsto \rfloor$.

Informally, we say c_2 is subsumed by c_1 if for all input states — afters performing a sequential big step — c_2 aborts only if c_1 aborts; or, if c_2 completes, then either c_1 aborts or there is a sequential big step taken by c_1 that ends in the same state. Also, if c_2 completes the big step and the execution terminates or reaches a concurrent command, there must be a corresponding command at the end of the big step taken by c_1 and the remaining parts of c_2 and c_1 still satisfy the relation. We also enforce that if c_2 diverges, then either c_1 aborts or it diverges as well. Furthermore, as explained earlier, if c_2 executes a finite number of steps with a given footprint δ_2, then either c_1 aborts or there must also be a finite number of steps taken by c_1 with a footprint δ_1 equal or larger than δ_2. Note that for those cases where c_1 aborts, c_2 can have any arbitrary behavior.

Properties of subsumption. Here we present some important properties of subsumption. The following lemmas are useful if we view $c_1 \rightsquigarrow c_2$ as a transformation from c_1 to c_2. They aid in the composition of transformation proofs.

Lemmas 3.34 and 3.35 shows that both the identity transformation and the composition of multiple transformations — given that they obey subsumption — do not violate subsumption. It is useful when composing smaller transformations into large complex sequences of transformations.

Lemma 3.34. The relation $\rightsquigarrow$ is reflexive.

Proof. We need to show that for all n and c, $c \rightsquigarrow_n c$. We prove by induction over n. If $n = 0$, by Def. 3.33, we conclude. If $n > 0$, by Def. 3.33, we have 4 cases:

- If (a) $\langle c, \sigma \rangle \longrightarrow^* \textsf{abort}$, we need to show that $\langle c, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (a), we conclude
- If (a) $\langle c, \sigma \rangle \Downarrow \langle c', \sigma' \rangle$, we will show that there exists c'' such that $\langle c, \sigma \rangle \Downarrow \langle c'', \sigma' \rangle$ and constraints (2.a) through (2.c) of Def. 3.33 hold over c' and c''. Instantiating the goal with c', knowing (a), remains to show that:
  - If $c' = \textbf{skip}$, then $c' = \textbf{skip}$, we conclude trivially
  - If $c' = \mathbf{S}[\, c_1 \parallel c_2 \,]$, then exists $\mathbf{S}'$, c_1', and c_2', such that $c' = \mathbf{S}'[\, c_1' \parallel c_2' \,]$, $c_1' \rightsquigarrow_{n-1} c_1$, $c_2' \rightsquigarrow_{n-1} c_2$, and $\mathbf{S}'[\,\textbf{skip}\,] \rightsquigarrow_{n-1} \mathbf{S}[\,\textbf{skip}\,]$. By the induction hypothesis, we know that (b) $c_1 \rightsquigarrow_{k-1} c_1$, (c) $c_2 \rightsquigarrow_{k-1} c_2$, and (d) $\mathbf{S}[\,\textbf{skip}\,] \rightsquigarrow_{k-1} \mathbf{S}[\,\textbf{skip}\,]$. Instantiating the goal with $\mathbf{S}$, c_1, and c_2, from (b), (c), and (d), we conclude
  - If $c' = \mathbf{S}[\,\textbf{atomic}\ c''\,]$, then exists $\mathbf{S}'$ and c''', such that $c' = \mathbf{S}'[\,\textbf{atomic}\ c'''\,]$, $c''' \rightsquigarrow_{n-1} c''$, and $\mathbf{S}'[\,\textbf{skip}\,] \rightsquigarrow_{n-1} \mathbf{S}[\,\textbf{skip}\,]$. By the induction hypothesis, we know (b) $c'' \rightsquigarrow_{k-1} c''$ and (c) $\mathbf{S}[\,\textbf{skip}\,] \rightsquigarrow_{k-1} \mathbf{S}[\,\textbf{skip}\,]$. Instantiating the goal with $\mathbf{S}$ and c'', from (b), and (c), we conclude
- If (a) $\langle c, \sigma \rangle \Uparrow$, we will show that $\langle c, \sigma \rangle \Uparrow$. From (a), we conclude

- If (a) $\langle c, \sigma \rangle \xrightarrow[\delta]{}^{*} \kappa$, we will to show that there exists δ' and κ' such that $\langle c, \sigma \rangle \xrightarrow[\delta']{}^{*} \kappa'$ and $\delta \subseteq \delta'$. We know that (b) $\delta \subseteq \delta$. Instantiating the goal with δ and κ, from (a) and (b), we conclude $\square$

Lemma 3.35. The relation $\rightsquigarrow$ is transitive.

Proof. We need to show that for all n, c_1, c_2, and c_3, assuming (a) $c_1 \rightsquigarrow c_2$ and (b) $c_2 \rightsquigarrow c_3$, we have $c_1 \rightsquigarrow_n c_3$. From (a), by Def. 3.33, we know (c) $c_1 \rightsquigarrow_n c_2$. From (b), by Def. 3.33, we know (d) $c_2 \rightsquigarrow_n c_3$. We prove by induction over n, assuming (c) and (d). If $n = 0$, by Def. 3.33, we conclude. If $n > 0$, by Def. 3.33, we have 4 cases:

- If (e) $\langle c_3, \sigma \rangle \longrightarrow^{*} \textsf{abort}$, we need to show that $\langle c_1, \sigma \rangle \longrightarrow^{*} \textsf{abort}$. From (e) and (d), by Def. 3.33 (item 1), we know (f) $\langle c_2, \sigma \rangle \longrightarrow^{*} \textsf{abort}$. From (f) and (c), by Def. 3.33 (item 1), we conclude
- If (e) $\langle c_3, \sigma \rangle \Downarrow \langle c_3', \sigma' \rangle$, we need to show that either $\langle c_1, \sigma \rangle \longrightarrow^{*} \textsf{abort}$ or there exists c_1' such that $\langle c_1, \sigma \rangle \Downarrow \langle c_1', \sigma' \rangle$ and constraints (2.a) through (2.c) of Def. 3.33 hold over c_1' and c_3'. From (e) and (d), by Def. 3.33 (item 2), we have 2 cases:
  - If (f) $\langle c_2, \sigma \rangle \longrightarrow^{*} \textsf{abort}$, we will show that $\langle c_1, \sigma \rangle \longrightarrow^{*} \textsf{abort}$. From (f) and (c), by Def. 3.33 (item 1), we conclude
  - We have (f) $\langle c_2, \sigma \rangle \Downarrow \langle c_2', \sigma' \rangle$ and (g) constraints (2.a) through (2.c) of Def. 3.33 hold over c_2' and c_3'. From (f) and (c), by Def. 3.33 (item 2), we have 2 cases:
    - If $\langle c_1, \sigma \rangle \longrightarrow^{*} \textsf{abort}$, we conclude
    - We have (h) $\langle c_1, \sigma \rangle \Downarrow \langle c_1', \sigma' \rangle$ and (i) constraints (2.a) through (2.c) of Def. 3.33 hold over c_1' and c_2'. Instantiating the goal with c_1, knowing (h), remains to show that:
      - If (j) $c_3' = \textbf{skip}$, then $c_1' = \textbf{skip}$. From (j) and (g), we know (k) $c_2' = \textbf{skip}$. From (k) and (i), we conclude
      - If (j) $c_3' = \mathbf{S}_3[\, c_3'' \parallel c_3''' \,]$, then exists $\mathbf{S}_1$, c_1'', and c_1''', such that $c_1' = \mathbf{S}_1[\, c_1'' \parallel c_1''' \,]$, $c_1'' \rightsquigarrow_{n-1} c_3''$, $c_1''' \rightsquigarrow_{n-1} c_3'''$, and $\mathbf{S}_1[\, \textbf{skip} \,] \rightsquigarrow_{n-1} \mathbf{S}_3[\, \textbf{skip} \,]$. From

(j) and (g), we know there exists $\mathbf{S}_2$, c_2'', and c_2''', such that (k) $c_2' = \mathbf{S}_2[\, c_2'' \parallel c_2''' \,]$, (l) $c_2'' \leadsto_{n-1} c_3''$, (m) $c_2''' \leadsto_{n-1} c_3'''$, and (n) $\mathbf{S}_2[\,\mathbf{skip}\,] \leadsto_{n-1} \mathbf{S}_3[\,\mathbf{skip}\,]$. From (k) and (i), we know there exists $\mathbf{S}_1$, c_1'', and c_1''', such that (o) $c_1' = \mathbf{S}_1[\, c_1'' \parallel c_1''' \,]$, (p) $c_1'' \leadsto_{n-1} c_2''$, (q) $c_1''' \leadsto_{n-1} c_2'''$, and (r) $\mathbf{S}_1[\,\mathbf{skip}\,] \leadsto_{n-1} \mathbf{S}_2[\,\mathbf{skip}\,]$. From (p) and (l), by the induction hypothesis, we know that (s) $c_1'' \leadsto_{n-1} c_3''$. From (q) and (m), by the induction hypothesis, we know that (t) $c_1''' \leadsto_{n-1} c_3'''$. From (r) and (n), by the induction hypothesis, we know that (u) $\mathbf{S}_1[\,\mathbf{skip}\,] \leadsto_{n-1} \mathbf{S}_3[\,\mathbf{skip}\,]$. Instantiating the goal with $\mathbf{S}_1$, c_1'', and c_1''', from (o), (s), (t), and (u), we conclude

· If (j) $c_3' = \mathbf{S}_3[\mathbf{atomic}\ c_3'']$, then exists $\mathbf{S}_1$ and c_1'' such that $c_1' = \mathbf{S}_1[\mathbf{atomic}\ c_1'']$, $c_1'' \leadsto_{n-1} c_3''$, and $\mathbf{S}_1[\,\mathbf{skip}\,] \leadsto_{n-1} \mathbf{S}_3[\,\mathbf{skip}\,]$. From (j) and (g), we know there exists $\mathbf{S}_2$ and c_2'', such that (k) $c_2' = \mathbf{S}_2[\,\mathbf{atomic}\ c_2''\,]$, (l) $c_2'' \leadsto_{n-1} c_3''$, and (m) $\mathbf{S}_2[\,\mathbf{skip}\,] \leadsto_{n-1} \mathbf{S}_3[\,\mathbf{skip}\,]$. From (k) and (i), we know there exists $\mathbf{S}_1$ and c_1'', such that (n) $c_1' = \mathbf{S}_1[\,\mathbf{atomic}\ c_1''\,]$, (o) $c_1'' \leadsto_{n-1} c_2''$, and (p) $\mathbf{S}_1[\,\mathbf{skip}\,] \leadsto_{n-1} \mathbf{S}_2[\,\mathbf{skip}\,]$. From (o) and (l), by the induction hypothesis, we know that (q) $c_1'' \leadsto_{n-1} c_3''$. From (p) and (m), by the induction hypothesis, we know that (r) $\mathbf{S}_1[\,\mathbf{skip}\,] \leadsto_{n-1} \mathbf{S}_3[\,\mathbf{skip}\,]$. Instantiating the goal with $\mathbf{S}_1$ and c_1'', from (n), (q), and (r), we conclude

- If (e) $\langle c_3, \sigma \rangle \Uparrow$, we need to show that either $\langle c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$ or $\langle c_1, \sigma \rangle \Uparrow$. From (e) and (d), by Def. 3.33 (item 3), we have 2 cases:
  - We have (f) $\langle c_2, \sigma \rangle \longrightarrow^* \mathsf{abort}$, then we will show $\langle c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$. From (f) and (c), by Def. 3.33 (item 1), we conclude
  - We have (f) $\langle c_2, \sigma \rangle \Uparrow$. From (f) and (c), by Def. 3.33 (item 3), we conclude
- If (e) $\langle c_3, \sigma \rangle \underset{\delta_3}{\longrightarrow}^* \kappa_3$, we need to show that either $\langle c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$ or there exists δ_1 and κ_1 such that $\langle c_1, \sigma \rangle \underset{\delta_1}{\longrightarrow}^* \kappa_1$ and $\delta_3 \subseteq \delta_1$. From (e) and (d), by Def. 3.33 (item 4),

we have 2 cases:

- We have (f) $\langle c_2, \sigma \rangle \longrightarrow^* \textsf{abort}$, then we will show that $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (f) and (c), by Def. 3.33 (item 1), we conclude
- We have (f) $\langle c_2, \sigma \rangle \underset{\delta_2}{\longrightarrow^*} \kappa_2$ and (g) $\delta_3 \subseteq \delta_2$. From (f) and (c), by Def. 3.33 (item 4), we have 2 cases:
    * If $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$, we conclude
    * We have (h) $\langle c_1, \sigma \rangle \underset{\delta_1}{\longrightarrow^*} \kappa_1$ and (i) $\delta_2 \subseteq \delta_1$. From (g) and (i), we know that (j) $\delta_3 \subseteq \delta_1$. Instantiating the goal with δ_1 and κ_1, from (h) and (j), we conclude □

Lemma 3.36 ensures that local transformations that obey subsumption also hold in any larger context. This helps modular proofs of transformation. Note that $\mathcal{C}$ does not have to be an execution context **S**. It can be any context, i.e. a program with a hole in it.

Lemma 3.36. If $c_1 \rightsquigarrow c_2$, then, for all contexts $\mathcal{C}$, $\mathcal{C}[\, c_1 \,] \rightsquigarrow \mathcal{C}[\, c_2 \,]$.

Proof. By structural induction over $\mathcal{C}$, assuming (a) $c_1 \rightsquigarrow c_2$, we have 9 cases:

- We have (b) $\mathcal{C} = \bullet$. From (a), we conclude
- We have (b) $\mathcal{C} = (\mathcal{C}'; c)$. From (a), using the induction hypothesis, we have (c) $\mathcal{C}'[\, c_1 \,] \rightsquigarrow \mathcal{C}'[\, c_2 \,]$. From (c), using Lemma 3.38, we conclude
- We have (b) $\mathcal{C} = (c; \mathcal{C}')$. From (a), using the induction hypothesis, we have (c) $\mathcal{C}'[\, c_1 \,] \rightsquigarrow \mathcal{C}'[\, c_2 \,]$. From (c), using Lemma 3.39, we conclude
- We have (b) $\mathcal{C} = \textbf{if } b \textbf{ then } \mathcal{C}' \textbf{ else } c$. From (a), using the induction hypothesis, we have (c) $\mathcal{C}'[\, c_1 \,] \rightsquigarrow \mathcal{C}'[\, c_2 \,]$. From (c), using Lemma 3.40, we conclude
- We have (b) $\mathcal{C} = \textbf{if } b \textbf{ then } c \textbf{ else } \mathcal{C}'$. From (a), using the induction hypothesis, we have (c) $\mathcal{C}'[\, c_1 \,] \rightsquigarrow \mathcal{C}'[\, c_2 \,]$. From (c), using Lemma 3.41, we conclude

- We have (b) $\mathcal{C} = \textbf{while } b \textbf{ do } \mathcal{C}'$. From (a), using the induction hypothesis, we have (c) $\mathcal{C}'[\, c_1 \,] \rightsquigarrow \mathcal{C}'[\, c_2 \,]$. From (c), using Lemma 3.42, we conclude

- We have (b) $\mathcal{C} = (\mathcal{C}' \parallel c)$. From (a), using the induction hypothesis, we have (c) $\mathcal{C}'[\, c_1 \,] \rightsquigarrow \mathcal{C}'[\, c_2 \,]$. From (c), using Lemma 3.43, we conclude

- We have (b) $\mathcal{C} = (c \parallel \mathcal{C}')$. From (a), using the induction hypothesis, we have (c) $\mathcal{C}'[\, c_1 \,] \rightsquigarrow \mathcal{C}'[\, c_2 \,]$. From (c), using Lemma 3.44, we conclude

- We have (b) $\mathcal{C} = \textbf{atomic } \mathcal{C}'$. From (a), using the induction hypothesis, we have (c) $\mathcal{C}'[\, c_1 \,] \rightsquigarrow \mathcal{C}'[\, c_2 \,]$. From (c), using Lemma 3.45, we conclude □

The proof of Lemma 3.36 uses the following set of lemmas:

Lemma 3.37. If $c_1 \rightsquigarrow_{n_1} c_2$, and $n_2 \leq n_1$, then $c_1 \rightsquigarrow_{n_2} c_2$

Proof. By induction over n_1, assuming (a) $c_1 \rightsquigarrow_{n_1} c_2$ and (b) $n_2 \leq n_1$. If $n_1 = 0$, then $n_2 = 0$. By Def. 3.33, we conclude. If $n_1 > 0$, considering $n_2 > 0$, by Def. 3.33, we have 4 cases:

- If (a) $\langle c_2, \sigma \rangle \longrightarrow^* \mathsf{abort}$, we need to show that $\langle c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$. From (a), by Def. 3.33 (item 1), we conclude

- If (a) $\langle c_2, \sigma \rangle \Downarrow \langle c_2', \sigma' \rangle$, we need to show that either $\langle c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$ or there exists c_1' such that $\langle c_1, \sigma \rangle \Downarrow \langle c_1', \sigma' \rangle$ and constraints (2.a) through (2.c) of Def. 3.33 hold over c_1' and c_2' observing indexing using $n_2 - 1$. From (a), by Def. 3.33 (items 2.a through 2.c), using the induction hypothesis, we conclude

- If (a) $\langle c_2, \sigma \rangle \Uparrow$, we need to show that either $\langle c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$ or $\langle c_1, \sigma \rangle \Uparrow$. From (a), by Def. 3.33 (item 3), we conclude

- If (a) $\langle c_2, \sigma \rangle \underset{\delta_2}{\longrightarrow^*} \kappa_2$, we need to show that either $\langle c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$ or there exists δ_1 and κ_1 such that $\langle c_1, \sigma \rangle \underset{\delta_1}{\longrightarrow^*} \kappa_1$ and $\delta_2 \subseteq \delta_1$. From (a), by Def. 3.33 (item 4), we conclude □

Lemma 3.38. If $c_1 \rightsquigarrow c_2$, then $(c_1; c) \rightsquigarrow (c_2; c)$

Proof. From Def. 3.33, we need to show for all n, that $(c_1; c) \rightsquigarrow_n (c_2; c)$. By induction over n, if $n = 0$, by Def. 3.33, we conclude. If $n > 0$, assuming (a) $c_1 \rightsquigarrow c_2$, by Def. 3.33, we have 4 cases to consider:

- If (b) $\langle c_2; c, \sigma\rangle \longrightarrow^* \textsf{abort}$, we need to show that $\langle c_1; c, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (b), and the semantics, we have 2 cases:
  - We consider (c) $\langle c_2, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (a) and (c), by Def. 3.33 (item 1), we have (d) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (d), and the semantics, we know (e) $\langle c_1; c, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (e), we conclude
  - We consider that there exists σ' such that (c) $\langle c_2, \sigma\rangle \Downarrow \langle \textbf{skip}, \sigma'\rangle$ and (d) $\langle c, \sigma'\rangle \longrightarrow^* \textsf{abort}$. From (a) and (c), by Def. 3.33 (item 2.a), we have 2 cases:
    - We consider (e) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (e), and the semantics, we know (f) $\langle c_1; c, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (f), we conclude
    - We consider (e) $\langle c_1, \sigma\rangle \Downarrow \langle \textbf{skip}, \sigma'\rangle$. From (e), (d), and the semantics, we know (f) $\langle c_1; c, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (f), we conclude
- If (b) $\langle c_2; c, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$, we need to show that either $\langle c_1; c, \sigma\rangle \longrightarrow^* \textsf{abort}$ or exists c_1' such that $\langle c_1; c, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and c_1' and c_2' satisfy constraints 2.a to 2.c of Def. 3.33. From (b), and the semantics, we have 2 cases:
  - We consider (c) $\langle c_2, \sigma\rangle \Downarrow \langle c_2'', \sigma'\rangle$ and (d) $c_2' = (c_2''; c)$. From (a) and (c), by Def. 3.33 (item 2), we have 2 cases:
    - We consider (e) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (e), and the semantics, we have (f) $\langle c_1; c, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (f), we conclude
    - We consider (e) $\langle c_1, \sigma\rangle \Downarrow \langle c_1'', \sigma'\rangle$ and (f) constraints 2.b and 2.c of Def. 3.33 hold for c_1'' and c_2''. From (e), and the semantics, we have (g) $\langle c_1; c, \sigma\rangle \Downarrow$

$\langle c_1''; c, \sigma' \rangle$. From (f), using the induction hypothesis, we know (h) constraints 2.b and 2.c of Def. 3.33 hold for $c_1''; c$ and $c_2''; c$. Instantiating the goal with $c_1''; c$, from (g), (h), we conclude

  - We consider (c) $\langle c_2, \sigma \rangle \Downarrow \langle \mathbf{skip}, \sigma'' \rangle$ and (d) $\langle c, \sigma'' \rangle \Downarrow \langle c_2', \sigma' \rangle$. From (a) and (c), by Def. 3.33 (item 2), we have 2 cases:
    * We consider (e) $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (e), and the semantics, we have (f) $\langle c_1; c, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (f), we conclude
    * We consider (e) $\langle c_1, \sigma \rangle \Downarrow \langle \mathbf{skip}, \sigma'' \rangle$. From (e), (d), and the semantics, we have (f) $\langle c_1; c, \sigma \rangle \Downarrow \langle c_2', \sigma' \rangle$. Using Lemma 3.34, we know (g) constraints 2.a to 2.c of Def. 3.33 hold for c_2' and c_2'. Instantiating the goal with c_2', from (f) and (g), we conclude

- If (b) $\langle c_2; c, \sigma \rangle \Uparrow$, we need to show that either $\langle c_1; c, \sigma \rangle \longrightarrow^* \textsf{abort}$ or $\langle c_1; c, \sigma \rangle \Uparrow$. From (b), and the semantics, we have 2 cases:
  - We consider (c) $\langle c_2, \sigma \rangle \Uparrow$. From (a) and (c), by Def. 3.33 (item 3), we have 2 cases:
    * We consider (d) $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (d), and the semantics, we know (e) $\langle c_1; c, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (e), we conclude
    * We consider (d) $\langle c_1, \sigma \rangle \Uparrow$. From (d), and the semantics, we know (e) $\langle c_1; c, \sigma \rangle \Uparrow$. From (e), we conclude
  - We consider that there exists σ' such that (c) $\langle c_2, \sigma \rangle \Downarrow \langle \mathbf{skip}, \sigma' \rangle$ and (d) $\langle c, \sigma' \rangle \Uparrow$. From (a) and (c), by Def. 3.33 (item 2.a), we have 2 cases:
    * We consider (e) $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (e), and the semantics, we know (f) $\langle c_1; c, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (f), we conclude
    * We consider (e) $\langle c_1, \sigma \rangle \Downarrow \langle \mathbf{skip}, \sigma' \rangle$. From (e), (d), and the semantics, we know (f) $\langle c_1; c, \sigma \rangle \Uparrow$. From (f), we conclude

- If (b) $\langle c_2; c, \sigma\rangle \xrightarrow[\delta_2]{}^* \kappa_2$, we need to show that either $\langle c_1; c, \sigma\rangle \longrightarrow^* \mathsf{abort}$ or there exists δ_1 and κ_1 such that $\langle c_1; c, \sigma\rangle \xrightarrow[\delta_1]{}^* \kappa_1$ and $\delta_2 \subseteq \delta_1$. From (b), and the semantics, we have 2 cases:
  - We consider (c) $\langle c_2, \sigma\rangle \xrightarrow[\delta_2]{}^* \kappa_2'$. From (a) and (c), by Def. 3.33 (item 4), we have 2 cases:
    - We consider (d) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (d), and the semantics, we know (e) $\langle c_1; c, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (e), we conclude
    - We consider that there exists δ_1' and κ_1' such that (d) $\langle c_1, \sigma\rangle \xrightarrow[\delta_1']{}^* \kappa_1'$ and (e) $\delta_2 \subseteq \delta_1'$. From (d), and the semantics, we know that there exists κ_1'' such that (f) $\langle c_1; c, \sigma\rangle \xrightarrow[\delta_1']{}^* \kappa_1''$. Instantiating the goal with δ_1' and κ_1'', from (f) and (e), we conclude
  - We consider (c) $\langle c_2, \sigma\rangle \xrightarrow[\delta_2']{}^* \langle \mathbf{skip}, \sigma'\rangle$ and (d) $\langle c, \sigma'\rangle \xrightarrow[\delta_2'']{}^* \kappa_2'$, where (e) $\delta_2 = \delta_2' \cup \delta_2''$. From (a) and (c), by Def. 3.33 (item 2.a), we have 2 cases:
    - We consider (f) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (f), and the semantics, we know (g) $\langle c_1; c, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (g), we conclude
    - We consider (f) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma'\rangle$. From (f), (d), and the semantics, we know there exists δ_1' and κ_1' such that (g) $\langle c_1; c, \sigma\rangle \xrightarrow[\delta_1' \cup \delta_2'']{}^* \kappa_1'$ and (h) $\delta_2' \subseteq \delta_1'$. Instantiating the goal with $\delta_1' \cup \delta_2''$ and κ_1', from (g) and (h), we conclude □

Lemma 3.39. If $c_1 \rightsquigarrow c_2$, then $(c; c_1) \rightsquigarrow (c; c_2)$

Proof. From Def. 3.33, we need to show for all n, that $(c; c_1) \rightsquigarrow_n (c; c_2)$. By induction over n, if $n = 0$, by Def. 3.33, we conclude. If $n > 0$, assuming (a) $c_1 \rightsquigarrow c_2$, by Def. 3.33, we have 4 cases to consider:

- If (b) $\langle c; c_2, \sigma\rangle \longrightarrow^* \mathsf{abort}$, then we need to show that $\langle c; c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (b), and the semantics, we have 2 cases:
  - We consider (c) $\langle c, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (c), and the semantics, we know (d) $\langle c; c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (d), we conclude

  - We consider (c) $\langle c, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma'\rangle$ and (d) $\langle c_2, \sigma'\rangle \longrightarrow^* \textsf{abort}$. From (a) and (d), by Def. 3.33 (item 1), we know (e) $\langle c_1, \sigma'\rangle \longrightarrow^* \textsf{abort}$. From (c), (e), and the semantics, we know (f) $\langle c; c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (f), we conclude

- If (b) $\langle c; c_2, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$, we need to show that either $\langle c; c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$ or exists c_1' such that $\langle c; c_1, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and c_1' and c_2' satisfy constraints 2.a to 2.c of Def. 3.33. From (b), and the semantics, we have 2 cases:

  - We consider (c) $\langle c, \sigma\rangle \Downarrow \langle c_2'', \sigma'\rangle$ and (d) $c_2' = (c_2''; c_2)$. From (c), and the semantics, we have (e) $\langle c; c_1, \sigma\rangle \Downarrow \langle c_2''; c_1, \sigma'\rangle$. From (a), using the induction hypothesis, we know (f) constraints 2.b and 2.c of Def. 3.33 hold for $c_2''; c_1$ and $c_2''; c_2$. Instantiating the goal with $c_2''; c_1$, from (e), (f), we conclude
  - We consider (c) $\langle c, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma''\rangle$ and (d) $\langle c_2, \sigma''\rangle \Downarrow \langle c_2', \sigma'\rangle$. From (a) and (d), by Def. 3.33 (item 2), we have 2 cases:
    - We consider (e) $\langle c_1, \sigma''\rangle \longrightarrow^* \textsf{abort}$. From (c), (e), and the semantics, we have (f) $\langle c; c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (f), we conclude
    - We consider (e) $\langle c_1, \sigma''\rangle \Downarrow \langle c_1', \sigma'\rangle$ and (f) constraints 2.a to 2.c of Def. 3.33 hold for c_1' and c_2'. From (c), (e), and the semantics, we have (g) $\langle c; c_1, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$. Instantiating the goal with c_1', from (g) and (f), we conclude

- If (b) $\langle c; c_2, \sigma\rangle \Uparrow$, then we need to show that either $\langle c; c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$ or $\langle c; c_1, \sigma\rangle \Uparrow$. From (b), and the semantics, we have 2 cases:

  - We consider (c) $\langle c, \sigma\rangle \Uparrow$. From (c), and the semantics, we know (d) $\langle c; c_1, \sigma\rangle \Uparrow$. From (d), we conclude
  - We consider (c) $\langle c, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma'\rangle$ and (d) $\langle c_2, \sigma'\rangle \Uparrow$. From (a) and (d), by Def. 3.33 (item 3), we have 2 cases:
    - We consider (e) $\langle c_1, \sigma'\rangle \longrightarrow^* \textsf{abort}$. From (c), (e), and the semantics, we have (f) $\langle c; c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (f), we conclude

  * We consider (e) $\langle c_1, \sigma' \rangle \Uparrow$. From (c), (e), and the semantics, we have (f) $* \Uparrow \langle c; c_1, \sigma \rangle$. From (f), we conclude

- If (b) $\langle c; c_2, \sigma \rangle \xrightarrow[\delta_2]{}^* \kappa_2$, then we need to show that either $\langle c; c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$ or there exists δ_1 and κ_1 such that $\langle c; c_1, \sigma \rangle \xrightarrow[\delta_1]{}^* \kappa_1$ and $\delta_2 \subseteq \delta_1$. From (b), and the semantics we have 2 cases:

  - We consider (c) $\langle c, \sigma \rangle \xrightarrow[\delta_2]{}^* \langle c', \sigma' \rangle$ and (d) $\kappa_2 = \langle c'; c_2, \sigma' \rangle$. From (c), and the semantics, we know (e) $\langle c; c_1, \sigma \rangle \xrightarrow[\delta_2]{}^* \langle c'; c_1, \sigma' \rangle$. We also know (f) $\delta_2 \subseteq \delta_2$. Instantiating the goal with δ_2 and $\langle c'; c_1, \sigma' \rangle$, from (e) and (f), we conclude

  - We consider (c) $\langle c, \sigma \rangle \xrightarrow[\delta_2']{}^* \langle \textbf{skip}, \sigma' \rangle$, (d) $\langle c_2, \sigma' \rangle \xrightarrow[\delta_2'']{}^* \kappa_2$, and (e) $\delta_2 = \delta_2' \cup \delta_2''$. From (a) and (d), by Def. 3.33 (item 4), we know there exists δ_1'' and κ_1 such that (f) $\langle c_1, \sigma' \rangle \xrightarrow[\delta_1'']{}^* \kappa_1$ and (g) $\delta_2'' \subseteq \delta_1''$. From (c), (f), and the semantics, we know (h) $\langle c; c_1, \sigma \rangle \xrightarrow[\delta_2' \cup \delta_1'']{}^* \kappa_1$. From (g), we know (i) $\delta_2' \cup \delta_2'' \subseteq \delta_2' \cup \delta_1''$. Instantiating the goal with $\delta_2' \cup \delta_1$ and κ_1, from (h) and (i), we conclude □

Lemma 3.40. If $c_1 \rightsquigarrow c_2$, then $(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c) \rightsquigarrow (\textbf{if } b \textbf{ then } c_2 \textbf{ else } c)$

Proof. From Def. 3.33, we need to show that for all n, we have $(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c) \rightsquigarrow_n (\textbf{if } b \textbf{ then } c_2 \textbf{ else } c)$. By induction over n, if $n = 0$, by Def. 3.33, we conclude. If $n > 0$, assuming (a) $c_1 \rightsquigarrow c_2$, by Def. 3.33, we have 4 cases to consider:

- If (b) $\langle \textbf{if } b \textbf{ then } c_2 \textbf{ else } c, \sigma \rangle \longrightarrow^* \textsf{abort}$, we need to show that $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (b), and the semantics, we have 3 cases:

  - We consider (c) $\textbf{if } b \textbf{ then } c_2 \textbf{ else } c \longrightarrow \textsf{abort}$. From (c), and the semantics, we know (d) $\nexists z.\ [\![b]\!]_\sigma = z$. From (d), and the semantics, we know (e) $\textbf{if } b \textbf{ then } c_1 \textbf{ else } c \longrightarrow \textsf{abort}$. From (e), we conclude

  - We consider (c) $\textbf{if } b \textbf{ then } c_2 \textbf{ else } c \longrightarrow \langle c_2, \sigma \rangle$, (d) $[\![b]\!]_\sigma = \mathit{true}$, and (e) $\langle c_2, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (d), and the semantics, we know (f) $\textbf{if } b \textbf{ then } c_1 \textbf{ else } c \longrightarrow \langle c_1, \sigma \rangle$. From (a) and (e), by Def. 3.33 (item 1), we have (g) $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (f), (g), and the semantics, we know (h) $\textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c \longrightarrow^* \textsf{abort}$. From (h), we conclude

  - We consider (c) $\textbf{if}\ b\ \textbf{then}\ c_2\ \textbf{else}\ c \longrightarrow \langle c, \sigma\rangle$, (d) $[\![b]\!]_\sigma = \mathit{false}$, and (e) $\langle c, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (d), and the semantics, we know (f) $\textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c \longrightarrow \langle c, \sigma\rangle$. From (f), (e), and the semantics, we know (g) $\textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c \longrightarrow^* \textsf{abort}$. From (g), we conclude

- If (b) $\langle \textbf{if}\ b\ \textbf{then}\ c_2\ \textbf{else}\ c, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$, we need to show that either $\langle \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c, \sigma\rangle \longrightarrow^* \textsf{abort}$ or exists c_1' such that $\langle \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and constraints 2.a to 2.c of Def. 3.33 hold for c_1' and c_2'. From (b), and the semantics, we have 2 cases:

  - We consider (c) $\langle \textbf{if}\ b\ \textbf{then}\ c_2\ \textbf{else}\ c, \sigma\rangle \longrightarrow \langle c_2, \sigma\rangle$, (d) $[\![b]\!]_\sigma = \mathit{true}$, and (e) $\langle c_2, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$. From (a) and (e), by Def. 3.33 (item 2), we have 2 cases:

    * We consider (f) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (d), and the semantics, we have (g) $\langle \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c, \sigma\rangle \longrightarrow \langle c_1, \sigma\rangle$. From (g), (f), and the semantics, we have (h) $\langle \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (h), we conclude

    * We consider that exists c_1' such that (f) $\langle c_1, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and (g) constrains 2.a to 2.c of Def. 3.33 hold for c_1' and c_2'. From (d), and the semantics, we have (h) $\langle \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c, \sigma\rangle \longrightarrow \langle c_1, \sigma\rangle$. From (g), (f), and the semantics, we have (i) $\langle \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$. Instantiating the goal with c_1', from (i) and (g), we conclude

  - We consider (c) $\langle \textbf{if}\ b\ \textbf{then}\ c_2\ \textbf{else}\ c, \sigma\rangle \longrightarrow \langle c, \sigma\rangle$, (d) $[\![b]\!]_\sigma = \mathit{false}$, and (e) $\langle c, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$. From (d), and the semantics, we have (f) $\langle \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c, \sigma\rangle \longrightarrow \langle c, \sigma\rangle$. From (f), (e), and the semantics, we have (g) $\langle \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$. Using Lemma 3.34, we know that (h) constraints 2.a to 2.c of Def. 3.33 hold for c_2' and c_2'. Instantiating the goal with c_2', from (g) and (h), we conclude

- If (b) $\langle \textbf{if}\ b\ \textbf{then}\ c_2\ \textbf{else}\ c, \sigma\rangle \Uparrow$, then we need to show that either

$\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \longrightarrow^* \textsf{abort}$ or $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \Uparrow$. From (b), and the semantics, we have 2 cases:

- We consider (c) $\langle \textbf{if } b \textbf{ then } c_2 \textbf{ else } c, \sigma\rangle \longrightarrow \langle c_2, \sigma\rangle$, (d) $[\![b]\!]_\sigma = \mathit{true}$, and (e) $\langle c_2, \sigma\rangle \Uparrow$. From (a) and (e), by Def. 3.33 (item 3), we have 2 cases:
  - We consider (f) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (d), and the semantics, we have (g) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \longrightarrow \langle c_1, \sigma\rangle$. From (g), (f), and the semantics, we have (h) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (h), we conclude
  - We consider (f) $\langle c_1, \sigma\rangle \Uparrow$. From (d), and the semantics, we have (g) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \longrightarrow \langle c_1, \sigma\rangle$. From (g), (f), and the semantics, we have (h) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \Uparrow$. From (h), we conclude
- We consider (c) $\langle \textbf{if } b \textbf{ then } c_2 \textbf{ else } c, \sigma\rangle \longrightarrow \langle c, \sigma\rangle$, (d) $[\![b]\!]_\sigma = \mathit{false}$, and (e) $\langle c, \sigma\rangle \Uparrow$. From (d), and the semantics, we have (f) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \longrightarrow \langle c, \sigma\rangle$. From (f), (e), and the semantics, we have (g) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \Uparrow$. From (g), we conclude

- If (b) $\langle \textbf{if } b \textbf{ then } c_2 \textbf{ else } c, \sigma\rangle \xrightarrow[\delta_2]{}^* \kappa_2$, then we need to show that either $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \longrightarrow^* \textsf{abort}$ or there exists δ_1 and κ_1 such that $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \xrightarrow[\delta_1]{}^* \kappa_1$ and $\delta_2 \subseteq \delta_1$. From (b), and the semantics, we have 3 cases:
  - We consider (c) $\kappa_2 = \langle \textbf{if } b \textbf{ then } c_2 \textbf{ else } c, \sigma\rangle$ and (d) $\delta_2 = \mathit{emp}$. From the semantics, we know that (e) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \xrightarrow[\mathit{emp}]{}^* \langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle$. We know (f) $\mathit{emp} \subseteq \mathit{emp}$. Instantiating the goal with emp and $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle$, from (e) and (f), we conclude
  - We consider (c) $\langle \textbf{if } b \textbf{ then } c_2 \textbf{ else } c, \sigma\rangle \xrightarrow[\Delta^b_\sigma]{} \langle c_2, \sigma\rangle$, (d) $[\![b]\!]_\sigma = \mathit{true}$, (e) $\langle c_2, \sigma\rangle \xrightarrow[\delta'_2]{}^* \kappa_2$, and (f) $\delta_2 = \Delta^b_\sigma \cup \delta'_2$. From (a) and (e), by Def. 3.33 (item 4), we have 2 cases:
    - We consider (g) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (d), and the semantics, we have (h) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \longrightarrow \langle c_1, \sigma\rangle$. From (h), (g), and the semantics, we

have (i) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (i), we conclude

  * We consider that exists δ_1' and κ_1 such that (g) $\langle c_1, \sigma\rangle \underset{\delta_1'}{\longrightarrow^*} \kappa_1$ and (h) $\delta_2' \subseteq \delta_1'$. From (h), we know (i) $\Delta_\sigma^b \cup \delta_2' \subseteq \Delta_\sigma^b \cup \delta_1'$. From (d), and the semantics, we have (j) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \underset{\Delta_\sigma^b}{\longrightarrow} \langle c_1, \sigma\rangle$. From (j), (g), and the semantics, we have (k) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \underset{\Delta_\sigma^b \cup \delta_1'}{\longrightarrow^*} \kappa_1$. Instantiating the goal with $\Delta_\sigma^b \cup \delta_1'$ and κ_1, from (i) and (k), we conclude

– We consider (c) $\langle \textbf{if } b \textbf{ then } c_2 \textbf{ else } c, \sigma\rangle \underset{\Delta_\sigma^b}{\longrightarrow} \langle c, \sigma\rangle$, (d) $[\![b]\!]_\sigma = \mathit{false}$, (e) $\langle c, \sigma\rangle \underset{\delta_2'}{\longrightarrow^*} \kappa_2$, and (f) $\delta_2 = \Delta_\sigma^b \cup \delta_2'$. From (d), and the semantics, we have (g) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \underset{\Delta_\sigma^b}{\longrightarrow} \langle c, \sigma\rangle$. From (g), (e), and the semantics, we have (h) $\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c, \sigma\rangle \underset{\Delta_\sigma^b \cup \delta_2'}{\longrightarrow^*} \kappa_2$. We know (i) $\Delta_\sigma^b \cup \delta_2' \subseteq \Delta_\sigma^b \cup \delta_2'$. Instantiating the goal with $\Delta_\sigma^b \cup \delta_2'$ and κ_2, from (h) and (i), we conclude $\square$

Lemma 3.41. If $c_1 \rightsquigarrow c_2$, then $(\textbf{if } b \textbf{ then } c \textbf{ else } c_1) \rightsquigarrow (\textbf{if } b \textbf{ then } c \textbf{ else } c_2)$

Proof. From Def. 3.33, we need to show that for all n, we have $(\textbf{if } b \textbf{ then } c \textbf{ else } c_1) \rightsquigarrow_n (\textbf{if } b \textbf{ then } c \textbf{ else } c_2)$. By induction over n, if $n = 0$, by Def. 3.33, we conclude. If $n > 0$, assuming (a) $c_1 \rightsquigarrow c_2$, by Def. 3.33, we have 4 cases to consider:

- If (b) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_2, \sigma\rangle \longrightarrow^* \textsf{abort}$, we need to show that $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (b), and the semantics, we have 3 cases:

  – We consider (c) $\textbf{if } b \textbf{ then } c \textbf{ else } c_2 \longrightarrow \textsf{abort}$. From (c), and the semantics, we know (d) $\nexists z.\ [\![b]\!]_\sigma = z$. From (d), and the semantics, we know (e) $\textbf{if } b \textbf{ then } c \textbf{ else } c_1 \longrightarrow \textsf{abort}$. From (e), we conclude

  – We consider (c) $\textbf{if } b \textbf{ then } c \textbf{ else } c_2 \longrightarrow \langle c, \sigma\rangle$, (d) $[\![b]\!]_\sigma = \mathit{true}$, and (e) $\langle c, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (d), and the semantics, we know (f) $\textbf{if } b \textbf{ then } c \textbf{ else } c_1 \longrightarrow \langle c, \sigma\rangle$. From (f), (e), and the semantics, we know (g) $\textbf{if } b \textbf{ then } c \textbf{ else } c_1 \longrightarrow^* \textsf{abort}$. From (g), we conclude

  – We consider (c) $\textbf{if } b \textbf{ then } c \textbf{ else } c_2 \longrightarrow \langle c_2, \sigma\rangle$, (d) $[\![b]\!]_\sigma = \mathit{false}$, and (e) $\langle c_2, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (d), and the semantics, we know (f) $\textbf{if } b \textbf{ then } c \textbf{ else } c_1 \longrightarrow \langle c_1, \sigma\rangle$.

From (a) and (e), by Def. 3.33 (item 1), we have (g) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (f), (g), and the semantics, we know (h) $\textbf{if } b \textbf{ then } c \textbf{ else } c_1 \longrightarrow^* \textsf{abort}$. From (h), we conclude

- If (b) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_2, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$, we need to show that either $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$ or exists c_1' such that $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and constraints 2.a to 2.c of Def. 3.33 hold for c_1' and c_2'. From (b), and the semantics, we have 2 cases:
  - We consider (c) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle c, \sigma\rangle$, (d) $[\![b]\!]_\sigma = \mathit{true}$, and (e) $\langle c, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$. From (d), and the semantics, we have (f) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \longrightarrow \langle c, \sigma\rangle$. From (f), (e), and the semantics, we have (g) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$. Using Lemma 3.34, we know that (h) constraints 2.a to 2.c of Def. 3.33 hold for c_2' and c_2'. Instantiating the goal with c_2', from (g) and (h), we conclude
  - We consider (c) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle c_2, \sigma\rangle$, (d) $[\![b]\!]_\sigma = \mathit{false}$, and (e) $\langle c_2, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$. From (a) and (e), by Def. 3.33 (item 2), we have 2 cases:
    - We consider (f) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (d), and the semantics, we have (g) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \longrightarrow \langle c_1, \sigma\rangle$. From (g), (f), and the semantics, we have (h) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (h), we conclude
    - We consider that exists c_1' such that (f) $\langle c_1, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and (g) constrains 2.a to 2.c of Def. 3.33 hold for c_1' and c_2'. From (d), and the semantics, we have (h) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \longrightarrow \langle c_1, \sigma\rangle$. From (g), (f), and the semantics, we have (i) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$. Instantiating the goal with c_1', from (i) and (g), we conclude

- If (b) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_2, \sigma\rangle \Uparrow$, then we need to show that either $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$ or $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \Uparrow$. From (b), and the semantics, we have 2 cases:
  - We consider (c) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle c, \sigma\rangle$, (d) $[\![b]\!]_\sigma = \mathit{true}$, and (e) $\langle c, \sigma\rangle \Uparrow$.

From (d), and the semantics, we have (f) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle \longrightarrow \langle c, \sigma \rangle$. From (f), (e), and the semantics, we have (g) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle \Uparrow$. From (g), we conclude

 - We consider (c) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_2, \sigma \rangle \longrightarrow \langle c_2, \sigma \rangle$, (d) $[\![b]\!]_\sigma = \mathit{false}$, and (e) $\langle c_2, \sigma \rangle \Uparrow$. From (a) and (e), by Def. 3.33 (item 3), we have 2 cases:
   * We consider (f) $\langle c_1, \sigma \rangle \longrightarrow^* \texttt{abort}$. From (d), and the semantics, we have (g) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle \longrightarrow \langle c_1, \sigma \rangle$. From (g), (f), and the semantics, we have (h) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle \longrightarrow^* \texttt{abort}$. From (h), we conclude
   * We consider (f) $\langle c_1, \sigma \rangle \Uparrow$. From (d), and the semantics, we have (g) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle \longrightarrow \langle c_1, \sigma \rangle$. From (g), (f), and the semantics, we have (h) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle \Uparrow$. From (h), we conclude

- If (b) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_2, \sigma \rangle \underset{\delta_2}{\longrightarrow^*} \kappa_2$, then we need to show that either $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle \longrightarrow^* \texttt{abort}$ or there exists δ_1 and κ_1 such that $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle \underset{\delta_1}{\longrightarrow^*} \kappa_1$ and $\delta_2 \subseteq \delta_1$. From (b), and the semantics, we have 3 cases:

  - We consider (c) $\kappa_2 = \langle \textbf{if } b \textbf{ then } c \textbf{ else } c_2, \sigma \rangle$ and (d) $\delta_2 = \mathit{emp}$. From the semantics, we know that (e) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle \underset{\mathit{emp}}{\longrightarrow^*} \langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle$. We know (f) $\mathit{emp} \subseteq \mathit{emp}$. Instantiating the goal with emp and $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle$, from (e) and (f), we conclude
  - We consider (c) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_2, \sigma \rangle \underset{\Delta_\sigma^b}{\longrightarrow} \langle c, \sigma \rangle$, (d) $[\![b]\!]_\sigma = \mathit{true}$, (e) $\langle c, \sigma \rangle \underset{\delta_2'}{\longrightarrow^*} \kappa_2$, and (f) $\delta_2 = \Delta_\sigma^b \cup \delta_2'$. From (d), and the semantics, we have (g) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle \underset{\Delta_\sigma^b}{\longrightarrow} \langle c, \sigma \rangle$. From (g), (e), and the semantics, we have (h) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma \rangle \underset{\Delta_\sigma^b \cup \delta_2'}{\longrightarrow^*} \kappa_2$. We know (i) $\Delta_\sigma^b \cup \delta_2' \subseteq \Delta_\sigma^b \cup \delta_2'$. Instantiating the goal with $\Delta_\sigma^b \cup \delta_2'$ and κ_2, from (h) and (i), we conclude
  - We consider (c) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_2, \sigma \rangle \underset{\Delta_\sigma^b}{\longrightarrow} \langle c_2, \sigma \rangle$, (d) $[\![b]\!]_\sigma = \mathit{false}$, (e) $\langle c_2, \sigma \rangle \underset{\delta_2'}{\longrightarrow^*} \kappa_2$, and (f) $\delta_2 = \Delta_\sigma^b \cup \delta_2'$. From (a) and (e), by Def. 3.33 (item 4), we have 2 cases:

    * We consider (g) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (d), and the semantics, we have (h) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \longrightarrow \langle c_1, \sigma\rangle$. From (h), (g), and the semantics, we have (i) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (i), we conclude
    * We consider that exists δ'_1 and κ_1 such that (g) $\langle c_1, \sigma\rangle \underset{\delta'_1}{\longrightarrow^*} \kappa_1$ and (h) $\delta'_2 \subseteq \delta'_1$. From (h), we know (i) $\Delta^b_\sigma \cup \delta'_2 \subseteq \Delta^b_\sigma \cup \delta'_1$. From (d), and the semantics, we have (j) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \underset{\Delta^b_\sigma}{\longrightarrow} \langle c_1, \sigma\rangle$. From (j), (g), and the semantics, we have (k) $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c_1, \sigma\rangle \underset{\Delta^b_\sigma \cup \delta'_1}{\longrightarrow^*} \kappa_1$. Instantiating the goal with $\Delta^b_\sigma \cup \delta'_1$ and κ_1, from (i) and (k), we conclude □

Lemma 3.42. If $c_1 \rightsquigarrow c_2$, then $(\textbf{while } b \textbf{ do } c_1) \rightsquigarrow (\textbf{while } b \textbf{ do } c_2)$

Proof. From Def. 3.33, we need to show for all n, that $(\textbf{while } b \textbf{ do } c_1) \rightsquigarrow_n (\textbf{while } b \textbf{ do } c_2)$. By induction over n, if $n = 0$, by Def. 3.33, we conclude. If $n > 0$, assuming (a) $c_1 \rightsquigarrow c_2$, by Def. 3.33, we have 4 cases to consider:

- If (b) $\langle \textbf{while } b \textbf{ do } c_2, \sigma\rangle \longrightarrow^* \textsf{abort}$, we need to show that $\langle \textbf{while } b \textbf{ do } c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. We perform a second induction over n', the number of steps taken in (b), and we have 2 cases: If $n' = 0$, we know (b) is false and conclude; If $n' > 0$, from (b), and the semantics, we have 3 cases:
  - We consider $n' = 2$ and (c) $\nexists z.\ [\![b]\!]_\sigma = z$. From (c), we know (d) $\langle \textbf{while } b \textbf{ do } c_1, \sigma\rangle \longrightarrow^2 \textsf{abort}$. From (d), we conclude
  - We consider (c) $[\![b]\!]_\sigma = \mathit{true}$ and (d) $\langle c_2, \sigma\rangle \longrightarrow^{n'-2} \textsf{abort}$. From (a) and (d), by Def. 3.33 (item 1), we have (e) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (c), (e), and the semantics, we know (e) $\langle \textbf{while } b \textbf{ do } c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (e), we conclude
  - We consider that (c) $[\![b]\!]_\sigma = \mathit{true}$ and there exists σ' such that (d) $\langle c_2, \sigma\rangle \Downarrow \langle \textbf{skip}, \sigma'\rangle$ and (e) $\langle \textbf{while } b \textbf{ do } c_2, \sigma'\rangle \longrightarrow^{n''} \textsf{abort}$ where $n'' < n'$. From (a) and (d), by Def. 3.33 (item 2.a), we have 2 cases:
    * We consider (f) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (c), (f), and the semantics, we know (g) $\langle \textbf{while } b \textbf{ do } c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (g), we conclude

    * We consider (f) $\langle c_1, \sigma \rangle \Downarrow \langle \mathbf{skip}, \sigma' \rangle$. From (a) and (e), using the second induction hypothesis, we have (g) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma' \rangle \longrightarrow^* \mathsf{abort}$ From (c), (f), (g), and the semantics, we know (h) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$. From (h), we conclude

- If (b) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_2, \sigma \rangle \Downarrow \langle c_2', \sigma' \rangle$, we need to show that either $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$ or exists c_1' such that $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma \rangle \Downarrow \langle c_1', \sigma' \rangle$ and c_1' and c_2' satisfy constraints 2.a to 2.c of Def. 3.33. We perform a second induction over n', the number of steps taken in (b), and we have 2 cases: If $n' = 0$, we know (b) is false and conclude; If $n' > 0$, from (b), and the semantics, we have 3 cases:
  - We consider (c) $[\![b]\!]_\sigma = \mathit{false}$ and (d) $\langle c_2', \sigma' \rangle = \langle \mathbf{skip}, \sigma \rangle$. From (c), and the semantics, we know (e) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma \rangle \Downarrow \langle \mathbf{skip}, \sigma \rangle$. Instantiating the goal with **skip**, from (e), we conclude
  - We consider (c) $[\![b]\!]_\sigma = \mathit{true}$, (d) $\langle c_2, \sigma \rangle \Downarrow \langle c_2'', \sigma' \rangle$, and (e) $c_2' = (c_2''; \mathbf{while}\ b\ \mathbf{do}\ c_2)$. From (a) and (d), by Def. 3.33 (item 2), we have 2 cases:
    * We consider (f) $\langle c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$. From (c), (f), and the semantics, we have (g) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$. From (g), we conclude
    * We consider (f) $\langle c_1, \sigma \rangle \Downarrow \langle c_1'', \sigma' \rangle$ and (g) constraints 2.b and 2.c of Def. 3.33 hold for c_1'' and c_2''. From (c), (f), and the semantics, we have (h) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma \rangle \Downarrow \langle c_1''; \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma' \rangle$. From (g), using Lemma 3.38, we know (i) constraints 2.b and 2.c of Def. 3.33 hold for $c_1''; \mathbf{while}\ b\ \mathbf{do}\ c_1$ and $c_2''; \mathbf{while}\ b\ \mathbf{do}\ c_1$. From (a), using the induction hypothesis, we know (j) $\mathbf{while}\ b\ \mathbf{do}\ c_1 \rightsquigarrow_{n-1} \mathbf{while}\ b\ \mathbf{do}\ c_2$. From (j), using Lemma 3.39, we know (k) $c_2''; \mathbf{while}\ b\ \mathbf{do}\ c_1 \rightsquigarrow_{n-1} c_2''; \mathbf{while}\ b\ \mathbf{do}\ c_2$. From (i) and (k), using Lemma 3.35, we know (l) constraints 2.b and 2.c of Def. 3.33 hold for $c_1''; \mathbf{while}\ b\ \mathbf{do}\ c_1$ and $c_2''; \mathbf{while}\ b\ \mathbf{do}\ c_2$. Instantiating the goal with $c_1''; \mathbf{while}\ b\ \mathbf{do}\ c_1$, from (h), (l), we conclude

    - We consider (c) $[\![b]\!]_\sigma = \mathit{true}$, (d) $\langle c_2, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma''\rangle$, and (e) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_2, \sigma''\rangle \Downarrow \langle c_2', \sigma'\rangle$ where the number of steps is less than n'. From (a) and (d), by Def. 3.33 (item 2), we have 2 cases:
        * We consider (f) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (c), (f), and the semantics, we have (g) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (g), we conclude
        * We consider (f) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma''\rangle$. From (e), using the second induction hypothesis, we have 2 cases:
            · We consider (g) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma''\rangle \longrightarrow^* \mathsf{abort}$. From (c), (f), (g), and the semantics, we have (h) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (h), we conclude

            · We have (g) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma''\rangle \Downarrow \langle c_2', \sigma'\rangle$ and (h) c_1' and c_2' satisfy constraints 2.a to 2.c of Def. 3.33. From (c), (f), (g), and the semantics, we have (i) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$. Instantiating the goal with c_1', from (i) and (h), we conclude

- If (b) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_2, \sigma\rangle \Uparrow$, we need to show that either $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$ or $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma\rangle \Uparrow$. From (b), and the semantics, we know exists n' and σ' such that (c) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_2, \sigma\rangle \longrightarrow^{n'} \langle c_2; \mathbf{while}\ b\ \mathbf{do}\ c_2, \sigma'\rangle$ and (d) $\langle c_2, \sigma'\rangle \Uparrow$. We perform a second induction over n' and we have 3 cases: If $n' < 2$, we know (c) is false and conclude; If $n' = 2$, from (c), and the semantics, we know (e) $[\![b]\!]_\sigma = \mathit{true}$ and (f) $\langle c_2, \sigma\rangle \Uparrow$. From (a) and (f), by Def. 3.33 (item 3), we have 2 cases:
    - We consider (g) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (e), (g), and the semantics, we have (h) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (h), we conclude
    - We consider (g) $\langle c_1, \sigma\rangle \Uparrow$. From (e), (g), and the semantics, we have (h) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_1, \sigma\rangle \Uparrow$. From (h), we conclude

  If $n' > 2$, from (c), and the semantics, we know (e) $[\![b]\!]_\sigma = \mathit{true}$, (f) $\langle c_2, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma''\rangle$, and (g) $\langle \mathbf{while}\ b\ \mathbf{do}\ c_2, \sigma''\rangle \longrightarrow^{n''} \langle c_2; \mathbf{while}\ b\ \mathbf{do}\ c_2, \sigma'\rangle$ where $n'' < n'$. From (a) and (f), by Def. 3.33 (item 2.a), we have 2 cases:

- We consider (h) $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (e), (h), and the semantics, we have (i) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (i), we conclude
- We consider (h) $\langle c_1, \sigma \rangle \Downarrow \langle \textbf{skip}, \sigma'' \rangle$. From (g) and (d), using the second induction hypothesis, we have 2 cases:
  - We consider (i) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma'' \rangle \longrightarrow^* \textsf{abort}$. From (e), (h), (i), and the semantics, we have (j) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (j), we conclude
  - We have (i) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma'' \rangle \Uparrow$. From (e), (h), (i), and the semantics, we have (j) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma \rangle \Uparrow$. From (j), we conclude

- If (b) $\langle \textbf{while}\ b\ \textbf{do}\ c_2, \sigma \rangle \xrightarrow[\delta_2]{}^* \kappa_2$, we need to show that either $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$ or exists δ_1 and κ_1 such that $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma \rangle \xrightarrow[\delta_1]{}^* \kappa_1$ and $\delta_2 \subseteq \delta_1$. We perform a second induction over n', the number of steps taken in (b), and we have 2 cases: If $n' = 0$, we know (c) $\delta_2 = emp$ and (d) $\kappa_2 = \langle \textbf{while}\ b\ \textbf{do}\ c_2, \sigma \rangle$. From the semantics, we know (e) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma \rangle \longrightarrow^0 \langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma \rangle$. We know (f) $emp \subseteq emp$. Instantiating the goal with emp and $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma \rangle$, from (e) and (f), we conclude If $n' > 0$, from (b), and the semantics, we have 4 cases:
  - We consider (c) $[\![b]\!]_\sigma = \mathit{false}$ and (d) $\delta_2 = \Delta^b_\sigma$. From (c), and the semantics, we know (e) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma \rangle \xrightarrow[\Delta^b_\sigma]{}^2 \langle \textbf{skip}, \sigma \rangle$. We know (f) $\Delta^b_\sigma \subseteq \Delta^b_\sigma$. Instantiating the goal with Δ^b_σ and $\langle \textbf{skip}, \sigma \rangle$, from (e) and (f), we conclude
  - We consider (c) $[\![b]\!]_\sigma = \mathit{true}$, (d) $\kappa_2 = \textsf{abort}$, (e) $\delta_2 = \Delta^b_\sigma \cup \delta'_2$, and (f) $\langle c_2, \sigma \rangle \xrightarrow[\delta'_2]{}^* \textsf{abort}$. From (a) and (f), by Def. 3.33 (item 4), we know (g) $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (c), (g), and the semantics, we have (h) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (h), we conclude
  - We consider (c) $[\![b]\!]_\sigma = \mathit{true}$, (d) $\kappa_2 = \langle c'_2; \textbf{while}\ b\ \textbf{do}\ c_2, \sigma' \rangle$, (e) $\delta_2 = \Delta^b_\sigma \cup \delta'_2$, and (f) $\langle c_2, \sigma \rangle \xrightarrow[\delta'_2]{}^* \langle c'_2, \sigma' \rangle$. From (a) and (e), by Def. 3.33 (item 4), we have 2 cases:

    * We consider (g) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (c), (g), and the semantics, we have (h) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (h), we conclude
    * We consider (g) $\langle c_1, \sigma\rangle \xrightarrow[\delta_1]{}^* \langle c_1', \sigma''\rangle$ and (h) $\delta_2' \subseteq \delta_1$. From (c), (g), and the semantics, we have (i) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma\rangle \xrightarrow[\Delta_\sigma^b \cup \delta_1]{}^* \langle c_1'; \textbf{while}\ b\ \textbf{do}\ c_1, \sigma''\rangle$. From (h), we know (j) $\Delta_\sigma^b \cup \delta_2' \subseteq \Delta_\sigma^b \cup \delta_1$. Instantiating the goal with $\Delta_\sigma^b \cup \delta_1$ and $\langle c_1'; \textbf{while}\ b\ \textbf{do}\ c_1, \sigma''\rangle$, from (e), (i), (j), we conclude
  - We consider (c) $[\![b]\!]_\sigma = \mathit{true}$, (d) $\langle c_2, \sigma\rangle \Downarrow \langle \textbf{skip}, \sigma'\rangle$ with footprint δ_2', (e) $\langle \textbf{while}\ b\ \textbf{do}\ c_2, \sigma'\rangle \xrightarrow[\delta_2'']{}^{n''} \kappa_2$ where $n'' < n'$, and (f) $\delta_2 = \Delta_\sigma^b \cup \delta_2' \cup \delta_2''$. From (a) and (d), by Def. 3.33 (items 2.a and 4), we have 2 cases:
    * We consider (g) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (c), (g), and the semantics, we have (g) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (g), we conclude
    * We consider (g) $\langle c_1, \sigma\rangle \Downarrow \langle \textbf{skip}, \sigma'\rangle$ with footprint δ_1', and (h) $\delta_2' \subseteq \delta_1'$. From (e), using the second induction hypothesis, we have 2 cases:
      · We consider (i) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma'\rangle \longrightarrow^* \mathsf{abort}$. From (c), (g), (i), and the semantics, we have (j) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (j), we conclude
      · We have (i) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma'\rangle \xrightarrow[\delta_1'']{}^* \kappa_1$ and (j) $\delta_2'' \subseteq \delta_1''$. From (c), (g), (i), and the semantics, we have (k) $\langle \textbf{while}\ b\ \textbf{do}\ c_1, \sigma\rangle \xrightarrow[\Delta_\sigma^b \cup \delta_1' \cup \delta_1'']{}^* \kappa_1$. From (h) and (j), we know (l) $\Delta_\sigma^b \cup \delta_2' \cup \delta_2'' \subseteq \Delta_\sigma^b \cup \delta_1' \cup \delta_1''$. Instantiating the goal with $\Delta_\sigma^b \cup \delta_1' \cup \delta_1''$ and κ_1, from (f), (k), and (l), we conclude □

Lemma 3.43. If $c_1 \rightsquigarrow c_2$, then $(c_1 \,\|\, c) \rightsquigarrow (c_2 \,\|\, c)$

Proof. From Def. 3.33, we need to show for all n, that $(c_1 \,\|\, c) \rightsquigarrow_n (c_2 \,\|\, c)$. By induction over n, if $n = 0$, by Def. 3.33, we conclude. If $n > 0$, assuming (a) $c_1 \rightsquigarrow c_2$, by Def. 3.33, we have 4 cases to consider:

- We assume (b) $\langle c_2 \,\|\, c, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From the semantics, we know (b) is false and conclude

- If (b) $\langle c_2 \parallel c, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$, we will to show that exists c_1' such that $\langle c_1 \parallel c, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and c_1' and c_2' satisfy constraints 2.a to 2.c of Def. 3.33. From (b), and the semantics, we know (c) $c_2' = c_2 \parallel c$ and (d) $\sigma' = \sigma$. From the semantics, we know (e) $\langle c_1 \parallel c, \sigma\rangle \Downarrow \langle c_1 \parallel c, \sigma\rangle$. From (a), and the induction hypothesis, we know that (f) constraints 2.a to 2.c of Def. 3.33 hold for $c_1 \parallel c$ and $c_2 \parallel c$. Instantiating the goal with $c_1 \parallel c$, from (e) and (f), we conclude

- We assume (b) $\langle c_2 \parallel c, \sigma\rangle \Uparrow$. From the semantics, we know (b) is false and conclude

- If (b) $\langle c_2 \parallel c, \sigma\rangle \xrightarrow[\delta_2]{}^* \kappa_2$, we will show that there exists δ_1 and κ_1 such that $\langle c_1 \parallel c, \sigma\rangle \xrightarrow[\delta_1]{}^* \kappa_1$ and $\delta_2 \subseteq \delta_1$. From (b), and the semantics, we know (c) $\kappa_2 = \langle c_2 \parallel c, \sigma\rangle$ and (d) $\delta_2 = \mathit{emp}$. From the semantics, we know (e) $\langle c_1 \parallel c, \sigma\rangle \xrightarrow[\mathit{emp}]{}^* \langle c_1 \parallel c, \sigma\rangle$. We know (f) $\mathit{emp} \subseteq \mathit{emp}$. Instantiating the goal with $c_1 \parallel c$, from (e) and (f), we conclude □

Lemma 3.44. If $c_1 \rightsquigarrow c_2$, then $(c \parallel c_1) \rightsquigarrow (c \parallel c_2)$

Proof. From Def. 3.33, we need to show for all n, that $(c \parallel c_1) \rightsquigarrow_n (c \parallel c_2)$. By induction over n, if $n = 0$, by Def. 3.33, we conclude. If $n > 0$, assuming (a) $c_1 \rightsquigarrow c_2$, by Def. 3.33, we have 4 cases to consider:

- We assume (b) $\langle c \parallel c_2, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From the semantics, we know (b) is false and conclude

- If (b) $\langle c \parallel c_2, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$, we will to show that exists c_1' such that $\langle c \parallel c_1, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and c_1' and c_2' satisfy constraints 2.a to 2.c of Def. 3.33. From (b), and the semantics, we know (c) $c_2' = c \parallel c_2$ and (d) $\sigma' = \sigma$. From the semantics, we know (e) $\langle c \parallel c_1, \sigma\rangle \Downarrow \langle c \parallel c_1, \sigma\rangle$. From (a), and the induction hypothesis, we know that (f) constraints 2.a to 2.c of Def. 3.33 hold for $c \parallel c_1$ and $c \parallel c_2$. Instantiating the goal with $c \parallel c_1$, from (e) and (f), we conclude

- We assume (b) $\langle c \parallel c_2, \sigma\rangle \Uparrow$. From the semantics, we know (b) is false and conclude

- If (b) $\langle c \parallel c_2, \sigma\rangle \xrightarrow[\delta_2]{}^* \kappa_2$, we will show that there exists δ_1 and κ_1 such that $\langle c \parallel c_1, \sigma\rangle \xrightarrow[\delta_1]{}^* \kappa_1$ and $\delta_2 \subseteq \delta_1$. From (b), and the semantics, we know (c) $\kappa_2 = \langle c \,\|\, c_2, \sigma\rangle$ and (d) $\delta_2 = \mathit{emp}$. From the semantics, we know (e) $\langle c \,\|\, c_1, \sigma\rangle \xrightarrow[\mathit{emp}]{}^* \langle c \,\|\, c_1, \sigma\rangle$. We know (f) $\mathit{emp} \subseteq \mathit{emp}$. Instantiating the goal with $c \parallel c_1$, from (e) and (f), we conclude $\square$

Lemma 3.45. If $c_1 \rightsquigarrow c_2$, then $(\textbf{atomic}\ c_1) \rightsquigarrow (\textbf{atomic}\ c_2)$

Proof. From Def. 3.33, we need to show for all n, that $(\textbf{atomic}\ c_1) \rightsquigarrow_n (\textbf{atomic}\ c_2)$. By induction over n, if $n = 0$, by Def. 3.33, we conclude. If $n > 0$, assuming (a) $c_1 \rightsquigarrow c_2$, by Def. 3.33, we have 4 cases to consider:

- We assume (b) $\langle \textbf{atomic}\ c_2, \sigma\rangle \longrightarrow^* \texttt{abort}$. From the semantics, we know (b) is false and conclude

- If (b) $\langle \textbf{atomic}\ c_2, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$, we will to show that exists c_1' such that $\langle \textbf{atomic}\ c_1, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and c_1' and c_2' satisfy constraints 2.a to 2.c of Def. 3.33. From (b), and the semantics, we know (c) $c_2' = \textbf{atomic}\ c_2$ and (d) $\sigma' = \sigma$. From the semantics, we know (e) $\langle \textbf{atomic}\ c_1, \sigma\rangle \Downarrow \langle \textbf{atomic}\ c_1, \sigma\rangle$. From (a), and the induction hypothesis, we know that (f) constraints 2.a to 2.c of Def. 3.33 hold for **atomic** c_1 and **atomic** c_2. Instantiating the goal with **atomic** c_1, from (e) and (f), we conclude

- We assume (b) $\langle \textbf{atomic}\ c_2, \sigma\rangle \Uparrow$. From the semantics, we know (b) is false and conclude

- If (b) $\langle \textbf{atomic}\ c_2, \sigma\rangle \xrightarrow[\delta_2]{}^* \kappa_2$, we will show that there exists δ_1 and κ_1 such that $\langle \textbf{atomic}\ c_1, \sigma\rangle \xrightarrow[\delta_1]{}^* \kappa_1$ and $\delta_2 \subseteq \delta_1$. From (b), and the semantics, we know (c) $\kappa_2 = \langle \textbf{atomic}\ c_2, \sigma\rangle$ and (d) $\delta_2 = \mathit{emp}$. From the semantics, we know (e) $\langle \textbf{atomic}\ c_1, \sigma\rangle \xrightarrow[\mathit{emp}]{}^* \langle \textbf{atomic}\ c_1, \sigma\rangle$. We know (f) $\mathit{emp} \subseteq \mathit{emp}$. Instantiating the goal with **atomic** c_1, from (e) and (f), we conclude $\square$

3.10 Relaxed Semantics

We obtain our relaxed operational semantics by instantiating Λ from our parameterized semantics (from Fig. 3.13) with our subsumption relation from Definition 3.33:

$$[\rightsquigarrow]\ \kappa \longmapsto \kappa'$$

This semantics performs a program transformation, according to our subsumption relation, at each step. This resembles a dynamic compiler that modifies the program as it executes. However, as we show in Lemma 3.46, the execution following this semantics is equivalent to performing one single initial program transformation and then executing the target program using the interleaving semantics. This resembles a static compiler that modifies the program prior to execution. Note that in order to establish Lemma 3.46 and auxiliary Lemma 3.47 and Lemma 3.48, we assume a weaker notion of command equality where a command c_1 is equal to c_2 if, and only if, $c_1 = c_2$ or $c_1 = (\mathbf{skip}; \ldots; \mathbf{skip}; c_2)$, and vice-versa.

Lemma 3.46. $[\rightsquigarrow]\ \langle T, \sigma\rangle \longmapsto^* \kappa$ holds if, and only if, there exists a T' such that $T \rightsquigarrow_{\mathsf{t}} T'$ and $\langle T', \sigma\rangle \longmapsto^* \kappa$.

Proof. We have two cases:

- If (a) $[\rightsquigarrow]\ \langle T, \sigma\rangle \longmapsto^* \kappa$, we have to show that exists T' such that $T \rightsquigarrow_{\mathsf{t}} T'$ and $\langle T', \sigma\rangle \longmapsto^* \kappa$. Proof by induction over the number of steps n in (a).
  - We consider the base case, where (b) $n = 0$ and (c) $\kappa = \langle T, \sigma\rangle$. By Def. 3.33 and Def. 3.23 (and using Lemma 3.34), we have (d) $T \rightsquigarrow_{\mathsf{t}} T$. From the semantics, we know (e) $\langle T, \sigma\rangle \longmapsto^0 \langle T, \sigma\rangle$. Instantiating the goal with T, from (d) and (e), we conclude
  - In the inductive case, we know (b) $[\rightsquigarrow]\ \langle T, \sigma\rangle \longmapsto \kappa'$ and (c) $[\rightsquigarrow]\ \kappa' \longmapsto^{n-1} \kappa$. From the semantics, we need to consider 3 cases:

    * We consider (d) $\kappa' = \textsf{abort}$, (e) $n-1 = 0$, and (f) $\kappa = \kappa'$. From (b), and the semantics, we know that exists T' such that (g) $T \leadsto_{\mathsf{t}} T'$ and (h) $\langle T', \sigma \rangle \longmapsto \textsf{abort}$. Instantiating the goal with T', from (g) and (h), we conclude
    * We consider (d) $\kappa' = \textsf{race}$, (e) $n-1 = 0$, and (f) $\kappa = \kappa'$. From (b), and the semantics, we know that exists T' such that (g) $T \leadsto_{\mathsf{t}} T'$ and (h) $\langle T', \sigma \rangle \longmapsto \textsf{race}$. Instantiating the goal with T', from (g) and (h), we conclude
    * We consider (d) $\kappa' = \langle T', \sigma' \rangle$. From (b), (d), and the semantics, we know there exists T'' such that (e) $T \leadsto_{\mathsf{t}} T''$ and (f) $\langle T'', \sigma \rangle \longmapsto \langle T', \sigma' \rangle$ From (c), (d), and the induction hypothesis, we know there exists T''' such that (g) $T' \leadsto_{\mathsf{t}} T'''$ and (h) $\langle T''', \sigma' \rangle \longmapsto^{*} \kappa$. From (f) and (g), using Lemma 3.48, we know there exists T'''' such that (i) $T'' \leadsto_{\mathsf{t}} T''''$ and (j) $\langle T'''', \sigma \rangle \longmapsto \langle T''', \sigma' \rangle$. From (e) and (i), by Def. 3.33 and Def. 3.23 (and using Lemma 3.35), we know (k) $T \leadsto_{\mathsf{t}} T''''$. From (j), (h), and the semantics, we have (l) $\langle T'''', \sigma \rangle \longmapsto^{*} \kappa$. Instantiating the goal with T'''', from (k) and (l), we conclude

- If (a) $T \leadsto_{\mathsf{t}} T'$ and (b) $\langle T', \sigma \rangle \longmapsto^{*} \kappa$, we have to show that $[\leadsto]\ \langle T, \sigma \rangle \longmapsto^{*} \kappa$. Proof by induction over the number of steps n in (b).
  - We consider the base case, where (c) $n = 0$ and (d) $\kappa = \langle T', \sigma \rangle$, and we will show that $[\leadsto]\ \langle T, \sigma \rangle \longmapsto^{1} \langle T', \sigma \rangle$. We know that (e) $T' = \mathbf{T}[\, c\,]$. From the semantics, we know that (f) $\langle \mathbf{T}[\, \mathbf{skip}; c\,], \sigma \rangle \longmapsto \langle \mathbf{T}[\, c\,], \sigma \rangle$. It is not hard to show that (g) $c \leadsto (\mathbf{skip}; c)$. From (g), by Def. 3.33 and Def. 3.23 (and using Lemma 3.34), we have (h) $\mathbf{T}[\, c\,] \leadsto_{\mathsf{t}} \mathbf{T}[\, \mathbf{skip}; c\,]$. From (a), (e), and (h), by Def. 3.33 and Def. 3.23 (and using Lemma 3.35), we have (i) $T \leadsto_{\mathsf{t}} \mathbf{T}[\, \mathbf{skip}; c\,]$. From (i), (f), and (e), we conclude
  - In the inductive case, we know (c) $\langle T', \sigma \rangle \longmapsto \kappa'$ and (d) $\kappa' \longmapsto^{n-1} \kappa$. From the semantics, we need to consider 3 cases:
    * We consider (e) $\kappa' = \textsf{abort}$, (f) $n-1 = 0$, (g) $\kappa = \kappa'$, and we will show that

$[\rightsquigarrow]\ \langle T, \sigma \rangle \longmapsto^1$ abort. From (a) and (c), we conclude

$*$ We consider (e) $\kappa' = \text{race}$, (f) $n-1 = 0$, (g) $\kappa = \kappa'$, and we will show that $[\rightsquigarrow]\ \langle T, \sigma \rangle \longmapsto^1$ race. From (a) and (c), we conclude

$*$ We consider (e) $\kappa' = \langle T'', \sigma' \rangle$. By Def. 3.33 and Def. 3.23 (and also using Lemma 3.34), we have (f) $T'' \rightsquigarrow_{\mathsf{t}} T''$. From (e), (f), and (d), using the induction hypothesis, we have (g) $[\rightsquigarrow]\ \langle T'', \sigma' \rangle \longmapsto^* \kappa$. From (a), (c), and the semantics, we have (h) $[\rightsquigarrow]\ \langle T, \sigma \rangle \longmapsto \langle T'', \sigma' \rangle$ From (h), (g), and the semantics, we conclude $\square$

The key idea in the proof of Lemma 3.46 is given by Lemma 3.47 that basically says that a step followed by a transformation can commute; the converse is not possible.

Lemma 3.47. If $\langle c_1, \sigma \rangle \longrightarrow \langle c_1', \sigma' \rangle$, and $c_1' \rightsquigarrow c_2'$, then there exists c_2 such that $c_1 \rightsquigarrow c_2$ and $\langle c_2, \sigma \rangle \longrightarrow \langle c_2', \sigma' \rangle$.

Proof. We assume (a) $\langle c_1, \sigma \rangle \longrightarrow \langle c_1', \sigma' \rangle$ and (b) $c_1' \rightsquigarrow c_2'$. From (a) we have 5 cases:

- We consider (c) $c_1 = \mathbf{S}[\,a\,]$, (d) $c_1' = \mathbf{S}[\,\mathbf{skip}\,]$, and (e) $(\sigma, \sigma') \in [\![a]\!]$. We instantiate the goal with $a; c_2'$, and it remains to show that $\mathbf{S}[\,a\,] \rightsquigarrow a; c_2'$ and $\langle a; c_2', \sigma \rangle \longrightarrow \langle c_2', \sigma' \rangle$. Considering (f) $\mathbf{S}' = \bullet; c_2'$, from (e), and the semantics, we have (g) $\langle \mathbf{S}'[\,a\,], \sigma \rangle \longrightarrow \langle \mathbf{S}'[\,\mathbf{skip}\,], \sigma' \rangle$. From (f), and the weaker notion of equality, we have (h) $\mathbf{S}'[\,\mathbf{skip}\,] = c_2'$. From (b), using Lemma 3.39, we have (i) $(a; c_1') \rightsquigarrow (a; c_2')$. From (d), and the weaker notion of equality, we have (j) $\mathbf{S}[\,a\,] = a; c_1'$. From (i), (j), (g), and (h), we conclude

- We consider (c) $c_1 = \mathbf{S}[\,\mathbf{skip}; c\,]$, (d) $c_1' = \mathbf{S}[\,c\,]$, and (e) $\sigma = \sigma'$. We instantiate the goal with $\mathbf{skip}; c_2'$, and it remains to show that $\mathbf{S}[\,\mathbf{skip}; c\,] \rightsquigarrow \mathbf{skip}; c_2'$ and $\langle \mathbf{skip}; c_2', \sigma \rangle \longrightarrow \langle c_2', \sigma \rangle$. From the weaker notion of equality, we know that (f) $\mathbf{skip}; c = c$ and (g) $\mathbf{skip}; c_2' = c_2'$. From (b), (f), (g), and the semantics, we conclude

- We consider (c) $c_1 = \mathbf{S}[\,\mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{else}\ c'\,]$, (d) $c_1' = \mathbf{S}[\,c\,]$, (e) $\sigma = \sigma'$, and (f) $[\![b]\!]_\sigma = \mathit{true}$. We instantiate the goal with $\mathbf{if}\ b\ \mathbf{then}\ c_2'\ \mathbf{else}\ \mathbf{S}[\,c'\,]$, and it remains to show that

$\mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ c'\,] \rightsquigarrow \textbf{if}\ b\ \textbf{then}\ c'_2\ \textbf{else}\ \mathbf{S}[\,c'\,]$ and $\langle \textbf{if}\ b\ \textbf{then}\ c'_2\ \textbf{else}\ \mathbf{S}[\,c'\,], \sigma\rangle \longrightarrow \langle c'_2, \sigma\rangle$. From (b), using Lemma 3.40, we have
(g) $\textbf{if}\ b\ \textbf{then}\ \mathbf{S}[\,c\,]\ \textbf{else}\ \mathbf{S}[\,c'\,] \rightsquigarrow \textbf{if}\ b\ \textbf{then}\ c'_2\ \textbf{else}\ \mathbf{S}[\,c'\,]$. It is not hard to show that (h) $\mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ c'\,] \rightsquigarrow \textbf{if}\ b\ \textbf{then}\ \mathbf{S}[\,c\,]\ \textbf{else}\ \mathbf{S}[\,c'\,]$. From (h) and (g), using Lemma 3.35, we have (i) $\mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ c'\,] \rightsquigarrow \textbf{if}\ b\ \textbf{then}\ c'_2\ \textbf{else}\ \mathbf{S}[\,c'\,]$. From (i), (e), and the semantics; we conclude

- We consider (c) $c_1 = \mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ c'\ \textbf{else}\ c\,]$, (d) $c'_1 = \mathbf{S}[\,c\,]$, (e) $\sigma = \sigma'$, and (f) $[\![b]\!]_\sigma = \mathit{false}$. We instantiate the goal with $\textbf{if}\ b\ \textbf{then}\ \mathbf{S}[\,c'\,]\ \textbf{else}\ c'_2$, and it remains to show that $\mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ c'\,] \rightsquigarrow \textbf{if}\ b\ \textbf{then}\ \mathbf{S}[\,c'\,]\ \textbf{else}\ c'_2$ and $\langle \textbf{if}\ b\ \textbf{then}\ \mathbf{S}[\,c'\,]\ \textbf{else}\ c'_2, \sigma\rangle \longrightarrow \langle c'_2, \sigma\rangle$. From (b), using Lemma 3.41, we have
(g) $\textbf{if}\ b\ \textbf{then}\ \mathbf{S}[\,c'\,]\ \textbf{else}\ \mathbf{S}[\,c\,] \rightsquigarrow \textbf{if}\ b\ \textbf{then}\ \mathbf{S}[\,c'\,]\ \textbf{else}\ c'_2$. It is not hard to show that (h) $\mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ c'\ \textbf{else}\ c\,] \rightsquigarrow \textbf{if}\ b\ \textbf{then}\ \mathbf{S}[\,c'\,]\ \textbf{else}\ \mathbf{S}[\,c\,]$. From (h) and (g), using Lemma 3.35, we have (i) $\mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ c'\ \textbf{else}\ c\,] \rightsquigarrow \textbf{if}\ b\ \textbf{then}\ \mathbf{S}[\,c'\,]\ \textbf{else}\ c'_2$. From (i), (e), and the semantics; we conclude

- We consider (c) $c_1 = \mathbf{S}[\,\textbf{while}\ b\ \textbf{do}\ c\,]$, (d) $c'_1 = \mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ (c; \textbf{while}\ b\ \textbf{do}\ c)\ \textbf{else}\ \textbf{skip}\,]$, and (e) $\sigma = \sigma'$. We instantiate the goal with $\textbf{skip}; c'_2$, and it remains to show that $\mathbf{S}[\,\textbf{while}\ b\ \textbf{do}\ c\,] \rightsquigarrow \textbf{skip}; c'_2$ and $\langle \textbf{skip}; c'_2, \sigma\rangle \longrightarrow \langle c'_2, \sigma\rangle$. From the weaker notion of equality, we know that (f) $\textbf{skip}; c'_2 = c'_2$. It is not hard to show that (g) $\textbf{while}\ b\ \textbf{do}\ c \rightsquigarrow \textbf{if}\ b\ \textbf{then}\ (c; \textbf{while}\ b\ \textbf{do}\ c)\ \textbf{else}\ \textbf{skip}$. From (g), using Lemma 3.36, we have (h) $\mathbf{S}[\,\textbf{while}\ b\ \textbf{do}\ c\,] \rightsquigarrow \mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ (c; \textbf{while}\ b\ \textbf{do}\ c)\ \textbf{else}\ \textbf{skip}\,]$. From (b) and (h), we have (i) $\mathbf{S}[\,\textbf{if}\ b\ \textbf{then}\ (c; \textbf{while}\ b\ \textbf{do}\ c)\ \textbf{else}\ \textbf{skip}\,] \rightsquigarrow \textbf{skip}; c'_2$ From (h) and (i), using Lemma 3.35, we have (j) $\mathbf{S}[\,\textbf{while}\ b\ \textbf{do}\ c\,] \rightsquigarrow \textbf{skip}; c'_2$ From (j), and the semantics, we conclude □

Lemma 3.48 extends Lemma 3.47 to handle thread trees and the interleaved semantics. It is a bit more general, taking a 0-atomic thread tree context $\mathbf{T}$, so that we can have a more powerful induction hypothesis.

Lemma 3.48. If $\langle T_1, \sigma\rangle \longmapsto \langle T_1', \sigma'\rangle$, and $T_1' \rightsquigarrow_{\mathsf{t}} T_2'$, then there exists T_2 such that $T_1 \rightsquigarrow_{\mathsf{t}} T_2$ and, for all 0-atomic $\mathbf{T}$, we have $\langle \mathbf{T}[\,T_2\,], \sigma\rangle \longmapsto \langle \mathbf{T}[\,T_2'\,], \sigma'\rangle$.

Proof. By structural induction over T_1, assuming (a) $\langle T_1, \sigma\rangle \longmapsto \langle T_1', \sigma'\rangle$ and (b) $T_1' \rightsquigarrow_{\mathsf{t}} T_2'$, have 3 cases:

- We consider (c) $T_1 = c_1$. From (a), (c), and the semantics, we have 3 cases:
  - We consider (d) $T_1' = c_1'$ and (e) $\langle c_1, \sigma\rangle \longrightarrow \langle c_1', \sigma'\rangle$. From (b) and (d), by Def. 3.33 and Def. 3.23, we know (f) $T_2' = c_2'$. From (e), (d), (f), and (b), using Lemma 3.47, we know there exists c_2 such that (g) $c_1 \rightsquigarrow c_2$ and (h) $\langle c_2, \sigma\rangle \longrightarrow \langle c_2', \sigma'\rangle$. From (g) and (d), by Def. 3.33 and Def. 3.23, we know (i) $T_1 \rightsquigarrow_{\mathsf{t}} T_2$ where (j) $T_2 = c_2$. From (h), (j), (f), and the semantics, we know (k) $\langle \mathbf{T}[\,T_2\,], \sigma\rangle \longmapsto \langle \mathbf{T}[\,T_2'\,], \sigma'\rangle$. Instantiating the goal with T_2, from (i) and (k), we conclude
  - We consider (d) $c_1 = c_1' \parallel c_1''$, (e) $T_1' = \langle\!\langle c_1', c_1''\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}$, and (f) $\sigma = \sigma'$. From (b) and (e), by Def. 3.33 and Def. 3.23, we know (g) $T_2' = \langle\!\langle c_2', c_2''\rangle\!\rangle_{\mathsf{p}} c$ where (h) $c_1' \rightsquigarrow c_2'$, (i) $c_1'' \rightsquigarrow c_2''$, and (j) $\mathbf{skip} \rightsquigarrow c$. From (h), using Lemma 3.43, we know (k) $c_1' \parallel c_1'' \rightsquigarrow c_2' \parallel c_1''$. From (i), using Lemma 3.44, we know (l) $c_2' \parallel c_1'' \rightsquigarrow c_2' \parallel c_2''$. From (k) and (l), using Lemma 3.35, we know (m) $c_1' \parallel c_1'' \rightsquigarrow c_2' \parallel c_2''$. From (j), using Lemma 3.39, we know (n) $(c_1' \parallel c_1''); \mathbf{skip} \rightsquigarrow (c_1' \parallel c_1''); c$. From (m), using Lemma 3.38, we know (o) $(c_1' \parallel c_1''); c \rightsquigarrow (c_2' \parallel c_2''); c$. From (n) and (o), using Lemma 3.35, we know (p) $(c_1' \parallel c_1''); \mathbf{skip} \rightsquigarrow (c_2' \parallel c_2''); c$. From the semantics, we know (q) $\langle \mathbf{T}[\,(c_2' \parallel c_2''); c\,], \sigma\rangle \longmapsto \langle \mathbf{T}[\langle\!\langle c_2', c_2''\rangle\!\rangle_{\mathsf{p}}(\mathbf{skip}; c)\,], \sigma\rangle$. From the weaker notion of equality, we know that (r) $\mathbf{skip}; c = c$. Instantiating the goal with $(c_2' \parallel c_2''); c$, from (p), (q), and (r), we conclude
  - We consider (d) $c_1 = \mathbf{atomic}\ c_1'$, (e) $T_1' = \langle\!\langle c_1'\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}$, and (f) $\sigma = \sigma'$. From (b) and (e), by Def. 3.33 and Def. 3.23, we know (g) $T_2' = \langle\!\langle c_2'\rangle\!\rangle_{\mathsf{a}} c$ where (h) $c_1' \rightsquigarrow c_2'$ and (i) $\mathbf{skip} \rightsquigarrow c$. From (h), using Lemma 3.45, we know (j) $\mathbf{atomic}\ c_1' \rightsquigarrow \mathbf{atomic}\ c_2'$.

From (i), using Lemma 3.39, we know (k) $(\textbf{atomic}\ c_1'); \textbf{skip} \rightsquigarrow (\textbf{atomic}\ c_1'); c$. From (j), using Lemma 3.38, we know (l) $(\textbf{atomic}\ c_1'); c \rightsquigarrow (\textbf{atomic}\ c_2'); c$. From (k) and (l), using Lemma 3.35, we know (m) $(\textbf{atomic}\ c_1'); \textbf{skip} \rightsquigarrow (\textbf{atomic}\ c_2'); c$. From the semantics we know (n) $\langle \mathbf{T}[(\textbf{atomic}\ c_2'); c], \sigma\rangle \longmapsto \langle \mathbf{T}[\langle\!\langle c_2' \rangle\!\rangle_{\mathbf{a}}(\textbf{skip}; c)], \sigma\rangle$. From the weaker notion of equality, we know that (o) $\textbf{skip}; c = c$. Instantiating the goal with $(\textbf{atomic}\ c_2'); c$, from (m), (n), and (o), we conclude

- We consider (c) $T_1 = \langle\!\langle T_1'', T_1''' \rangle\!\rangle_{\mathbf{p}} c_1$. From (a), (c), and the semantics, we have 3 cases:
  - We consider (d) $T_1'' = \textbf{skip}$, (e) $T_1''' = \textbf{skip}$, (f) $T_1' = c_1$, (g) and $\sigma = \sigma'$. From (b) and (f), by Def. 3.33 and Def. 3.23, we know (h) $T_2' = c_2$ and (i) $c_1 \rightsquigarrow c_2$. Using Lemma 3.34, we know (j) $\textbf{skip} \rightsquigarrow \textbf{skip}$. From (j) and (i), by Def. 3.33 and Def. 3.23, we know (k) $\langle\!\langle \textbf{skip}, \textbf{skip} \rangle\!\rangle_{\mathbf{p}} c_1 \rightsquigarrow_{\mathbf{t}} \langle\!\langle \textbf{skip}, \textbf{skip} \rangle\!\rangle_{\mathbf{p}} c_2$ From the semantics, we know (l) $\langle \mathbf{T}[\,\langle\!\langle \textbf{skip}, \textbf{skip} \rangle\!\rangle_{\mathbf{p}} c_2\,], \sigma\rangle \longmapsto \langle \mathbf{T}[\,c_2\,], \sigma\rangle$. Instantiating the goal with $\langle\!\langle \textbf{skip}, \textbf{skip} \rangle\!\rangle_{\mathbf{p}} c_2$, from (k) and (l), we conclude
  - We consider (d) $T_1' = \langle\!\langle T_1'''', T_1''' \rangle\!\rangle_{\mathbf{p}} c_1$ and (e) $\langle T_1'', \sigma\rangle \longmapsto \langle T_1'''', \sigma'\rangle$. From (b) and (d), by Def. 3.33 and Def. 3.23, we know (f) $T_2' = \langle\!\langle T_2'''', T_2''' \rangle\!\rangle_{\mathbf{p}} c_2$ where (g) $T_1'''' \rightsquigarrow_{\mathbf{t}} T_2''''$, (h) $T_1''' \rightsquigarrow_{\mathbf{t}} T_2'''$, and (i) $c_1 \rightsquigarrow c_2$. From (e) and (g), using the induction hypothesis (with context $\mathbf{T}[\,\langle\!\langle \bullet, T_2''' \rangle\!\rangle_{\mathbf{p}} c_2\,]$), we know there exists T_2'' such that (j) $T_1'' \rightsquigarrow_{\mathbf{t}} T_2''$ and (k) $\langle \mathbf{T}[\,\langle\!\langle T_2'', T_2''' \rangle\!\rangle_{\mathbf{p}} c_2\,], \sigma\rangle \longmapsto \langle \mathbf{T}[\,\langle\!\langle T_2'''', T_2''' \rangle\!\rangle_{\mathbf{p}} c_2\,], \sigma'\rangle$. From (j), (h), and (i), by Def. 3.33 and Def. 3.23, we know (l) $\langle\!\langle T_1'', T_1''' \rangle\!\rangle_{\mathbf{p}} c_1 \rightsquigarrow_{\mathbf{t}} \langle\!\langle T_2'', T_2''' \rangle\!\rangle_{\mathbf{p}} c_2$. Instantiating the goal with $\langle\!\langle T_2'', T_2''' \rangle\!\rangle_{\mathbf{p}} c_2$, from (l) and (k), we conclude
  - We consider (d) $T_1' = \langle\!\langle T_1'', T_1'''' \rangle\!\rangle_{\mathbf{p}} c_1$ and (e) $\langle T_1''', \sigma\rangle \longmapsto \langle T_1'''', \sigma'\rangle$. From (b) and (d), by Def. 3.33 and Def. 3.23, we know (f) $T_2' = \langle\!\langle T_2'', T_2'''' \rangle\!\rangle_{\mathbf{p}} c_2$ where (g) $T_1'' \rightsquigarrow_{\mathbf{t}} T_2''$, (h) $T_1'''' \rightsquigarrow_{\mathbf{t}} T_2''''$, and (i) $c_1 \rightsquigarrow c_2$. From (e) and (h), using the induction hypothesis (with context $\mathbf{T}[\,\langle\!\langle T_2'', \bullet \rangle\!\rangle_{\mathbf{p}} c_2\,]$), we know there exists T_2''' such that (j) $T_1''' \rightsquigarrow_{\mathbf{t}} T_2'''$ and (k) $\langle \mathbf{T}[\,\langle\!\langle T_2'', T_2''' \rangle\!\rangle_{\mathbf{p}} c_2\,], \sigma\rangle \longmapsto \langle \mathbf{T}[\,\langle\!\langle T_2''', T_2'''' \rangle\!\rangle_{\mathbf{p}} c_2\,], \sigma'\rangle$. From (g), (j), and (i), by Def. 3.33 and Def. 3.23, we know (l) $\langle\!\langle T_1'', T_1''' \rangle\!\rangle_{\mathbf{p}} c_1 \rightsquigarrow_{\mathbf{t}} \langle\!\langle T_2'', T_2''' \rangle\!\rangle_{\mathbf{p}} c_2$. Instantiating the goal with $\langle\!\langle T_2'', T_2''' \rangle\!\rangle_{\mathbf{p}} c_2$, from (l) and (k), we conclude

- We consider (c) $T_1 = \langle\!\langle T_1'' \rangle\!\rangle_{\mathbf{a}} c_1$. From (a), (c), and the semantics, we have 2 cases:
  - We consider (d) $T_1'' = \mathbf{skip}$, (e) $T_1' = c_1$, (f) and $\sigma = \sigma'$. From (b) and (e), by Def. 3.33 and Def. 3.23, we know (g) $T_2' = c_2$ and (h) $c_1 \rightsquigarrow c_2$. Using Lemma 3.34, we know (i) $\mathbf{skip} \rightsquigarrow \mathbf{skip}$. From (i) and (h), by Def. 3.33 and Def. 3.23, we know (j) $\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathbf{a}} c_1 \rightsquigarrow_{\mathbf{t}} \langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathbf{a}} c_2$ From the semantics, we know (k) $\langle \mathbf{T}[\, \langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathbf{a}} c_2 \,], \sigma \rangle \longmapsto \langle \mathbf{T}[\, c_2 \,], \sigma \rangle$. Instantiating the goal with $\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathbf{a}} c_2$, from (j) and (k), we conclude
  - We consider (d) $T_1' = \langle\!\langle T_1''' \rangle\!\rangle_{\mathbf{a}} c_1$ and (e) $\langle T_1'', \sigma \rangle \longmapsto \langle T_1''', \sigma' \rangle$. From (b) and (d), by Def. 3.33 and Def. 3.23, we know (f) $T_2' = \langle\!\langle T_2''' \rangle\!\rangle_{\mathbf{p}} c_2$ where (g) $T_1''' \rightsquigarrow_{\mathbf{t}} T_2'''$, and (h) $c_1 \rightsquigarrow c_2$. From (e) and (g), using the induction hypothesis (with context $\mathbf{T}[\, \langle\!\langle \bullet \rangle\!\rangle_{\mathbf{a}} c_2 \,]$), we know there exists T_2'' such that (i) $T_1'' \rightsquigarrow_{\mathbf{t}} T_2''$ and (j) $\langle \mathbf{T}[\langle\!\langle T_2'' \rangle\!\rangle_{\mathbf{a}} c_2], \sigma \rangle \longmapsto \langle \mathbf{T}[\langle\!\langle T_2''' \rangle\!\rangle_{\mathbf{a}} c_2], \sigma' \rangle$. From (i) and (h), by Def. 3.33 and Def. 3.23, we know (k) $\langle\!\langle T_1'' \rangle\!\rangle_{\mathbf{a}} c_1 \rightsquigarrow_{\mathbf{t}} \langle\!\langle T_2'' \rangle\!\rangle_{\mathbf{a}} c_2$. Instantiating the goal with $\langle\!\langle T_2'' \rangle\!\rangle_{\mathbf{a}} c_2$, from (k) and (j), we conclude $\square$

Chapter 4

Relaxed Semantics: Examples

There are different aspects that characterize a particular memory model, including memory ordering constraints, support for general compiler transformations, write atomicity constraints, presence of write buffers, cache coherence protocols, availability of memory barriers, etc. In this section, we show how some of these aspects are reflected in our semantics. Our goal is to familiarize the reader with the $\rightsquigarrow$ relation. The examples are written using the following naming convention: `v1`, `v2`, `v3`, etc, are variables that hold values; `x`, `y`, `z`, etc, are variables that hold memory addresses.

Data dependencies

Before we discuss the memory ordering of our model, we need to make it clear that we can support precise discovery of data dependencies. We do it by showing the following example:

$$[\mathtt{x}] := 1; \mathtt{v1} := [\mathtt{y}]$$

In this small program, the data dependency between the two statements exists only for those initial states where `x` and `y` are aliased. At a first glance, our $\rightsquigarrow$ definition is too restrictive since its definition quantifies over *all* input states. So it does not allow the following:

$$([\mathtt{x}] := 1; \mathtt{v1} := [\mathtt{y}]) \not\rightsquigarrow (\mathtt{v1} := [\mathtt{y}]; [\mathtt{x}] := 1)$$

However, through the $\rightsquigarrow$ relation, we can obtain the following transformation:

$$[\mathtt{x}] := 1; \mathtt{v1} := [\mathtt{y}] \rightsquigarrow \textbf{if } \mathtt{x} = \mathtt{y} \textbf{ then } ([\mathtt{x}] := 1; \mathtt{v1} := [\mathtt{x}]) \textbf{ else } (\mathtt{v1} := [\mathtt{y}]; [\mathtt{x}] := 1)$$

where we insert a dynamic test to see if x is aliased to y. We also replace y by x in one branch where there is dependency, and reorder the statements in the other branch. Based on this example, one can convince himself that the relaxed semantics will allow the reordering memory accesses that do not have data dependencies at runtime. But, the reader should also be aware that the $\rightsquigarrow$ relation does not violate data dependencies, such as the ones shown below:

$$\mathtt{v1} := [\mathtt{x}]; [\mathtt{x}] := 2 \not\rightsquigarrow [\mathtt{x}] := 2; \mathtt{v1} := [\mathtt{x}]$$
$$[\mathtt{x}] := 1; \mathtt{v2} := [\mathtt{x}] \not\rightsquigarrow \mathtt{v2} := [\mathtt{x}]; [\mathtt{x}] := 1$$
$$[\mathtt{x}] := 1; [\mathtt{x}] := 2 \quad \not\rightsquigarrow \quad [\mathtt{x}] := 2; [\mathtt{x}] := 1$$

Memory ordering

From the example just shown, it is not hard to see that the $\rightsquigarrow$ relation supports all 4 types of memory reordering (R,W $\rightarrow$ R,W). Examples of this can be seen below (we use the context (**if** $\mathtt{x} = \mathtt{y}$ **then skip else** $\bullet$) but these are supported in any context where $\mathtt{x} \neq \mathtt{y}$ can be inferred):

- Reads with reads:

$$\textbf{if } \mathtt{x} = \mathtt{y} \textbf{ then skip else } (\mathtt{v1} := [\mathtt{x}]; \mathtt{v2} := [\mathtt{y}])$$
$$\rightsquigarrow$$
$$\textbf{if } \mathtt{x} = \mathtt{y} \textbf{ then skip else } (\mathtt{v2} := [\mathtt{y}]; \mathtt{v1} := [\mathtt{x}])$$

- Reads with writes:

$$\textbf{if } \mathtt{x} = \mathtt{y} \textbf{ then skip else } (\mathtt{v1} := [\mathrm{x}];\ [\mathrm{y}] := 2)$$
$$\rightsquigarrow$$
$$\textbf{if } \mathtt{x} = \mathtt{y} \textbf{ then skip else } ([\mathrm{y}] := 2;\ \mathtt{v1} := [\mathrm{x}])$$

- Writes with reads:

$$\textbf{if } \mathtt{x} = \mathtt{y} \textbf{ then skip else } ([\mathrm{x}] := 1;\ \mathtt{v2} := [\mathrm{y}])$$
$$\rightsquigarrow$$
$$\textbf{if } \mathtt{x} = \mathtt{y} \textbf{ then skip else } (\mathtt{v2} := [\mathrm{y}];\ [\mathrm{x}] := 1)$$

- Writes with writes:

$$\textbf{if } \mathtt{x} = \mathtt{y} \textbf{ then skip else } ([\mathrm{x}] := 1;\ [\mathrm{y}] := 2)$$
$$\rightsquigarrow$$
$$\textbf{if } \mathtt{x} = \mathtt{y} \textbf{ then skip else } ([\mathrm{y}] := 2;\ [\mathrm{x}] := 1)$$

Write buffer with read bypassing

A write buffer is a hardware feature that delays writes to memory in an attempt to overlap the latency of writes with subsequent code. The actual behavior obtained is that a processor might read its own writes earlier, i.e. before they are actually committed to memory. This can be supported by a simple program transformation as seen in the example below:

$$[\mathrm{x}] := 1;\ \mathtt{v2} := [\mathrm{x}] \rightsquigarrow \mathtt{v2} := 1;\ [\mathrm{x}] := 1$$

Redundancy introduction and elimination

Redundant memory reads and writes can be introduced and eliminated, as shown by the following examples:

$$\begin{aligned}
\texttt{v1} := [\texttt{x}]; \texttt{v2} := 1 &\leadsto \texttt{v1} := [\texttt{x}]; \texttt{v2} := [\texttt{x}]; \texttt{v2} := 1\\
\texttt{v1} := [\texttt{x}]; \texttt{v2} := [\texttt{x}] &\leadsto \texttt{v1} := [\texttt{x}]; \texttt{v2} := \texttt{v1}\\
[\texttt{x}] := \texttt{v1}; \texttt{v2} := [\texttt{x}] &\leadsto [\texttt{x}] := \texttt{v1}; \texttt{v2} := \texttt{v1}\\
[\texttt{x}] := \texttt{v1} &\leadsto [\texttt{x}] := 1; [\texttt{x}] := \texttt{v1}\\
[\texttt{x}] := 1; [\texttt{x}] := \texttt{v1} &\leadsto [\texttt{x}] := \texttt{v1}
\end{aligned}$$

Furthermore, we can eliminate dead memory operations when that yields a smaller memory footprint:

$$\texttt{v1} := [\texttt{x}]; \texttt{v1} := 1 \leadsto \texttt{v1} := 1$$

Note that the reverse is not true. A program can only decrease its footprint given the $\leadsto$ relation.

$$\texttt{v1} := 1 \not\leadsto \texttt{v1} := [\texttt{x}]; \texttt{v1} := 1$$

Write atomicity

Given the $\leadsto$ relation, write atomicity is not preserved. This might not be clear at first, but can be shown in the example below (here `v1` is a temporary, we assume it is reused later on, by the artifact of assigning an arbitrary value to it):

$$\begin{gathered}
[\texttt{x}] := 1; \texttt{v1} := 42\\
\leadsto\\
\texttt{v1} := [\texttt{x}]; [\texttt{x}] := 1; [\texttt{x}] := \texttt{v1}; [\texttt{x}] := 1; \texttt{v1} := 42
\end{gathered}$$

Here the write is replaced by 3 writes, oscillating between the original write and the write of the initial value into the memory location. In fact, it might oscillate with any value (not only the initial value) as shown below:

$$[\mathtt{x}] := 1$$

$$\leadsto$$

$$[\mathtt{x}] := 1;\, [\mathtt{x}] := 42;\, [\mathtt{x}] := 1;\, [\mathtt{x}] := 69;\, [\mathtt{x}] := 1$$

therefore, a write can store arbitrary values to memory before completing; which in practice means that the memory value is undefined until the write completes.

Compiler optimizations

The $\leadsto$ relation is general enough to support many sequential compiler optimizations. For instance, it is not hard to see that we can support instruction scheduling

$$\mathtt{v3} := \mathtt{v1} + \mathtt{v2};\, \mathtt{v6} := \mathtt{v4} + \mathtt{v5};\, \mathtt{v7} := \mathtt{v3} + \mathtt{v6}$$

$$\leadsto$$

$$\mathtt{v6} := \mathtt{v4} + \mathtt{v5};\, \mathtt{v3} := \mathtt{v1} + \mathtt{v2};\, \mathtt{v7} := \mathtt{v3} + \mathtt{v6}$$

algebraic transformations (here we assume `v4` is a temporary)

$$\mathtt{v4} := \mathtt{v1} + \mathtt{v2};\, \mathtt{v5} := \mathtt{v4} + \mathtt{v3};\, \mathtt{v4} := 42$$

$$\leadsto$$

$$\mathtt{v4} := \mathtt{v2} + \mathtt{v3};\, \mathtt{v5} := \mathtt{v1} + \mathtt{v4};\, \mathtt{v4} := 42$$

register allocation (we have to test for aliasing of z and w)

$$\begin{gathered}\texttt{v1}:=[\mathrm{x}];\,\texttt{v2}:=[\mathrm{y}];\,\texttt{v3}:=\texttt{v1}+\texttt{v2};\,[\texttt{w}]:=\texttt{v3};\\ \texttt{v1}:=[\mathrm{z}];\,\texttt{v2}:=[\texttt{w}];\,\texttt{v3}:=\texttt{v1}+\texttt{v2};\,[\texttt{w}]:=\texttt{v3}\\ \rightsquigarrow\\ \texttt{v1}:=[\mathrm{x}];\,\texttt{v2}:=[\mathrm{y}];\,\texttt{v3}:=\texttt{v1}+\texttt{v2};\\ (\textbf{if}\ \texttt{z}=\texttt{w}\ \textbf{then}\ \texttt{v1}:=\texttt{v3}\ \textbf{else}\ \texttt{v1}:=[\mathrm{z}]);\\ \texttt{v2}:=\texttt{v3};\,\texttt{v3}:=\texttt{v1}+\texttt{v2};\,[\texttt{w}]:=\texttt{v3}\end{gathered}$$

and many others, including control transformations and redundancy elimination such as the ones already presented in Sec. 4.

Concurrent behaviors

Here we present some concurrent behaviors of the semantics yielded by $\rightsquigarrow$. In all examples, we assume a sequential interleaving of commands according to the standard semantics after considering a program transformation through the $\rightsquigarrow$ relation. We also assume initial memory values are all 0, unless otherwise noted.

We start with the following example (not supported by Boudol and Petri [15])

$$(\texttt{v1}:=[\mathrm{x}];\,[\mathrm{y}]:=1)\ \ \|\ \ (\texttt{v2}:=[\mathrm{y}];\,[\mathrm{x}]:=1)$$

in which we can perceive $\texttt{v1} = \texttt{v2} = 1$ if $\texttt{x} \neq \texttt{y}$. It can be supported in our semantics by reordering the commands in the second thread,

$$\begin{gathered}\texttt{v2}:=[\mathrm{y}];\,[\mathrm{x}]:=1\\ \rightsquigarrow\\ \textbf{if}\ \texttt{x}=\texttt{y}\ \textbf{then}\ (\texttt{v2}:=[\mathrm{x}];\,[\mathrm{x}]:=1)\ \textbf{else}\ ([\mathrm{x}]:=1;\,\texttt{v2}:=[\mathrm{y}])\end{gathered}$$

yielding a new program that produces the desired result through an interleaved scheduling.

Similarly, we can support the classic crossover example:

$$([\mathtt{x}] := 1; \mathtt{v1} := [\mathtt{x}]) \;\; || \;\; ([\mathtt{x}] := 2; \mathtt{v2} := [\mathtt{x}])$$

in which we can perceive $\mathtt{v1} = 2$ and $\mathtt{v2} = 1$. That is achieved by inserting a redundant write in the right hand side thread:

$$[\mathtt{x}] := 2; \mathtt{v2} := [\mathtt{x}] \rightsquigarrow [\mathtt{x}] := 2; \mathtt{v2} := [\mathtt{x}]; [\mathtt{x}] := 2$$

Yet another similar example is the prescient write test:

$$(\mathtt{v1} := [\mathtt{x}]; [\mathtt{x}] := 1) \;\; || \;\; (\mathtt{v2} := [\mathtt{x}]; [\mathtt{x}] := \mathtt{v2})$$

where we could perceive $\mathtt{v1} = \mathtt{v2} = 1$. That is also supported by inserting a redundant write and a redundant read in the right hand side thread:

$$\mathtt{v2} := [\mathtt{x}]; [\mathtt{x}] := \mathtt{v2} \rightsquigarrow \mathtt{v2} := [\mathtt{x}]; [\mathtt{x}] := 1; [\mathtt{x}] := \mathtt{v2}; \mathtt{v2} := [\mathtt{x}]$$

As one can seen, the semantics derived from $\rightsquigarrow$ leads to possibly unwanted behaviors of raceful programs. First, a read from a shared location can return any value. For instance, there is a scheduling of the program below:

$$\mathtt{v1} := [\mathtt{x}] \;\; || \;\; [\mathtt{x}] := 1$$

where $\mathtt{v1} = 33$ is allowed. That happens if we consider the following replacement of the right hand side thread:

$$[\mathtt{x}] := 1 \rightsquigarrow [\mathtt{x}] := 33; [\mathtt{x}] := 1$$

This is commonly referred to as “out-of-thin-air” behavior.

A similar behavior happens when we have simultaneous write to the same location:

$$(\texttt{v1} := 1; [x] := \texttt{v1}) \;\;\|\;\; [\texttt{x}] := 2$$

in this case, the final value of [x] can also be arbitrary. For instance, it could be 3 if we replace the left hand side thread as below

$$\texttt{v1} := 1; [\texttt{x}] := \texttt{v1} \rightsquigarrow [\texttt{x}] := 0; \texttt{v1} := [\texttt{x}]; \texttt{v1} := \texttt{v1}+1; [\texttt{x}] := \texttt{v1}$$

Another unwanted behavior happens when the implementations of mutual exclusions rely on memory ordering (such as the core of Dekker's algorithm presented in Chapter 1):

$$([\texttt{x}] := 1; \texttt{v1} := [\texttt{y}]) \;\;\|\;\; ([\texttt{y}] := 1; \texttt{v2} := [\texttt{x}])$$

In this case, we would not want the behavior $\texttt{v1} = \texttt{v2} = 0$ to happen. However, it may happen if we consider the reordering of the two commands of the right hand side thread:

$$\begin{array}{c} [\texttt{y}] := 1; \texttt{v2} := [\texttt{x}] \\ \rightsquigarrow \\ \textbf{if } \texttt{x} = \texttt{y} \textbf{ then } ([\texttt{x}] := 1; \texttt{v2} := [\texttt{x}]) \textbf{ else } (\texttt{v2} := [\texttt{x}]; [\texttt{y}] := 1) \end{array}$$

Note that, naturally, we are assuming initial values $[\texttt{x}] = [\texttt{y}] = 0$ and $\texttt{x} \neq \texttt{y}$.

Many other examples of raceful code can be shown to have unwanted behaviors in such a relaxed execution. They are obtained by either reordering of memory operations or relying on the non-atomic undefined nature of raceful reads and writes. On the other hand, race-free programs do not have unwanted behaviors (see the DRF-guarantee in Chapter 5). In the example below:

$$\begin{pmatrix} \texttt{v1} := [\texttt{x}]; \\ \textbf{if } \texttt{v1} = 1 \textbf{ then } [\texttt{y}] := 1 \end{pmatrix} \;\;\|\;\; \begin{pmatrix} \texttt{v2} := [\texttt{y}]; \\ \textbf{if } \texttt{v2} = 1 \textbf{ then } [\texttt{x}] := 1 \end{pmatrix}$$

the only behavior allowed is $\texttt{v1} = \texttt{v2} = 0$. Its data-race-freedom might not be obvious, but there are no sequentially consistent executions of this program that may reach the code within the branches (again, assuming $[\texttt{x}] = [\texttt{y}] = 0$ and $\texttt{x} \neq \texttt{y}$ initially). So, the program never issues a memory write, therefore it is race-free. And, if you consider the code of each one of the threads in isolation — through the $\rightsquigarrow$ relation — it is impossible to insert a race when the initial state has $[\texttt{x}] = [\texttt{y}] = 0$. That is guaranteed from the fact that the footprints of both threads are disjoint, and they can only decrease through the $\rightsquigarrow$ relation.

In order to preserve the DRF-property, subsumption must disallow some specific types of transformations such as write speculation as show below

$$\left(\begin{array}{l}\texttt{v1}:=[\texttt{x}];\\ \textbf{if } \texttt{v1}=1 \textbf{ then } [\texttt{x}]:=2\end{array}\right) \not\rightsquigarrow \left(\begin{array}{l}\texttt{v}:=[\texttt{x}];\\ [\texttt{x}]:=2;\\ \textbf{if } \texttt{v1}\neq 1 \textbf{ then } [\texttt{x}]:=\texttt{v1}\end{array}\right)$$

in which the footprint of the program is actually larger after the transformation; given that if $\texttt{v1} \neq 1$ the original program does not write to x, while the transformed program always write to x. Since the footprint has increased, this program transformation might produce races if the original code was used in the context of a DRF-program.

Strong barrier

In our relaxed semantics, we can enforce both atomicity and ordering by using the **atomic** c command. In the following examples we use the macro MF (memory fence) as a syntactic sugar for **atomic skip**, a command that does nothing but enforcing the ordering.

The first example we analyze is about cache coherence. Cache coherence ensures that everybody agrees on the order of writes to the same location. Since the $\rightsquigarrow$ relation does not preserve the atomicity of writes, coherence is not preserved by the semantics, as can be seen in the following example:

$$[\mathtt{x}] := 1 \;\; \| \;\; [\mathtt{x}] := 2 \;\; \| \;\; \begin{pmatrix} \mathtt{v1} := [\mathtt{x}]; \\ \mathsf{MF}; \\ \mathtt{v2} := [\mathtt{x}] \end{pmatrix} \;\; \| \;\; \begin{pmatrix} \mathtt{v3} := [\mathtt{x}]; \\ \mathsf{MF}; \\ \mathtt{v4} := [\mathtt{x}] \end{pmatrix}$$

in which the outcome $\mathtt{v1} = \mathtt{v4} = 1$ and $\mathtt{v2} = \mathtt{v3} = 2$ can be noticed once we rewrite the leftmost thread as

$$[\mathtt{x}] := 1 \rightsquigarrow [\mathtt{x}] := 1; [\mathtt{x}] := 1$$

Another related example is the independent-reads-independent-writes (IRIW) example shown below

$$[\mathtt{x}] := 1 \;\; \| \;\; [\mathtt{y}] := 1 \;\; \| \;\; \begin{pmatrix} \mathtt{v1} := [\mathtt{x}]; \\ \mathsf{MF}; \\ \mathtt{v2} := [\mathtt{y}] \end{pmatrix} \;\; \| \;\; \begin{pmatrix} \mathtt{v3} := [\mathtt{y}]; \\ \mathsf{MF}; \\ \mathtt{v4} := [\mathtt{x}] \end{pmatrix}$$

where the behavior $\mathtt{v1} = \mathtt{v3} = 1$ and $\mathtt{v2} = \mathtt{v4} = 0$ is permissible (again assuming $[\mathtt{x}] = [\mathtt{y}] = 0$ initially). That can be perceived in our semantics once we replace the leftmost thread through

$$[\mathtt{x}] := 1 \rightsquigarrow [\mathtt{x}] := 1; [\mathtt{x}] := 0; [\mathtt{x}] := 1$$

Other similar examples shown by [12] can also be supported. It might be intuitive that they happen because $\rightsquigarrow$ does not enforce write atomicity.

Chapter 5

Proof of Data-Race-Free Guarantee

In this chapter, we prove that the relaxed semantics yield by subsumption, as defined in Sec. 3.10, provides the so called data-race-free (DRF) guarantee.

The DRF-guarantee is a property of a concurrent semantics. Informally, it ensures that for any program that is absent of data-races its execution according to the given concurrent semantics should be equivalent to an execution according to a standard interleaved semantics. This is an important guarantee as reasoning about a data-race free program in the given semantics is no different from the reasoning about it in the interleaved semantics.

In the next section, we will discuss possible definitions for the DRF-guarantee, and present our choice. We will also present a mixed-step version of the small-step interleaved semantics of Sec. 3.7 which works as an intermediate semantics for the proofs.

5.1 DRF-guarantee Definition

As we briefly explained, the DRF-guarantee is a property of a concurrent semantics. In our setting, we will abuse of the term by associating the DRF-guarantee property to a given Λ relation. As we know, we can obtain different concurrent semantics by instantiating the parameterized semantics of Sec. 3.8 with different Λ's. Therefore, when we say that a given Λ provides the DRF-guarantee; we actually mean that its instantiation has the property.

Using the semantic framework of Chapter 3 we can easily define the absence of data-races of a program configuration κ as:

$$\neg\kappa \longmapsto^* \mathsf{race}$$

which is defined in terms of the interleaved semantics. We also know how to define program equivalence between a given concurrent semantics and the interleaved semantics:

$$[\Lambda]\ \kappa \longmapsto^* \langle \mathbf{skip}, \sigma \rangle \iff \kappa \longmapsto^* \langle \mathbf{skip}, \sigma \rangle$$

which is means that any complete execution starting from program configuration κ reaches the same set of states. Since we also want to preserve the safety of programs we should also require that safety is preserved:

$$\neg\kappa \longmapsto^* \mathsf{abort} \iff \neg[\Lambda]\ \kappa \longmapsto^* \mathsf{abort}$$

which means that if a program does not abort in the interleaved semantics it should not abort in the given concurrent semantics. At last, we can also have a similar requirement regarding program termination, if we are looking at equivalence from the point of view of total correctness:

$$\neg\kappa \longmapsto^\infty \iff \neg[\Lambda]\ \kappa \longmapsto^\infty$$

By combining the requirements above, We can now make our first attempt to define the DRF-guarantee. As it will be come clear soon, the following definition is too strong, thus we call it Strong DRF-guarantee.

Definition 5.1. A relation Λ provides a *Strong* DRF-guarantee, if and only if, for all κ, where $\neg\kappa \longmapsto^* \mathsf{race}$, the following hold:

1. $[\Lambda]\ \kappa \longmapsto^* \langle \mathbf{skip}, \sigma\rangle \iff \kappa \longmapsto^* \langle \mathbf{skip}, \sigma\rangle$

2. $\neg\kappa \longmapsto^* \mathsf{abort} \iff \neg[\Lambda]\ \kappa \longmapsto^* \mathsf{abort}$

3. $\neg\kappa \longmapsto^\infty \iff \neg[\Lambda]\ \kappa \longmapsto^\infty$

Naturally, the more requirements we place over the Λ-parameterized semantics, the less flexibility we have to apply program transformations. Therefore, we can argue that the Strong DRF-guarantee is too strong for practical reasons. Its first requirement, is only reasonable in one direction. Of course, we want the execution in the Λ-parameterized semantics to yield an outcome that is possible in the interleaved semantics. But the other way around is not necessary. Removing this constraint allows the Λ-parameterized semantics to be compatible-with but less deterministic than the interleaved semantics.

Moreover, just as we do not provide any guarantees for programs that are not DRF, we should not provide guarantees for programs that are not safe. This is reasonable. In many contexts, particularly in the context of formal methods, unsafe programs are plain bad. So why should the Λ-parameterized preserve their semantics? The answer is it should not.

Finally, we must discuss the termination requirement. It is reasonable that we require that if the program terminates in the interleaved semantics, it should also terminate in the Λ-parameterized semantics. So, only one direction is required, allowing the Λ-parameterized semantics to only exhibit terminating executions. This is good. However, although it might seem strange, our parameterized semantics should be allowed to turn terminating programs into diverging ones due to dynamic transformations. This can happen with subsumption, for instance. Imagine our current configuration is a final configuration $\langle \mathbf{skip}, \sigma\rangle$, then given the relaxed semantics yield by subsumption we can have $[\rightsquigarrow]\ \langle \mathbf{skip}, \sigma\rangle \longmapsto \langle \mathbf{skip}, \sigma\rangle$. This looks surprising, but it may happen if we consider this step occurring as the composition of $\mathbf{skip} \rightsquigarrow \mathbf{skip}; \mathbf{skip}$ and $\langle \mathbf{skip}; \mathbf{skip}, \sigma\rangle \longmapsto \langle \mathbf{skip}, \sigma\rangle$. It is not hard to show that all programs executed in this semantics may not terminate, because it is always possible to continue execution from a final configuration, i.e. $[\rightsquigarrow]\ \langle \mathbf{skip}, \sigma\rangle \longmapsto^\infty$ (note, however, that subsumption alone does preserve termination). This behavior is strange but plausible. Furthermore, in this thesis, we only study verification of partial correctness, and dropping termination requirements will not affect our findings. For these reasons, we will not impose requirements over divergence in our Weak DRF-guarantee presented below.

Definition 5.2. A relation Λ provides the *Weak* DRF-guarantee, if and only if, for all κ, where $\neg\kappa \longmapsto^* \textsf{race}$, and $\neg\kappa \longmapsto^* \textsf{abort}$, the following hold:

1. $\neg[\Lambda]\ \kappa \longmapsto^* \textsf{abort}$

2. If $[\Lambda]\ \kappa \longmapsto^* \langle \mathbf{skip}, \sigma\rangle$, then $\kappa \longmapsto^* \langle \mathbf{skip}, \sigma\rangle$

Through this text, when we refer to DRF-guarantee without further specification, we actually mean the Weak DRF-guarantee, as in Def. 5.2.

5.2 Interleaved Mixed-Step Semantics

In this section, we define the so called interleaved mixed-step semantics. This semantics is an intermediate semantics used to establish the DRF-guarantee of the $\rightsquigarrow$-parameterized semantics. It performs multiple operations of a given thread in a single step, but still interleaves operations from different threads at synchronization points (i.e. beginning and ending of atomic blocks and parallel composition).

The mixed-step semantics is defined in Fig. 5.1. It is structurally equivalent to the interleaved semantics presented in Sec.3.7, but with remarkable differences in the rules for command execution, command abortion, and race detection. In those rules, instead of a single sequential step ($\longrightarrow$), we make use of multiple sequential steps ($\longrightarrow^*$). In particular, command execution must evaluate completely to a synchronized operation ($\Downarrow$), and the evaluation must be non-empty (captured by the inequality $c \neq c'$) to avoid stuttering steps.

Next, we define divergence for the multi-step semantics. It can happen in two forms: either there is an infinite interleaving of big-steps; or the execution reaches a point where

$$
\begin{array}{rcll}
\langle \mathbf{T}[\,c\,], \sigma\rangle & \Longrightarrow & \textsf{abort} & \text{if } \langle c, \sigma\rangle \longrightarrow^* \textsf{abort} \\
\langle \mathbf{T}[\,c\,], \sigma\rangle & \Longrightarrow & \langle \mathbf{T}[\,c'\,], \sigma'\rangle & \text{if } \langle c, \sigma\rangle \Downarrow \langle c', \sigma'\rangle \wedge c \neq c' \\
\langle \mathbf{T}[\,\mathbf{S}[\,c_1 \,\|\, c_2\,]\,], \sigma\rangle & \Longrightarrow & \langle \mathbf{T}[\,\langle\!\langle c_1, c_2\rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,])\,], \sigma\rangle & \text{always} \\
\langle \mathbf{T}[\,\langle\!\langle \mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathbf{p}} c\,], \sigma\rangle & \Longrightarrow & \langle \mathbf{T}[\,c\,], \sigma\rangle & \text{always} \\
\langle \mathbf{T}[\,\mathbf{S}[\,\mathbf{atomic}\ c\,]\,], \sigma\rangle & \Longrightarrow & \langle \mathbf{T}[\,\langle\!\langle c\rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\mathbf{skip}\,])\,], \sigma\rangle & \text{if } \mathbf{T} \text{ is } 0\text{-atomic} \\
\langle \mathbf{T}[\,\langle\!\langle \mathbf{skip}\rangle\!\rangle_{\mathbf{a}} c\,], \sigma\rangle & \Longrightarrow & \langle \mathbf{T}[\,c\,], \sigma\rangle & \text{always} \\
\langle \mathbf{T}[\,\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,]\rangle\!\rangle_{\mathbf{p}} c\,], \sigma\rangle & \Longrightarrow & \textsf{race} & \text{if } \exists \delta_1, \delta_2, c_1', \sigma', \kappa. \\
 & & & \quad \langle c_1, \sigma\rangle \xrightarrow[\delta_1]{}^* \langle c_1', \sigma'\rangle \wedge \\
 & & & \quad \langle c_2, \sigma'\rangle \xrightarrow[\delta_2]{}^* \kappa \wedge \delta_1 \not\smile \delta_2 \\
\langle \mathbf{T}[\,\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,]\rangle\!\rangle_{\mathbf{p}} c\,], \sigma\rangle & \Longrightarrow & \textsf{race} & \text{if } \exists \delta_1, \delta_2, c_2', \sigma', \kappa. \\
 & & & \quad \langle c_2, \sigma\rangle \xrightarrow[\delta_2]{}^* \langle c_2', \sigma'\rangle \wedge \\
 & & & \quad \langle c_1, \sigma'\rangle \xrightarrow[\delta_1]{}^* \kappa \wedge \delta_2 \not\smile \delta_1
\end{array}
$$

Figure 5.1: Interleaved mixed-step semantics

the sequential evaluation of a thread diverges.

Definition 5.3. A diverging mixed-step is defined as follows:

$$\kappa \Longrightarrow^{\infty} \qquad \text{if } (\forall n.\ \exists \kappa'.\ \kappa \Longrightarrow^{n} \kappa') \vee (\exists \mathbf{T}, c, \sigma.\ \kappa \Longrightarrow^{*} \langle \mathbf{T}[\,c\,], \sigma\rangle \wedge \langle c, \sigma\rangle \Uparrow)$$

Relation between mixed- and small-step semantics. Now we are able to demonstrate the equivalence properties between the small-step interleaved semantics and the mixed-step semantics.

Lemma 5.4. The following holds:

1. If $\kappa \Longrightarrow^{*} \kappa'$, then $\kappa \longmapsto^{*} \kappa'$
2. If $\kappa \Longrightarrow^{\infty}$, then $\kappa \longmapsto^{\infty}$

Proof. There are 2 propositions:

- By induction over n, the number of steps in $\kappa \Longrightarrow^{*} \kappa'$, we have 2 cases: If $n = 0$, then we know (a) $\kappa \Longrightarrow^{0} \kappa'$. From (a), we know (b) $\kappa = \kappa'$. From (b), we know (c)

$\kappa \longmapsto^0 \kappa'$. From (c), we conclude; If $n > 0$, the we know there exists κ'' such that (a) $\kappa \Longrightarrow \kappa''$ and (b) $\kappa'' \Longrightarrow^{n-1} \kappa'$. We then show that from (a) we have (c) $\kappa \longmapsto^* \kappa''$, where from the semantics we have 8 cases:

– We consider (d) $\kappa = \langle \mathbf{T}[\,c\,], \sigma\rangle$, (e) $\kappa'' = \textsf{abort}$, and (f) $\langle c, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (f), and the semantics, we know (g) $\langle \mathbf{T}[\,c\,], \sigma\rangle \longmapsto^* \textsf{abort}$. From (d), (e), and (g), we have (h) $\kappa \longmapsto^* \kappa''$. From (h), we conclude

– We consider (d) $\kappa = \langle \mathbf{T}[\,c\,], \sigma\rangle$, (e) $\kappa'' = \langle \mathbf{T}[\,c'\,], \sigma'\rangle$, (f) $\langle c, \sigma\rangle \Downarrow \langle c', \sigma'\rangle$, and (g) $c \neq c'$. From (f), we know (h) $\langle c, \sigma\rangle \longrightarrow^* \langle c', \sigma'\rangle$. From (h), and the semantics, we know (i) $\langle \mathbf{T}[\,c\,], \sigma\rangle \longmapsto^* \langle \mathbf{T}[\,c'\,], \sigma'\rangle$. From (d), (e), and (i), we have (j) $\kappa \longmapsto^* \kappa''$. From (j), we conclude

– We consider (d) $\kappa = \langle \mathbf{T}[\,\mathbf{S}[\,c_1 \parallel c_2\,]\,], \sigma\rangle$ and (e) $\kappa'' = \langle \mathbf{T}[\,\langle\!\langle c_1, c_2\rangle\!\rangle_{\mathsf{p}}(\mathbf{S}[\,\mathbf{skip}\,])\,], \sigma\rangle$. From (d), (e), and the semantics, we know (f) $\kappa \longmapsto \kappa''$. From (f), we conclude

– We consider (d) $\kappa = \langle \mathbf{T}[\,\langle\!\langle \mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathsf{p}} c\,], \sigma\rangle$ and (e) $\kappa'' = \langle \mathbf{T}[\,c\,], \sigma\rangle$. From (d), (e), and the semantics, we know (f) $\kappa \longmapsto \kappa''$. From (f), we conclude

– We consider (d) $\kappa = \langle \mathbf{T}[\,\mathbf{S}[\,\mathbf{atomic}\ c\,]\,], \sigma\rangle$, (e) $\kappa'' = \langle \mathbf{T}[\,\langle\!\langle c\rangle\!\rangle_{\mathsf{a}}(\mathbf{S}[\,\mathbf{skip}\,])\,], \sigma\rangle$, and (f) $\mathbf{T}$ is 0-atomic. From (d), (e), (f), and the semantics, we know (g) $\kappa \longmapsto \kappa''$. From (g), we conclude

– We consider (d) $\kappa = \langle \mathbf{T}[\,\langle\!\langle \mathbf{skip}\rangle\!\rangle_{\mathsf{a}} c\,], \sigma\rangle$ and (e) $\kappa'' = \langle \mathbf{T}[\,c\,], \sigma\rangle$. From (d), (e), and the semantics, we know (f) $\kappa \longmapsto \kappa''$. From (f), we conclude

– We consider (d) $\kappa = \langle \mathbf{T}[\,\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,]\rangle\!\rangle_{\mathsf{p}} c\,], \sigma\rangle$, (e) $\kappa'' = \textsf{race}$, (f) $\langle c_1, \sigma\rangle \xrightarrow[\delta_1]{}^* \langle c_1', \sigma'\rangle$, (g) $\langle c_2, \sigma'\rangle \xrightarrow[\delta_2]{}^* \kappa'''$, and (h) $\delta_1 \not\smile \delta_2$. From (f), (g), and (h), using Remark 5.8, we have 2 cases:

  * We know there exists δ_1', c_1'', σ'', δ_2', c_2'', σ''', δ_1'', c_1''', σ'''', δ_2'', and κ'''', such that (i) $\langle c_1, \sigma\rangle \xrightarrow[\delta_1']{}^* \langle c_1'', \sigma''\rangle$, (j) $\langle c_2, \sigma''\rangle \xrightarrow[\delta_2']{}^* \langle c_2'', \sigma'''\rangle$, (k) $\delta_1' \smile \delta_2'$, (l) $\langle c_1'', \sigma'''\rangle \xrightarrow[\delta_1'']{} \langle c_1''', \sigma''''\rangle$, (m) $\langle c_2'', \sigma''''\rangle \xrightarrow[\delta_2'']{} \kappa''''$, and (n) $\delta_1'' \not\smile \delta_2''$. From (i), we know (o) $\langle \mathbf{T}[\,\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,]\rangle\!\rangle_{\mathsf{p}} c\,], \sigma\rangle \longmapsto^* \langle \mathbf{T}[\,\langle\!\langle \mathbf{T}_1[\,c_1''\,], \mathbf{T}_2[\,c_2\,]\rangle\!\rangle_{\mathsf{p}} c\,], \sigma''\rangle$.

From (j), and the semantics, we know (p) $\langle \mathbf{T}[\langle\!\langle \mathbf{T}_1[\, c_1'' \,], \mathbf{T}_2[\, c_2 \,] \rangle\!\rangle_{\mathbf{p}} c \,], \sigma'' \rangle \longmapsto^* \langle \mathbf{T}[\langle\!\langle \mathbf{T}_1[\, c_1'' \,], \mathbf{T}_2[\, c_2'' \,] \rangle\!\rangle_{\mathbf{p}} c \,], \sigma''' \rangle$. From (l), (m), (n), and the semantics, we know (q) $\langle \mathbf{T}[\langle\!\langle \mathbf{T}_1[\, c_1'' \,], \mathbf{T}_2[\, c_2'' \,] \rangle\!\rangle_{\mathbf{p}} c \,], \sigma''' \rangle \longmapsto \textsf{race}$. From (d), (e), (o), (p), and (q), we have (r) $\kappa \longmapsto^* \kappa''$. From (r), we conclude

  * We know there exists δ_1', c_1'', σ'', δ_2', c_2'', σ''', δ_2'', c_2''', σ'''', δ_1'', and κ'''', such that (i) $\langle c_1, \sigma \rangle \xrightarrow[\delta_1']{}^* \langle c_1'', \sigma'' \rangle$, (j) $\langle c_2, \sigma'' \rangle \xrightarrow[\delta_2']{}^* \langle c_2'', \sigma''' \rangle$, (k) $\delta_1' \smile \delta_2'$, (l) $\langle c_2'', \sigma''' \rangle \xrightarrow[\delta_2'']{} \langle c_2''', \sigma'''' \rangle$, (m) $\langle c_1'', \sigma'''' \rangle \xrightarrow[\delta_1'']{} \kappa''''$, and (n) $\delta_2'' \not\smile \delta_1''$. Proof is symmetric to the previous case

- We consider (d) $\kappa = \langle \mathbf{T}[\langle\!\langle \mathbf{T}_1[\, c_1 \,], \mathbf{T}_2[\, c_2 \,] \rangle\!\rangle_{\mathbf{p}} c \,], \sigma \rangle$, (e) $\kappa'' = \textsf{race}$, (f) $\langle c_2, \sigma \rangle \xrightarrow[\delta_2]{}^* \langle c_2', \sigma' \rangle$, (g) $\langle c_1, \sigma' \rangle \xrightarrow[\delta_1]{}^* \kappa'''$, and (h) $\delta_2 \not\smile \delta_1$. Proof is symmetric to the previous case.

From (b), and the induction hypothesis, we have (d) $\kappa'' \longmapsto^* \kappa'$. From (c), (d), and the semantics, we have (e) $\kappa \longmapsto^* \kappa'$. From (e), we conclude

- By induction over n, which by Def. 3.19 is the number of steps in $\kappa \longmapsto^\infty$, we need to show that there exists κ' such that $\kappa \longmapsto^n \kappa'$ We have 2 cases: If $n = 0$, we instantiate the goal with κ and conclude; If $n > 0$, given (a), by Def. 5.3, we have two cases:

  - We consider (b) $\forall n.\ \exists \kappa'.\ \kappa \Longrightarrow^n \kappa'$. From (b), we know there exists κ' such that (c) $\kappa \Longrightarrow^n \kappa'$. From (c), and the semantics, we have (d) $\kappa \longmapsto^{n'} \kappa'$ and (e) $n \leq n'$. From (d) and (e), we know there exists κ'' such that (f) $\kappa \longmapsto^n \kappa''$. Instantiating the goal with κ'', from (f), we conclude.

  - We consider (b) $\kappa \Longrightarrow^* \langle \mathbf{T}[\, c \,], \sigma \rangle$ and (c) $\langle c, \sigma \rangle \Uparrow$. From (b), we know (d) $\kappa \Longrightarrow^{n'} \langle \mathbf{T}[\, c \,], \sigma \rangle$. From (d), and the semantics, we know (e) $\kappa \longmapsto^{n''} \langle \mathbf{T}[\, c \,], \sigma \rangle$ and (f) $n' \leq n''$. We then consider 2 cases:

    * If (g) $n \leq n''$, from (e), we know there exists κ'' such that (f) $\kappa \longmapsto^n \kappa''$. Instantiating the goal with κ'', from (f), we conclude.

    * If (g) $n > n''$, from (c), we know exists c' and σ' such that (h) $\langle c, \sigma \rangle \longrightarrow^{n-n''}$

$\langle c', \sigma' \rangle$. From (h), we know (i) $\langle \mathbf{T}[\,c\,], \sigma \rangle \longmapsto^{n-n''} \langle \mathbf{T}[\,c'\,], \sigma' \rangle$. From (e), (i), and the semantics, we know (j) $\kappa \longmapsto^{n} \langle \mathbf{T}[\,c'\,], \sigma' \rangle$. Instantiating the goal with $\langle \mathbf{T}[\,c'\,], \sigma' \rangle$, from (j), we conclude $\square$

Lemma 5.5. The following holds:

1. If $\kappa \longmapsto^* \textsf{race}$, then $\kappa \Longrightarrow^* \textsf{race}$
2. If $\neg\kappa \longmapsto^* \textsf{race}$, then
   (a) If $\kappa \longmapsto^* \textsf{abort}$, then $\kappa \Longrightarrow^* \textsf{abort}$
   (b) If $\kappa \longmapsto^* \langle \mathbf{skip}, \sigma \rangle$, then $\kappa \Longrightarrow^* \langle \mathbf{skip}, \sigma \rangle$
   (c) If $\kappa \longmapsto^\infty$, then $\kappa \Longrightarrow^\infty$

Proof. There are 2 propositions:

- By induction over n, the number of steps in (a) $\kappa \longmapsto^* \textsf{race}$, we have 2 cases: If $n = 0$, from (a) and the semantics, we know (b) $\kappa = \textsf{race}$. From the semantics, we know (c) $\textsf{race} \Longrightarrow^0 \textsf{race}$. From (b) and (c), we have (d) $\kappa \Longrightarrow^* \textsf{race}$. From (d), we conclude; If $n > 0$, from (a) and the semantics, we know there exists κ' such that (b) $\kappa \longmapsto \kappa'$ and (c) $\kappa' \longmapsto^{n-1} \textsf{race}$. From (c), and the induction hypothesis, we know (d) $\kappa' \Longrightarrow^* \textsf{race}$. By a second induction, over n', the number of steps in (d), we have 2 cases: If $n' = 0$, from (d) and the semantics, we know (e) $\kappa' = \textsf{race}$. From (b) and (e), we obtain (f) $\kappa \Longrightarrow \textsf{race}$. From (f), we conclude; If $n' > 0$, from (d) and the semantics, we know there exists κ'' such that (e) $\kappa' \Longrightarrow \kappa''$ and (f) $\kappa'' \Longrightarrow^{n'-1} \textsf{race}$. From (b) and (e), using Lemma 5.9, we have 3 cases:
  - We consider (g) $\kappa \Longrightarrow \textsf{race}$. From (g), we conclude
  - We consider (g) $\kappa \Longrightarrow^* \kappa''$. From (g) and (f), we know (h) $\kappa \Longrightarrow^* \textsf{race}$. From (h), we conclude

    - We consider (g) $\kappa \Longrightarrow \kappa'''$ and (h) $\kappa''' \longmapsto \kappa''$. From (h) and (f), using the second induction hypothesis, we know (i) $\kappa''' \Longrightarrow^* \textsf{race}$. From (g) and (i), we know (j) $\kappa \Longrightarrow^* \textsf{race}$. From (j), we conclude

- We assume (a) $\neg\kappa \longmapsto^* \textsf{race}$, then we have 3 cases:

    - By induction over n, the number of steps in (b) $\kappa \longmapsto^* \textsf{abort}$, we have 2 cases: If $n = 0$, from (b) and the semantics, we have (c) $\kappa = \textsf{abort}$ From the semantics, we know (d) $\textsf{abort} \Longrightarrow^0 \textsf{abort}$. From (c) and (d), we have (e) $\kappa \Longrightarrow^* \textsf{abort}$. From (e), we conclude; If $n > 0$, from (b) and the semantics, we know there exists κ' such that (c) $\kappa \longmapsto \kappa'$ and (d) $\kappa' \longmapsto^{n-1} \textsf{abort}$. From (a), (c), and the semantics, we know (e) $\neg\kappa' \longmapsto^* \textsf{race}$. From (e) and (d), using the induction hypothesis, we know (f) $\kappa' \Longrightarrow^* \textsf{abort}$. By a second induction, over n', the number of steps in (f), we have 2 cases: If $n' = 0$, from (f) and the semantics, we know (g) $\kappa' = \textsf{abort}$. From (c) and (g), we obtain (h) $\kappa \Longrightarrow \textsf{abort}$. From (h), we conclude; If $n' > 0$, from (f) and the semantics, we know there exists κ'' such that (g) $\kappa' \Longrightarrow \kappa''$ and (h) $\kappa'' \Longrightarrow^{n'-1} \textsf{abort}$. From (c) and (g), using Lemma 5.9, we have 3 cases:

        * We consider (i) $\kappa \Longrightarrow \textsf{race}$. From (i), using Lemma 5.4 (item 1), we have (j) $\kappa \longmapsto^* \textsf{race}$. Given that (j) contradicts (a), we conclude

        * We consider (i) $\kappa \Longrightarrow^* \kappa''$. From (i) and (h), we know (h) $\kappa \Longrightarrow^* \textsf{abort}$. From (h), we conclude

        * We consider (i) $\kappa \Longrightarrow \kappa'''$ and (j) $\kappa''' \longmapsto \kappa''$. From (i), using Lemma 5.4 (item 1), we know (k) $\kappa \longmapsto^* \kappa'''$. From (a), (k), and the semantics, we know (l) $\neg\kappa''' \longmapsto^* \textsf{race}$. From (l), (j), and (h), using the second induction hypothesis, we know (m) $\kappa''' \Longrightarrow^* \textsf{abort}$. From (i) and (m), we know (n) $\kappa \Longrightarrow^* \textsf{abort}$. From (n), we conclude

    - By induction over n, the number of steps in (b) $\kappa \longmapsto^* \langle \textbf{skip}, \sigma\rangle$, we have 2 cases: If $n = 0$, from (b) and the semantics, we have (c) $\kappa = \langle \textbf{skip}, \sigma\rangle$ From the semantics, we know (d) $\langle \mathbf{skip}, \sigma \rangle \Longrightarrow^0 \langle \mathbf{skip}, \sigma \rangle$. From (c) and (d), we have (e) $\kappa \Longrightarrow^* \langle \mathbf{skip}, \sigma \rangle$. From (e), we conclude; If $n > 0$, from (b) and the semantics, we know there exists κ' such that (c) $\kappa \longmapsto \kappa'$ and (d) $\kappa' \longmapsto^{n-1} \langle \mathbf{skip}, \sigma \rangle$. From (a), (c), and the semantics, we know (e) $\neg\kappa' \longmapsto^* \mathsf{race}$. From (e) and (d), using the induction hypothesis, we know (f) $\kappa' \Longrightarrow^* \langle \mathbf{skip}, \sigma \rangle$. By a second induction, over n', the number of steps in (f), we have 2 cases: If $n' = 0$, from (f) and the semantics, we know (g) $\kappa' = \langle \mathbf{skip}, \sigma \rangle$. From (c) and (g), we obtain (h) $\kappa \Longrightarrow \langle \mathbf{skip}, \sigma \rangle$. From (h), we conclude; If $n' > 0$, from (f) and the semantics, we know there exists κ'' such that (g) $\kappa' \Longrightarrow \kappa''$ and (h) $\kappa'' \Longrightarrow^{n'-1} \langle \mathbf{skip}, \sigma \rangle$. From (c) and (g), using Lemma 5.9, we have 3 cases:

  * We consider (i) $\kappa \Longrightarrow \mathsf{race}$. From (i), using Lemma 5.4 (item 1), we have (j) $\kappa \longmapsto^* \mathsf{race}$. Given that (j) contradicts (a), we conclude
  * We consider (i) $\kappa \Longrightarrow^* \kappa''$. From (i) and (h), we know (h) $\kappa \Longrightarrow^* \langle \mathbf{skip}, \sigma \rangle$. From (h), we conclude
  * We consider (i) $\kappa \Longrightarrow \kappa'''$ and (j) $\kappa''' \longmapsto \kappa''$. From (i), using Lemma 5.4 (item 1), we know (k) $\kappa \longmapsto^* \kappa'''$. From (a), (k), and the semantics, we know (l) $\neg\kappa''' \longmapsto^* \mathsf{race}$. From (l), (j), and (h), using the second induction hypothesis, we know (m) $\kappa''' \Longrightarrow^* \langle \mathbf{skip}, \sigma \rangle$. From (i) and (m), we know (n) $\kappa \Longrightarrow^* \langle \mathbf{skip}, \sigma \rangle$. From (n), we conclude

– Given (b) $\kappa \longmapsto^\infty$, we consider 2 cases:

  * We assume (c) $\kappa \longmapsto^* \langle \mathbf{T}[\,c\,], \sigma \rangle$ and (d) $\langle c, \sigma \rangle \longrightarrow^\infty$. and we will establish that exists $\mathbf{T}'$, c', and σ', such that $\kappa \Longrightarrow^* \langle \mathbf{T}'[\,c'\,], \sigma' \rangle$ and $\langle c', \sigma' \rangle \longrightarrow^\infty$. By induction over n, the number of steps in (c), we have 2 cases: If $n = 0$, from (c) and the semantics, we have (e) $\kappa = \langle \mathbf{T}[\,c\,], \sigma \rangle$ From the semantics, we know (f) $\langle \mathbf{T}[\,c\,], \sigma \rangle \Longrightarrow^0 \langle \mathbf{T}[\,c\,], \sigma \rangle$. From (e) and (f), we have (g) $\kappa \Longrightarrow^* \langle \mathbf{T}[\,c\,], \sigma \rangle$. Instantiating the goal with $\mathbf{T}$, c, and σ, from (g) and (d), we conclude; If $n > 0$, from (c) and the semantics, we know there exists

κ' such that (e) $\kappa \longmapsto \kappa'$ and (f) $\kappa' \longmapsto^{n-1} \langle \mathbf{T}[\,c\,], \sigma\rangle$. From (a), (e), and the semantics, we know (g) $\neg\kappa' \longmapsto^{*}$ race. From (g), (f), and (d), using the induction hypothesis, we know (h) $\kappa' \Longrightarrow^{*} \langle \mathbf{T}''[\,c''\,], \sigma''\rangle$ and (j) $\langle c'', \sigma''\rangle \longrightarrow^{\infty}$. By a second induction, over n', the number of steps in (h), we have 2 cases: If $n' = 0$, from (h) and the semantics, we know (k) $\kappa' = \langle \mathbf{T}''[\,c''\,], \sigma''\rangle$. From (e) and (k), we have 3 possibilities:

- We consider (l) $\kappa = \langle \mathbf{T}''[\,c'''\,], \sigma'''\rangle$ and (m) $\langle c''', \sigma'''\rangle \longrightarrow \langle c'', \sigma''\rangle$. From (m) and (j), we have (n) $\langle c''', \sigma'''\rangle \longrightarrow^{\infty}$. From the semantics, we have (o) $\kappa \Longrightarrow^{0} \kappa$. Instantiating the goal with $\mathbf{T}''$, c''', and σ''', from (o), (l), and (n), we conclude
- We consider (l) $\kappa = \langle \mathbf{T}'''[\,c'''\,], \sigma'''\rangle$, (m) $\kappa' = \langle \mathbf{T}'''[\,c''''\,], \sigma''\rangle$, (n) $\langle c''', \sigma'''\rangle \underset{\delta}{\longrightarrow} \langle c'''', \sigma''\rangle$, and (o) $\mathbf{T}''' \neq \mathbf{T}''$. From (a), (o), (j), we know (p) $\langle c'', \sigma'''\rangle \longrightarrow^{\infty}$ and there exists $\mathbf{T}''''$ such that (q) $\mathbf{T}'''[\,c'''\,] = \mathbf{T}''''[\,c''\,]$. From the semantics, we have (r) $\kappa \Longrightarrow^{0} \kappa$. Instantiating the goal with $\mathbf{T}''''$, c'', and σ''', from (r), (l), (q), and (p), we conclude
- We consider (l) $\kappa \Longrightarrow \kappa'$. Instantiating the goal with $\mathbf{T}''$, c'', and σ'', from (l), (k), and (j), we conclude

If $n' > 0$, from (h) and the semantics, we know there exists κ'' such that (k) $\kappa' \Longrightarrow \kappa''$ and (l) $\kappa'' \Longrightarrow^{n'-1} \langle \mathbf{T}''[\,c''\,], \sigma''\rangle$. From (e) and (k), using Lemma 5.9, we have 3 cases:

- We consider (m) $\kappa \Longrightarrow$ race. From (m), using Lemma 5.4 (item 1), we have (n) $\kappa \longmapsto^{*}$ race. Given that (n) contradicts (a), we conclude
- We consider (m) $\kappa \Longrightarrow^{*} \kappa''$. From (m) and (l), we know (n) $\kappa \Longrightarrow^{*} \langle \mathbf{T}''[\,c''\,], \sigma''\rangle$. Instantiating the goal with $\mathbf{T}''$, c'', and σ'', from (n) and (j), we conclude
- We consider (m) $\kappa \Longrightarrow \kappa'''$ and (n) $\kappa''' \longmapsto \kappa''$. From (m), using Lemma 5.4 (item 1), we know (o) $\kappa \longmapsto^{*} \kappa'''$. From (a), (o), and the semantics,

we know (p) $\neg\kappa''' \longmapsto^{*}$ race. From (p), (n), (l), and (j), using the second induction hypothesis, we know (q) $\kappa''' \Longrightarrow^{*} \langle \mathbf{T}'''[\,c'''\,], \sigma'''\rangle$ and (r) $\langle c''', \sigma'''\rangle \longrightarrow^{\infty}$. From (m) and (q), we know (s) $\kappa \Longrightarrow^{*} \langle \mathbf{T}'''[\,c'''\,], \sigma'''\rangle$. Instantiating the goal with $\mathbf{T}'''$, c''', and σ''', from (s), we conclude

$*$ We assume (c) for all $\mathbf{T}$, c, and σ, such that $\kappa \longmapsto^{*} \langle \mathbf{T}[\,c\,], \sigma\rangle$, we know $\neg\langle c, \sigma\rangle \longrightarrow^{\infty}$. From (c), we know there is a n_{max} such that (d) for all $\mathbf{T}$, c, and σ, such that $\kappa \longmapsto^{*} \langle \mathbf{T}[\,c\,], \sigma\rangle$, we know there exists n and κ' such that $\langle c, \sigma\rangle \longrightarrow^{n} \kappa'$, $\neg\exists\kappa''.\ \kappa' \longrightarrow \kappa''$, and $n \leq n_{max}$. We will establish that $\forall n.\ \exists\kappa'.\ \kappa \Longrightarrow^{n} \kappa'$. By induction over n, we have 2 cases: If (f) $n = 0$, instantiating the goal with κ, we conclude; If (f) $n > 0$, from (a), (b), and (d), we know there exists κ'' such that (g) $\kappa \Longrightarrow \kappa''$, (h) $\kappa'' \longmapsto^{\infty}$, and (i) for all $\mathbf{T}$, c, and σ, such that $\kappa'' \longmapsto^{*} \langle \mathbf{T}[\,c\,], \sigma\rangle$, we know there exists n and κ' such that $\langle c, \sigma\rangle \longrightarrow^{n} \kappa'$, $\neg\exists\kappa''.\ \kappa' \longrightarrow \kappa''$, and $n \leq n_{max}$. From (g), using Lemma 5.4 (item 1), we know (k) $\kappa \longmapsto^{*} \kappa''$. From (a), (k), and the semantics, we know (l) $\neg\kappa'' \longmapsto^{*}$ race. From (l), (h), and (i), using the induction hypothesis, we know that exists κ''' such that (m) $\kappa'' \Longrightarrow^{n-1} \kappa'''$. From (g), (m), and the semantics, we know (n) $\kappa \Longrightarrow^{n} \kappa'''$. Instantiating the goal with κ''', from (n), we conclude $\square$

To prove these two lemmas, we observe the following remarks which define the conditions for swapping two consequent steps from non-interfering commands, and finding the earliest race between two consequent multi-steps from interfering commands.

Remark 5.6. If $\delta_1 \overset{\rightharpoonup}{\smile} \delta_2$, $\langle c_1, \sigma\rangle \xrightarrow[\delta_1]{} \langle c_1', \sigma'\rangle$, and $\langle c_2, \sigma'\rangle \xrightarrow[\delta_2]{}$ abort, then $\langle c_2, \sigma\rangle \xrightarrow[\delta_2]{}$ abort

Remark 5.7. If $\delta_1 \smile \delta_2$, and $\langle c_1, \sigma\rangle \xrightarrow[\delta_1]{} \langle c_1', \sigma'\rangle$, and $\langle c_2, \sigma'\rangle \xrightarrow[\delta_2]{} \langle c_2', \sigma''\rangle$, then exists σ''' such that $\langle c_2, \sigma\rangle \xrightarrow[\delta_2]{} \langle c_2', \sigma'''\rangle$ and $\langle c_1, \sigma'''\rangle \xrightarrow[\delta_1]{} \langle c_1', \sigma''\rangle$

Remark 5.8. If $\langle c_1, \sigma\rangle \xrightarrow[\delta_1]{}^{*} \langle c_1', \sigma'\rangle$, $\langle c_2, \sigma'\rangle \xrightarrow[\delta_2]{}^{*} \kappa$, and $\delta_1 \not\smile \delta_2$, then exists δ_1', c_1'', σ'', δ_2', c_2'', σ''', δ_1'', δ_2'', c_1''', c_2''', σ'''', κ' such that $\langle c_1, \sigma\rangle \xrightarrow[\delta_1']{}^{*} \langle c_1'', \sigma''\rangle$, $\langle c_2, \sigma''\rangle \xrightarrow[\delta_2']{}^{*} \langle c_2'', \sigma'''\rangle$, $\delta_1' \smile \delta_2'$, and either: $\langle c_1'', \sigma''' \rangle \xrightarrow[\delta_1'']{} \langle c_1''', \sigma'''' \rangle$, $\langle c_2'', \sigma'''' \rangle \xrightarrow[\delta_2'']{}^* \kappa'$, and $\delta_1'' \not\smile \delta_2''$; or $\langle c_2'', \sigma''' \rangle \xrightarrow[\delta_2'']{} \langle c_2''', \sigma'''' \rangle$, $\langle c_1'', \sigma'''' \rangle \xrightarrow[\delta_1'']{}^* \kappa'$, and $\delta_2'' \not\smile \delta_1''$

We also rely on the following composition/swap lemma:

Lemma 5.9. If $\kappa \longmapsto \kappa'$, and $\kappa' \Longrightarrow \kappa''$, then either:

1. $\kappa \Longrightarrow \mathsf{race}$
2. or $\kappa \Longrightarrow^* \kappa''$
3. or exists κ''' such that $\kappa \Longrightarrow \kappa'''$ and $\kappa''' \longmapsto \kappa''$

Proof. We assume (a) $\kappa \longmapsto \kappa'$ and (b) $\kappa' \Longrightarrow \kappa''$. From (a), and the semantics, we have 8 cases:

- We consider (c) $\kappa = \langle \mathbf{T}[\,c\,], \sigma \rangle$, (d) $\kappa' = \mathsf{abort}$, and (e) $\langle c, \sigma \rangle \longrightarrow \mathsf{abort}$. From (d), and the semantics, we know (b) is false and conclude
- We consider (c) $\kappa = \langle \mathbf{T}[\,c\,], \sigma \rangle$, (d) $\kappa' = \langle \mathbf{T}[\,c'\,], \sigma' \rangle$, and (e) $\langle c, \sigma \rangle \xrightarrow[\delta]{} \langle c', \sigma' \rangle$. From (b), and the semantics, we have 8 cases:
  - We consider (f) $\kappa' = \langle \mathbf{T}'[\,c''\,], \sigma' \rangle$, (g) $\kappa'' = \mathsf{abort}$, and (h) $\langle c'', \sigma' \rangle \xrightarrow[\delta']{}^* \mathsf{abort}$. We then consider 2 cases:
    * If (i) $\mathbf{T} = \mathbf{T}'$, from (d) and (f), we have (j) $c' = c''$. From (e), (f), (j), (h), and the semantics, we have (k) $\langle c, \sigma \rangle \longrightarrow^* \mathsf{abort}$. From (k), and the semantics, we have (l) $\kappa \Longrightarrow \mathsf{abort}$. From (l), (g), and the semantics, we have (m) $\kappa \Longrightarrow^* \kappa''$. From (m), we conclude
    * If (i) $\mathbf{T} \neq \mathbf{T}'$, we need to consider 2 cases:
      · We consider (j) $\delta \overset{\frown}{\smile} \delta'$. From (j), (e), and (h), using Remark 5.6, we know (k) $\langle c'', \sigma \rangle \xrightarrow[\delta']{}^* \mathsf{abort}$. From (k), using the semantics, we know (l) $\kappa \Longrightarrow \mathsf{abort}$. From (l), (g), and the semantics, we have (m) $\kappa \Longrightarrow^* \kappa''$. From (m), we conclude

    · We consider (j) $\delta \not\overset{\rightarrow}{\smile} \delta'$. From (e), we know (k) $\langle c, \sigma \rangle \xrightarrow[\delta]{}^{*} \langle c', \sigma' \rangle$. From (k), (h), (j), and the semantics, we know (l) $\kappa \Longrightarrow \textsf{race}$. From (l), we conclude

- We consider (f) $\kappa' = \langle \mathbf{T}'[\,c''\,], \sigma' \rangle$, (g) $\kappa'' = \langle \mathbf{T}'[\,c'''\,], \sigma'' \rangle$, (h) $\langle c'', \sigma' \rangle \Downarrow \langle c''', \sigma'' \rangle$, and (i) $c'' \neq c'''$. We then consider 2 cases:

  * If (j) $\mathbf{T} = \mathbf{T}'$, from (d) and (f), we have (k) $c' = c''$. From (e), (f), (k), (h), and the semantics, we have (l) $\langle c, \sigma \rangle \Downarrow \langle c''', \sigma'' \rangle$ and (m) $c \neq c'''$. From (c), (g), (l), (m), and the semantics, we have (n) $\kappa \Longrightarrow \kappa''$. From (n), and the semantics, we have (o) $\kappa \Longrightarrow^{*} \kappa''$. From (o), we conclude

  * If (j) $\mathbf{T} \neq \mathbf{T}'$, we know from (h) that there exists δ' such that (k) $\langle c'', \sigma' \rangle \xrightarrow[\delta']{}^{*} \langle c''', \sigma'' \rangle$, and we need to consider 2 cases:

    · We consider (l) $\delta \overset{\rightarrow}{\smile} \delta'$, then we have 2 more cases: we consider (m) $\delta' \overset{\rightarrow}{\smile} \delta$. From (l), (m), (e), (k), using Remark 5.7, we know there exists σ''' such that (n) $\langle c''', \sigma''' \rangle \xrightarrow[\delta']{}^{\langle c'', \sigma \rangle}$ and (o) $\langle c, \sigma''' \rangle \longrightarrow \langle c', \sigma'' \rangle$. From (n), (h), and the semantics, we know (p) $\langle c'', \sigma \rangle \Downarrow \langle c''', \sigma''' \rangle$. From (c), (d), (f), (g), and (j), we know there exists $\mathbf{T}''$ and $\mathbf{T}'''$ such that (q) $\mathbf{T}[\,c\,] = \mathbf{T}''[\,c''\,]$ and (r) $\mathbf{T}''[\,c'''\,] = \mathbf{T}'''[\,c\,]$ and (s) $\mathbf{T}'''[\,c'\,] = \mathbf{T}'[\,c'''\,]$. From (p), (i), and the semantics, we have (t) $\langle \mathbf{T}''[\,c''\,], \sigma \rangle \Longrightarrow \langle \mathbf{T}''[\,c'''\,], \sigma''' \rangle$. From (o), and the semantics, we have (u) $\langle \mathbf{T}'''[\,c\,], \sigma''' \rangle \longmapsto \langle \mathbf{T}'''[\,c'\,], \sigma'' \rangle$ Instantiating the goal with $\langle \mathbf{T}'''[\,c\,], \sigma''' \rangle$, from (t), and (u), we conclude; we consider (m) $\delta' \not\overset{\rightarrow}{\smile} \delta$. From (m), (e), (k), and the semantics, it is not hard to see that there exists σ''', δ'', and κ''' such that (n) $\langle c'', \sigma \rangle \xrightarrow[\delta']{}^{*} \langle c''', \sigma''' \rangle$, (o) $\langle c, \sigma''' \rangle \xrightarrow[\delta'']{} \kappa'''$, and (p) $\delta' \not\overset{\rightarrow}{\smile} \delta''$. From (n), (o), (p), and the semantics, we know (q) $\kappa \Longrightarrow \textsf{race}$. From (q), we conclude

    · We consider (l) $\delta \not\overset{\rightarrow}{\smile} \delta'$. From (e), we know (m) $\langle c, \sigma \rangle \xrightarrow[\delta]{}^{*} \langle c', \sigma' \rangle$. From (m), (k), (l), and the semantics, we know (n) $\kappa \Longrightarrow \textsf{race}$. From (n), we conclude

– We consider (f) $\kappa' = \langle \mathbf{T}'[\,\mathbf{S}[\,c_1 \,\|\, c_2\,]\,], \sigma' \rangle$ and (g) $\kappa'' = \langle \mathbf{T}'[\,\langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,])\,], \sigma' \rangle$. We then consider 2 cases:

  ∗ If (h) $\mathbf{T} = \mathbf{T}'$, we know that (i) $c' = \mathbf{S}[\,c_1 \,\|\, c_2\,]$. From (e), (i), and the semantics, we know that (j) $\langle c, \sigma \rangle \Downarrow \langle c', \sigma' \rangle$ and (k) $c \neq c'$. From (c), (d), (j), (k), and the semantics, we know (l) $\kappa \Longrightarrow \kappa'$. From (l), (b), and the semantics, we know (m) $\kappa \Longrightarrow^* \kappa''$. From (m), we conclude

  ∗ If (h) $\mathbf{T} \neq \mathbf{T}'$, we know there exists $\mathbf{T}''$ and $\mathbf{T}'''$ such that (i) $\mathbf{T}[\,c\,] = \mathbf{T}''[\,\mathbf{S}[\,c_1 \,\|\, c_2\,]\,]$, (j) $\mathbf{T}''[\,\langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,])\,] = \mathbf{T}'''[\,c\,]$, and (k) $\mathbf{T}'''[\,c'\,] = \mathbf{T}'[\,\langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,])\,]$. From the semantics, we know
  (l) $\langle \mathbf{T}''[\,\mathbf{S}[\,c_1 \,\|\, c_2\,]\,], \sigma \rangle \Longrightarrow \langle \mathbf{T}''[\,\langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,])\,], \sigma \rangle$. From (e), and the semantics, we know (m) $\langle \mathbf{T}'''[\,c\,], \sigma \rangle \longmapsto \langle \mathbf{T}'''[\,c'\,], \sigma' \rangle$. Instantiating the goal with $\langle \mathbf{T}'''[\,c\,], \sigma \rangle$, from (c), (g), (i), (j), (k), (l), and (m), we conclude

– We consider (f) $\kappa' = \langle \mathbf{T}'[\,\langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathbf{p}} c''\,], \sigma' \rangle$ and (g) $\kappa'' = \langle \mathbf{T}'[\,c''\,], \sigma' \rangle$. We then consider 3 cases:

  ∗ If (h) $\mathbf{T} = \mathbf{T}'[\,\langle\!\langle \bullet, \mathbf{skip} \rangle\!\rangle_{\mathbf{p}} c''\,]$, we know that (i) $c' = \mathbf{skip}$. From (e), (i), and the semantics, we know that (j) $\langle c, \sigma \rangle \Downarrow \langle c', \sigma' \rangle$ and (k) $c \neq c'$. From (c), (d), (j), (k), and the semantics, we know (l) $\kappa \Longrightarrow \kappa'$. From (l), (b), and the semantics, we know (m) $\kappa \Longrightarrow^* \kappa''$. From (m), we conclude

  ∗ If (h) $\mathbf{T} = \mathbf{T}'[\,\langle\!\langle \mathbf{skip}, \bullet \rangle\!\rangle_{\mathbf{p}} c''\,]$, we know that (i) $c' = \mathbf{skip}$. From (e), (i), and the semantics, we know that (j) $\langle c, \sigma \rangle \Downarrow \langle c', \sigma' \rangle$ and (k) $c \neq c'$. From (c), (d), (j), (k), and the semantics, we know (l) $\kappa \Longrightarrow \kappa'$. From (l), (b), and the semantics, we know (m) $\kappa \Longrightarrow^* \kappa''$. From (m), we conclude

  ∗ If (h) $\mathbf{T} \neq \mathbf{T}'[\,\langle\!\langle \bullet, \mathbf{skip} \rangle\!\rangle_{\mathbf{p}} c''\,]$ and (i) $\mathbf{T} \neq \mathbf{T}'[\,\langle\!\langle \mathbf{skip}, \bullet \rangle\!\rangle_{\mathbf{p}} c''\,]$, we know there exists $\mathbf{T}''$ and $\mathbf{T}'''$ such that (j) $\mathbf{T}[\,c\,] = \mathbf{T}''[\,\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathbf{a}} c''\,]$, (k) $\mathbf{T}''[\,c''\,] = \mathbf{T}'''[\,c\,]$, and (l) $\mathbf{T}'''[\,c'\,] = \mathbf{T}'[\,c''\,]$. From the semantics, we know
  (m) $\langle \mathbf{T}''[\,\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathbf{a}} c''\,], \sigma \rangle \Longrightarrow \langle \mathbf{T}''[\,c''\,], \sigma \rangle$. From (e), and the semantics, we know (n) $\langle \mathbf{T}'''[\,c\,], \sigma \rangle \longmapsto \langle \mathbf{T}'''[\,c'\,], \sigma' \rangle$. Instantiating the goal with

$\langle \mathbf{T}'''[\,c\,], \sigma\rangle$, from (c), (g), (j), (k), (l), (m), and (n), we conclude

– We consider (f) $\kappa' = \langle \mathbf{T}'[\,\mathbf{S}[\,\textbf{atomic}\ c''\,]\,], \sigma'\rangle$, (g) $\kappa'' = \langle \mathbf{T}'[\,\langle\!\langle c''\rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\textbf{skip}\,])\,], \sigma'\rangle$, and (h) $\mathbf{T}'$ is 0-atomic. We then consider 2 cases:

  * If (i) $\mathbf{T} = \mathbf{T}'$, we know that (j) $c' = \mathbf{S}[\,\textbf{atomic}\ c''\,]$. From (e), (j), and the semantics, we know that (k) $\langle c, \sigma\rangle \Downarrow \langle c', \sigma'\rangle$ and (l) $c \neq c'$. From (c), (d), (k), (l), and the semantics, we know (m) $\kappa \Longrightarrow \kappa'$. From (m), (b), and the semantics, we know (n) $\kappa \Longrightarrow^* \kappa''$. From (n), we conclude

  * If (i) $\mathbf{T} \neq \mathbf{T}'$, we know there exists $\mathbf{T}''$ and $\mathbf{T}'''$ such that (j) $\mathbf{T}[\,c\,] = \mathbf{T}''[\,\mathbf{S}[\,\textbf{atomic}\ c''\,]\,]$, (k) $\mathbf{T}''[\,\langle\!\langle c''\rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\textbf{skip}\,])\,] = \mathbf{T}'''[\,c\,]$, and (l) $\mathbf{T}'''[\,c'\,] = \mathbf{T}'[\,\langle\!\langle c''\rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\textbf{skip}\,])\,]$. From (h), and the semantics, we know
  (m) $\langle \mathbf{T}''[\,\mathbf{S}[\,\textbf{atomic}\ c''\,]\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}''[\,\langle\!\langle c''\rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\textbf{skip}\,])\,], \sigma\rangle$. From (e), and the semantics, we know (n) $\langle \mathbf{T}'''[\,c\,], \sigma\rangle \longmapsto \langle \mathbf{T}'''[\,c'\,], \sigma'\rangle$. Instantiating the goal with $\langle \mathbf{T}'''[\,c\,], \sigma\rangle$, from (c), (g), (j), (k), (l), (m), and (n), we conclude

– We consider (f) $\kappa' = \langle \mathbf{T}'[\,\langle\!\langle \textbf{skip}\rangle\!\rangle_{\mathbf{a}} c''\,], \sigma'\rangle$ and (g) $\kappa'' = \langle \mathbf{T}'[\,c''\,], \sigma'\rangle$. We then consider 2 cases:

  * If (h) $\mathbf{T} = \mathbf{T}'[\,\langle\!\langle \bullet\rangle\!\rangle_{\mathbf{a}} c''\,]$, we know that (i) $c' = \textbf{skip}$. From (e), (i), and the semantics, we know that (j) $\langle c, \sigma\rangle \Downarrow \langle c', \sigma'\rangle$ and (k) $c \neq c'$. From (c), (d), (j), (k), and the semantics, we know (l) $\kappa \Longrightarrow \kappa'$. From (l), (b), and the semantics, we know (m) $\kappa \Longrightarrow^* \kappa''$. From (m), we conclude

  * If (h) $\mathbf{T} \neq \mathbf{T}'[\,\langle\!\langle \bullet\rangle\!\rangle_{\mathbf{a}} c''\,]$, we know there exists $\mathbf{T}''$ and $\mathbf{T}'''$ such that (i) $\mathbf{T}[\,c\,] = \mathbf{T}''[\,\langle\!\langle \textbf{skip}\rangle\!\rangle_{\mathbf{a}} c''\,]$, (j) $\mathbf{T}''[\,c''\,] = \mathbf{T}'''[\,c\,]$, and (k) $\mathbf{T}'''[\,c'\,] = \mathbf{T}'[\,c''\,]$. From the semantics, we know (l) $\langle \mathbf{T}''[\,\langle\!\langle \textbf{skip}\rangle\!\rangle_{\mathbf{a}} c''\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}''[\,c''\,], \sigma\rangle$. From (e), and the semantics, we know (m) $\langle \mathbf{T}'''[\,c\,], \sigma\rangle \longmapsto \langle \mathbf{T}'''[\,c'\,], \sigma'\rangle$. Instantiating the goal with $\langle \mathbf{T}'''[\,c\,], \sigma\rangle$, from (c), (g), (i), (j), (k), (l), and (m), we conclude

– We consider (f) $\kappa' = \langle \mathbf{T}[\,\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,]\rangle\!\rangle_{\mathbf{p}} c\,], \sigma'\rangle$, (g) $\kappa'' = \mathsf{race}$, (h) $\langle c_1, \sigma'\rangle \xrightarrow[\delta_1]{}^* \langle c_1', \sigma''\rangle$, (i) $\langle c_2, \sigma''\rangle \xrightarrow[\delta_2]{}^* \kappa'''$, and (j) $\delta_1 \not\smile \delta_2$. We then consider 2 cases:

1. If (k) $\delta \not\curvearrowleft \delta_1$, from (c), (f), (e), (h), (k), and the semantics, we know (l)

$\kappa \Longrightarrow$ race. From (l), we conclude

2. If (k) $\delta \overset{\rightarrow}{\smile} \delta_1$, we consider another 2 cases:

   (a) If (l) $\delta_1 \not\overset{\rightarrow}{\smile} \delta$, from (e), (h), and (k), it is not hard to see that there exists δ_1', c_1'', σ''', and κ'''', such that (m) $\langle c_1, \sigma\rangle \underset{\delta_1'}{\longrightarrow}^* \langle c_1'', \sigma'''\rangle$, (n) $\langle c, \sigma'''\rangle \underset{\delta}{\longrightarrow} \kappa''''$, and (o) $\delta_1' \not\overset{\rightarrow}{\smile} \delta$. From (c), (f), (m), (n), (o), and the semantics, we know (p) $\kappa \Longrightarrow$ race. From (p), we conclude

   (b) If (l) $\delta_1 \overset{\rightarrow}{\smile} \delta$, from (e), (h), and (k), using Remark 5.7, we know there exists σ''' such that (m) $\langle c_1, \sigma\rangle \underset{\delta_1}{\longrightarrow}^* \langle c_1', \sigma'''\rangle$, and (n) $\langle c, \sigma'''\rangle \underset{\delta}{\longrightarrow} \langle c', \sigma''\rangle$. We then consider 2 cases:

      i. If (o) $\delta \overset{\rightarrow}{\smile} \delta_2$, then it is not hard to see that exists κ'''' such that (p) $\langle c_2, \sigma'''\rangle \underset{\delta_2}{\longrightarrow}^* \kappa''''$. From (c), (f), (m), (p), (j), and the semantics, we have (q) $\kappa \Longrightarrow$ race. From (q), we conclude

      ii. If (o) $\delta \not\smile \delta_2$, from (n) and (i), using Remark 5.8, we know there exists δ_2', δ_2'', c_2', σ'''', σ''''', and κ'''', such that (p) $\langle c_2, \sigma'''\rangle \underset{\delta_2'}{\longrightarrow}^* \langle c_2', \sigma''''\rangle$, (q) $\langle c, \sigma''''\rangle \underset{\delta}{\longrightarrow} \langle c', \sigma'''''\rangle$, (r) $\langle c_2', \sigma'''''\rangle \underset{\delta_2''}{\longrightarrow} \kappa''''$, (s) $\delta \smile \delta_2'$, and (t) $\delta \not\overset{\rightarrow}{\smile} \delta_2''$ (and there is a second case for which the proof is symmetric). We have another 2 cases: if (u) $\delta_1 \not\overset{\rightarrow}{\smile} \delta_2'$, from (m), (p), we have (v) $\kappa \Longrightarrow$ race. From (v), we conclude; if (u) $\delta_2' \not\overset{\rightarrow}{\frown} \delta_1$, we know there exists σ'''''', δ_1', κ''''' such that (v) $\langle c_2, \sigma\rangle \underset{\delta_2'}{\longrightarrow}^* \langle c_2', \sigma''''''\rangle$, (w) $\langle c_1, \sigma''''''\rangle \underset{\delta_1'}{\longrightarrow}^* \kappa'''''$, (x) $\delta_2' \not\overset{\rightarrow}{\frown} \delta_1'$. From (v), (w), and (x), we have (y) $\kappa \Longrightarrow$ race. From (y), we conclude; if (u) $\delta_1 \smile \delta_2'$, from (k), (l), and (s), we know there exists σ''''''' and κ'''''' such that (v) $\langle c, \sigma\rangle \underset{\delta}{\longrightarrow} \langle c', \sigma'''''''\rangle$ and (w) $\langle c_2, \sigma'''''''\rangle \underset{\delta_2'\cup\delta_2''}{\longrightarrow}^* \kappa''''''$. From (t), we know (x) $\delta \not\overset{\rightarrow}{\frown} \delta_2' \cup \delta_2''$. From (v), (w), and (x), and the semantics, we have (y) $\kappa \Longrightarrow$ race. From (y), we conclude

– The proof is symmetric to the previous case.

• We consider (c) $\kappa = \langle \mathbf{T}[\,\mathbf{S}[\,c_1 \,\|\, c_2\,]\,], \sigma\rangle$ and (d) $\kappa' = \langle \mathbf{T}[\,\langle\!\langle c_1, c_2\rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,])\,], \sigma\rangle$. From (c), (d), and the semantics, we know (e) $\kappa \Longrightarrow \kappa'$. From (e), (b), and the semantics, we know (f) $\kappa \Longrightarrow^* \kappa''$. From (f), we conclude

- We consider (c) $\kappa = \langle \mathbf{T}[\,\langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathbf{p}} c\,], \sigma\rangle$ and (d) $\kappa' = \langle \mathbf{T}[\,c\,], \sigma\rangle$. From (c), (d), and the semantics, we know (e) $\kappa \Longrightarrow \kappa'$. From (e), (b), and the semantics, we know (f) $\kappa \Longrightarrow^* \kappa''$. From (f), we conclude

- We consider (c) $\kappa = \langle \mathbf{T}[\,\mathbf{S}[\,\mathbf{atomic}\ c\,]\,], \sigma\rangle$, (d) $\kappa' = \langle \mathbf{T}[\,\langle\!\langle c \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\mathbf{skip}\,])\,], \sigma\rangle$, and (e) $\mathbf{T}$ is 0-atomic. From (c), (d), (e), and the semantics, we know (f) $\kappa \Longrightarrow \kappa'$. From (f), (b), and the semantics, we know (g) $\kappa \Longrightarrow^* \kappa''$. From (g), we conclude

- We consider (c) $\kappa = \langle \mathbf{T}[\,\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathbf{a}} c\,], \sigma\rangle$ and (d) $\kappa' = \langle \mathbf{T}[\,c\,], \sigma\rangle$. From (c), (d), and the semantics, we know (e) $\kappa \Longrightarrow \kappa'$. From (e), (b), and the semantics, we know (f) $\kappa \Longrightarrow^* \kappa''$. From (f), we conclude

- We consider (c) $\kappa = \langle \mathbf{T}[\,\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,] \rangle\!\rangle_{\mathbf{p}} c\,], \sigma\rangle$, (d) $\kappa' = \textsf{race}$, (e) $\langle c_1, \sigma\rangle \underset{\delta_1}{\longrightarrow} \langle c_1', \sigma'\rangle$, (f) $\langle c_2, \sigma'\rangle \underset{\delta_2}{\longrightarrow} \kappa'''$, and (g) $\delta_1 \not\smile \delta_2$. From (d), and the semantics, we know (b) is false and conclude

- We consider (c) $\kappa = \langle \mathbf{T}[\,\langle\!\langle \mathbf{T}_1[\,c_1\,], \mathbf{T}_2[\,c_2\,] \rangle\!\rangle_{\mathbf{p}} c\,], \sigma\rangle$, (d) $\kappa' = \textsf{race}$, (e) $\langle c_2, \sigma\rangle \underset{\delta_2}{\longrightarrow} \langle c_2', \sigma'\rangle$, (f) $\langle c_1, \sigma'\rangle \underset{\delta_1}{\longrightarrow} \kappa'''$, and (g) $\delta_2 \not\smile \delta_1$. From (d), and the semantics, we know (b) is false and conclude ☐

Relation between mixed-step semantics and subsumption. We can relate the execution of a subsumed program with the original one, more naturally, using the mixed-step semantics[1].

Lemma 5.10. If $T_1 \rightsquigarrow_{\mathbf{t}} T_2$, and $\neg\langle T_1, \sigma\rangle \Longrightarrow^* \textsf{abort}$, then

1. $\neg\langle T_2, \sigma\rangle \Longrightarrow^* \textsf{abort}$

2. If $\langle T_2, \sigma\rangle \Longrightarrow^* \langle \mathbf{skip}, \sigma'\rangle$, then $\langle T_1, \sigma\rangle \Longrightarrow^* \langle \mathbf{skip}, \sigma'\rangle$

[1]For simplicity, the proof presentation omits subsumption indexing details for subsumption, but they must be accounted for properly.

3. If $\langle T_2, \sigma\rangle \Longrightarrow^{\infty}$, then $\langle T_1, \sigma\rangle \Longrightarrow^{\infty}$

4. If $\langle T_2, \sigma\rangle \Longrightarrow^{*}$ race, then $\langle T_1, \sigma\rangle \Longrightarrow^{*}$ race

Proof. We assume (a) $T_1 \rightsquigarrow_{\mathsf{t}} T_2$, and (b) $\neg\langle T_1, \sigma\rangle \Longrightarrow^{*}$ abort, then we have 5 cases:

- We will assume (c) $\langle T_2, \sigma\rangle \Longrightarrow^{*}$ abort and, from (b), show that $\langle T_1, \sigma\rangle \Longrightarrow^{*}$ abort. By induction over n, the number of steps of (c), we have 2 cases: If (d) $n = 0$, from (c) and the semantics, we have (e) $\langle T_2, \sigma\rangle =$ abort. Given that (e) is false, we conclude; If (d) $n > 0$, from (c) and the semantics, we know there exists κ such that (e) $\langle T_2, \sigma\rangle \Longrightarrow \kappa$ and (f) $\kappa \Longrightarrow^{n-1}$ abort. From (e), and the semantics, we have 8 cases:
  - We consider (g) $T_2 = \mathbf{T}_2[\,c_2\,]$, (h) $\kappa =$ abort, and (i) $\langle c_2, \sigma\rangle \longrightarrow^{*}$ abort. From (a) and (g), by Def. 3.23, we know there exists $\mathbf{T}_1$ and c_1 such that (j) $T_1 = \mathbf{T}_1[\,c_1\,]$ and (k) $c_1 \rightsquigarrow c_2$. From (k) and (i), by Def. 3.33, we know (l) $\langle c_1, \sigma\rangle \longrightarrow^{*}$ abort. From (l), and the semantics, we have (m) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow$ abort. From (m), we conclude
  - We consider (g) $T_2 = \mathbf{T}_2[\,c_2\,]$, (h) $\kappa = \langle \mathbf{T}_2[\,c_2'\,], \sigma'\rangle$, (i) $\langle c_2, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$, and (j) $c_2 \neq c_2'$. From (a) and (g), by Def. 3.23, we know there exists $\mathbf{T}_1$ and c_1 such that (j) $T_1 = \mathbf{T}_1[\,c_1\,]$, (k) $c_1 \rightsquigarrow c_2$, and (l) for all c_1' and c_2', if $c_1' \rightsquigarrow c_2'$, then $\mathbf{T}_1[\,c_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,c_2'\,]$. From (k) and (i), we have 2 cases:
    * We consider (m) $\langle c_1, \sigma\rangle \longrightarrow^{*}$ abort. From (m), and the semantics, we have (n) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow$ abort. From (n), we conclude
    * We consider (m) $\langle c_1, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and (n) $c_1' \rightsquigarrow c_2'$. From (n) and (l), we have (o) $\mathbf{T}_1[\,c_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,c_2'\,]$. From (o), (h), and (f), by the induction hypothesis, we have (p) $\langle \mathbf{T}_1[\,c_1'\,], \sigma'\rangle \Longrightarrow^{*}$ abort. We have then 2 cases:
      · If (q) $c_1 = c_1'$, from (m), using Remark 3.32, we know (r) $\sigma = \sigma'$. From (q), (r), and (p), we conclude

  · If (q) $c_1 \neq c_1'$, from (m) and the semantics, we have (r) $\langle \mathbf{T}_1[\, c_1\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\, c_1'\,], \sigma'\rangle$. From (r), (p), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\, \mathbf{S}_2[\, c_2 \parallel c_2'\,]\,]$ and (h) $\kappa = \langle \mathbf{T}_2[\, \langle\!\langle c_2, c_2' \rangle\!\rangle_{\mathsf{p}}(\mathbf{S}_2[\, \mathbf{skip}\,])\,], \sigma\rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$ and c_1 such that (i) $T_1 = \mathbf{T}_1[\, c_1\,]$, (j) $c_1 \rightsquigarrow \mathbf{S}_2[\, c_2 \parallel c_2'\,]$, and (k) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathsf{t}} T_2'$, then $\mathbf{T}_1[\, T_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\, T_2'\,]$. From (j), by Def. 3.33, we have 2 cases:

  * We consider (l) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (l), and the semantics, we know (m) $\langle \mathbf{T}_1[\, c_1\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i) and (m), we conclude

  * We consider (l) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{S}_1[\, c_1' \parallel c_1''\,], \sigma\rangle$, (m) $c_1' \rightsquigarrow c_2$, (n) $c_1'' \rightsquigarrow c_2'$, and (o) $\mathbf{S}_1[\, \mathbf{skip}\,] \rightsquigarrow \mathbf{S}_2[\, \mathbf{skip}\,]$. From (l), and the semantics, we know (p) $\langle \mathbf{T}_1[\, c_1\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\, \mathbf{S}_1[\, c_1' \parallel c_1''\,]\,], \sigma\rangle$. From the semantics, we know (q) $\langle \mathbf{T}_1[\, \mathbf{S}_1[\, c_1' \parallel c_1''\,]\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\, \langle\!\langle c_1', c_1'' \rangle\!\rangle_{\mathsf{p}}(\mathbf{S}_1[\, \mathbf{skip}\,])\,], \sigma\rangle$. From (m), (n), and (o), we know (r) $\langle\!\langle c_1', c_1'' \rangle\!\rangle_{\mathsf{p}}(\mathbf{S}_1[\, \mathbf{skip}\,]) \rightsquigarrow_{\mathsf{t}} \langle\!\langle c_2, c_2' \rangle\!\rangle_{\mathsf{p}}(\mathbf{S}_2[\, \mathbf{skip}\,])$. From (r) and (k), we have (s) $\mathbf{T}_1[\, \langle\!\langle c_1', c_1'' \rangle\!\rangle_{\mathsf{p}}(\mathbf{S}_1[\, \mathbf{skip}\,])\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\, \langle\!\langle c_2, c_2' \rangle\!\rangle_{\mathsf{p}}(\mathbf{S}_2[\, \mathbf{skip}\,])\,]$. From (s), (h), and (f), using the induction hypothesis, we have (t) $\langle \mathbf{T}_1[\, \langle\!\langle c_1', c_1'' \rangle\!\rangle_{\mathsf{p}}(\mathbf{S}_1[\, \mathbf{skip}\,])\,], \sigma\rangle \Longrightarrow^* \mathsf{abort}$. From (p), (q), (t), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\, \langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathsf{p}} c_2\,]$ and (h) $\kappa = \langle \mathbf{T}_2[\, c_2\,], \sigma\rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$, c_1, c_1', and c_1'', such that (i) $T_1 = \mathbf{T}_1[\, \langle\!\langle c_1, c_1' \rangle\!\rangle_{\mathsf{p}} c_1''\,]$, (j) $c_1 \rightsquigarrow \mathbf{skip}$, (k) $c_1' \rightsquigarrow \mathbf{skip}$, (l) $c_1'' \rightsquigarrow c_2$, and (m) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathsf{t}} T_2'$, then $\mathbf{T}_1[\, T_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\, T_2'\,]$. From (j), by Def. 3.33, we have 2 cases:

  * We consider (n) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\, \langle\!\langle c_1, c_1' \rangle\!\rangle_{\mathsf{p}} c_1''\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i) and (o), we conclude

  * We consider (n) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma\rangle$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\, \langle\!\langle c_1, c_1' \rangle\!\rangle_{\mathsf{p}} c_1''\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\, \langle\!\langle \mathbf{skip}, c_1' \rangle\!\rangle_{\mathsf{p}} c_1''\,], \sigma\rangle$. From (k), by Def. 3.33, we have 2 cases:

· We consider (p) $\langle c_1', \sigma\rangle \longrightarrow^* \textsf{abort}$. From (p), and the semantics, we know (q) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, c_1' \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma\rangle \Longrightarrow \textsf{abort}$. From (i), (o), and (q), we conclude

· We consider (p) $\langle c_1', \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma\rangle$. From (p), and the semantics, we know (q) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, c_1' \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma\rangle$. From the semantics, we know (r) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathbf{p}} c_1'\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,c_1'\,], \sigma\rangle$. From (l) and (m), we have (s) $\mathbf{T}_1[\,c_1'\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,c_2\,]$. From (s), (h), and (f), using the induction hypothesis, we have (t) $\langle \mathbf{T}_1[\,c_1'\,], \sigma\rangle \Longrightarrow^* \textsf{abort}$. From (o), (q), (r), (t), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\,\mathbf{S}_2[\,\mathbf{atomic}\; c_2\,]\,]$, (h) $\kappa = \langle \mathbf{T}_2[\,\langle\!\langle c_2 \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,], \sigma\rangle$, and (i) $\mathbf{T}_2$ is 0-atomic. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$ and c_1 such that (j) $T_1 = \mathbf{T}_1[\,c_1\,]$, (k) $c_1 \rightsquigarrow \mathbf{S}_2[\,\mathbf{atomic}\; c_2\,]$, (l) $\mathbf{T}_1$ is 0-atomic, and (m) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathbf{t}} T_2'$, then $\mathbf{T}_1[\,T_1'\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,T_2'\,]$. From (k), by Def. 3.33, we have 2 cases:

  ∗ We consider (n) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow \textsf{abort}$. From (j) and (o), we conclude

  ∗ We consider (n) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{S}_1[\,\mathbf{atomic}\; c_1'\,], \sigma\rangle$, (o) $c_1' \rightsquigarrow c_2$, and (p) $\mathbf{S}_1[\,\mathbf{skip}\,] \rightsquigarrow \mathbf{S}_2[\,\mathbf{skip}\,]$. From (n), and the semantics, we know (q) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\mathbf{S}_1[\,\mathbf{atomic}\; c_1'\,]\,], \sigma\rangle$. From (l) and the semantics, we know (r) $\langle \mathbf{T}_1[\,\mathbf{S}_1[\,\mathbf{atomic}\; c_1'\,]\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,\langle\!\langle c_1' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,], \sigma\rangle$. From (o) and (p), we know (s) $\langle\!\langle c_1' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,]) \rightsquigarrow_{\mathbf{t}} \langle\!\langle c_2 \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])$. From (s) and (m), we have (t) $\mathbf{T}_1[\,\langle\!\langle c_1' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,\langle\!\langle c_2 \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,]$. From (t), (h), and (f), using the induction hypothesis, we have (u) $\langle \mathbf{T}_1[\,\langle\!\langle c_1' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,], \sigma\rangle \Longrightarrow^* \textsf{abort}$. From (q), (r), (u), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\,\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathbf{a}} c_2\,]$ and (h) $\kappa = \langle \mathbf{T}_2[\,c_2\,], \sigma\rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$, c_1, and c_1', such that

(i) $T_1 = \mathbf{T}_1[\,\langle\!\langle c_1 \rangle\!\rangle_{\mathsf{a}} c_1'\,]$, (j) $c_1 \rightsquigarrow \textbf{skip}$, (k) $c_1' \rightsquigarrow c_2$, and (l) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathsf{t}} T_2'$, then $\mathbf{T}_1[\,T_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,T_2'\,]$. From (j), by Def. 3.33, we have 2 cases:

  * We consider (m) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (m), and the semantics, we know (n) $\langle \mathbf{T}_1[\,\langle\!\langle c_1 \rangle\!\rangle_{\mathsf{a}} c_1'\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i) and (n), we conclude
  * We consider (m) $\langle c_1, \sigma\rangle \Downarrow \langle \textbf{skip}, \sigma\rangle$. From (m), and the semantics, we know (n) $\langle \mathbf{T}_1[\,\langle\!\langle c_1 \rangle\!\rangle_{\mathsf{a}} c_1'\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\langle\!\langle \textbf{skip} \rangle\!\rangle_{\mathsf{a}} c_1'\,], \sigma\rangle$. From the semantics, we know (o) $\langle \mathbf{T}_1[\,\langle\!\langle \textbf{skip} \rangle\!\rangle_{\mathsf{a}} c_1'\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,c_1'\,], \sigma\rangle$. From (k) and (l), we have (p) $\mathbf{T}_1[\,c_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,c_2\,]$. From (p), (h), and (f), using the induction hypothesis, we have (q) $\langle \mathbf{T}_1[\,c_1'\,], \sigma\rangle \Longrightarrow^* \mathsf{abort}$. From (n), (o), (q), and the semantics, we conclude

- We consider (g) $T_2 = \mathbf{T}_2[\,\langle\!\langle \mathbf{T}_2'[\,c_2'\,], \mathbf{T}_2''[\,c_2''\,]\rangle\!\rangle_{\mathsf{p}} c_2\,]$, (h) $\kappa = \mathsf{race}$, (i) $\langle c_2', \sigma\rangle \underset{\delta_2}{\longrightarrow^*} \langle c_2''', \sigma'\rangle$, (j) $\langle c_2'', \sigma'\rangle \underset{\delta_2'}{\longrightarrow^*} \kappa_2$, and (k) $\delta_2 \not\smile \delta_2'$. From (h), and the semantics, we know (f) is false and conclude
- We consider (g) $T_2 = \mathbf{T}_2[\,\langle\!\langle \mathbf{T}_2'[\,c_2'\,], \mathbf{T}_2''[\,c_2''\,]\rangle\!\rangle_{\mathsf{p}} c_2\,]$, (h) $\kappa = \mathsf{race}$, (i) $\langle c_2'', \sigma\rangle \underset{\delta_2}{\longrightarrow^*} \langle c_2''', \sigma'\rangle$, (j) $\langle c_2', \sigma'\rangle \underset{\delta_2'}{\longrightarrow^*} \kappa_2$, and (k) $\delta_2 \not\smile \delta_2'$. From (h), and the semantics, we know (f) is false and conclude

• We assume (c) $\langle T_2, \sigma\rangle \Longrightarrow^* \langle \textbf{skip}, \sigma''\rangle$. By induction over n, the number of steps of (c), we have 2 cases: If (d) $n = 0$, from (c) and the semantics, we have (e) $\langle T_2, \sigma\rangle = \langle \textbf{skip}, \sigma''\rangle$. From (a) and (e), by Def. 3.23, we know (f) $T_1 = c_1$ and (g) $c_1 \rightsquigarrow \textbf{skip}$. From (g), by Def. 3.33, we have 2 cases:

  - We consider (h) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (h), and the semantics, we know (i) $\langle c_1, \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i), and (b), we find a contradiction and conclude
  - We consider (h) $\langle c_1, \sigma\rangle \Downarrow \langle \textbf{skip}, \sigma\rangle$. From (h), and the semantics, we know (i) $\langle c_1, \sigma\rangle \Longrightarrow^* \langle \textbf{skip}, \sigma\rangle$. From (i), we conclude

If (d) $n > 0$, from (c) and the semantics, we know there exists κ such that (e) $\langle T_2, \sigma\rangle \Longrightarrow \kappa$ and (f) $\kappa \Longrightarrow^{n-1} \langle \textbf{skip}, \sigma''\rangle$. From (e), and the semantics, we have

8 cases:

– We consider (g) $T_2 = \mathbf{T}_2[\, c_2 \,]$, (h) $\kappa = \textsf{abort}$, and (i) $\langle c_2, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (h), and the semantics, we know (f) is false and conclude

– We consider (g) $T_2 = \mathbf{T}_2[\, c_2 \,]$, (h) $\kappa = \langle \mathbf{T}_2[\, c_2' \,], \sigma' \rangle$, (i) $\langle c_2, \sigma \rangle \Downarrow \langle c_2', \sigma' \rangle$, and (j) $c_2 \neq c_2'$. From (a) and (g), by Def. 3.23, we know there exists $\mathbf{T}_1$ and c_1 such that (j) $T_1 = \mathbf{T}_1[\, c_1 \,]$, (k) $c_1 \rightsquigarrow c_2$, and (l) for all c_1' and c_2', if $c_1' \rightsquigarrow c_2'$, then $\mathbf{T}_1[\, c_1' \,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\, c_2' \,]$. From (k) and (i), we have 2 cases:

  ∗ We consider (m) $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (m), and the semantics, we have (n) $\langle \mathbf{T}_1[\, c_1 \,], \sigma \rangle \Longrightarrow \textsf{abort}$. From (n) and (b), we find a contradiction and conclude

  ∗ We consider (m) $\langle c_1, \sigma \rangle \Downarrow \langle c_1', \sigma' \rangle$ and (n) $c_1' \rightsquigarrow c_2'$. From (n) and (l), we have (o) $\mathbf{T}_1[\, c_1' \,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\, c_2' \,]$. From (o), (h), and (f), by the induction hypothesis, we have (p) $\langle \mathbf{T}_1[\, c_1' \,], \sigma' \rangle \Longrightarrow^* \langle \mathbf{skip}, \sigma'' \rangle$. We have then 2 cases:

    · If (q) $c_1 = c_1'$, from (m), using Remark 3.32, we know (r) $\sigma = \sigma'$. From (q), (r), and (p), we conclude

    · If (q) $c_1 \neq c_1'$, from (m) and the semantics, we have (r) $\langle \mathbf{T}_1[\, c_1 \,], \sigma \rangle \Longrightarrow \langle \mathbf{T}_1[\, c_1' \,], \sigma' \rangle$. From (r), (p), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\, \mathbf{S}_2[\, c_2 \parallel c_2' \,] \,]$ and (h) $\kappa = \langle \mathbf{T}_2[\, \langle\!\langle c_2, c_2' \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_2[\, \mathbf{skip} \,]) \,], \sigma \rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$ and c_1 such that (i) $T_1 = \mathbf{T}_1[\, c_1 \,]$, (j) $c_1 \rightsquigarrow \mathbf{S}_2[\, c_2 \parallel c_2' \,]$, and (k) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathbf{t}} T_2'$, then $\mathbf{T}_1[\, T_1' \,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\, T_2' \,]$. From (j), by Def. 3.33, we have 2 cases:

  ∗ We consider (l) $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (l), and the semantics, we know (m) $\langle \mathbf{T}_1[\, c_1 \,], \sigma \rangle \Longrightarrow \textsf{abort}$. From (i), (m), and (b), we find a contradiction and conclude

  ∗ We consider (l) $\langle c_1, \sigma \rangle \Downarrow \langle \mathbf{S}_1[\, c_1' \parallel c_1'' \,], \sigma \rangle$, (m) $c_1' \rightsquigarrow c_2$, (n) $c_1'' \rightsquigarrow c_2'$, and (o) $\mathbf{S}_1[\, \mathbf{skip} \,] \rightsquigarrow \mathbf{S}_2[\, \mathbf{skip} \,]$. From (l), and the semantics, we know (p) $\langle \mathbf{T}_1[\, c_1 \,], \sigma \rangle \Longrightarrow^* \langle \mathbf{T}_1[\, \mathbf{S}_1[\, c_1' \parallel c_1'' \,] \,], \sigma \rangle$. From the semantics, we know (q)

$\langle \mathbf{T}_1[\,\mathbf{S}_1[\,c_1' \parallel c_1''\,]\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,\langle\!\langle c_1', c_1''\rangle\!\rangle_{\mathtt{p}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,], \sigma\rangle$. From (m), (n), and (o), we know (r) $\langle\!\langle c_1', c_1''\rangle\!\rangle_{\mathtt{p}}(\mathbf{S}_1[\,\mathbf{skip}\,]) \rightsquigarrow_{\mathtt{t}} \langle\!\langle c_2, c_2'\rangle\!\rangle_{\mathtt{p}}(\mathbf{S}_2[\,\mathbf{skip}\,])$. From (r) and (k), we have (s) $\mathbf{T}_1[\,\langle\!\langle c_1', c_1''\rangle\!\rangle_{\mathtt{p}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,] \rightsquigarrow_{\mathtt{t}} \mathbf{T}_2[\,\langle\!\langle c_2, c_2'\rangle\!\rangle_{\mathtt{p}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,]$. From (s), (h), and (f), using the induction hypothesis, we have (t) $\langle \mathbf{T}_1[\,\langle\!\langle c_1', c_1''\rangle\!\rangle_{\mathtt{p}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{skip}, \sigma''\rangle$. From (p), (q), (t), and the semantics, we conclude

- We consider (g) $T_2 = \mathbf{T}_2[\,\langle\!\langle \mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathtt{p}} c_2\,]$ and (h) $\kappa = \langle \mathbf{T}_2[\,c_2\,], \sigma\rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$, c_1, c_1', and c_1'', such that (i) $T_1 = \mathbf{T}_1[\,\langle\!\langle c_1, c_1'\rangle\!\rangle_{\mathtt{p}} c_1''\,]$, (j) $c_1 \rightsquigarrow \mathbf{skip}$, (k) $c_1' \rightsquigarrow \mathbf{skip}$, (l) $c_1'' \rightsquigarrow c_2$, and (m) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathtt{t}} T_2'$, then $\mathbf{T}_1[\,T_1'\,] \rightsquigarrow_{\mathtt{t}} \mathbf{T}_2[\,T_2'\,]$. From (j), by Def. 3.33, we have 2 cases:

  * We consider (n) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\,\langle\!\langle c_1, c_1'\rangle\!\rangle_{\mathtt{p}} c_1''\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i), (o), and (b), we find a contradiction and conclude

  * We consider (n) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma\rangle$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\,\langle\!\langle c_1, c_1'\rangle\!\rangle_{\mathtt{p}} c_1''\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, c_1'\rangle\!\rangle_{\mathtt{p}} c_1''\,], \sigma\rangle$. From (k), by Def. 3.33, we have 2 cases:

    · We consider (p) $\langle c_1', \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (p), and the semantics, we know (q) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, c_1'\rangle\!\rangle_{\mathtt{p}} c_1''\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i), (o), (q), and (b), we find a contradiction and conclude

    · We consider (p) $\langle c_1', \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma\rangle$. From (p), and the semantics, we know (q) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, c_1'\rangle\!\rangle_{\mathtt{p}} c_1''\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathtt{p}} c_1''\,], \sigma\rangle$. From the semantics, we know (r) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathtt{p}} c_1'\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,c_1'\,], \sigma\rangle$. From (l) and (m), we have (s) $\mathbf{T}_1[\,c_1'\,] \rightsquigarrow_{\mathtt{t}} \mathbf{T}_2[\,c_2\,]$. From (s), (h), and (f), using the induction hypothesis, we have (t) $\langle \mathbf{T}_1[\,c_1'\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{skip}, \sigma''\rangle$. From (o), (q), (r), (t), and the semantics, we conclude

- We consider (g) $T_2 = \mathbf{T}_2[\,\mathbf{S}_2[\,\mathbf{atomic}\; c_2\,]\,]$, (h) $\kappa = \langle \mathbf{T}_2[\,\langle\!\langle c_2\rangle\!\rangle_{\mathtt{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,], \sigma\rangle$, and

(i) $\mathbf{T}_2$ is 0-atomic. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$ and c_1 such that (j) $T_1 = \mathbf{T}_1[\,c_1\,]$, (k) $c_1 \rightsquigarrow \mathbf{S}_2[\,\textbf{atomic}\ c_2\,]$, (l) $\mathbf{T}_1$ is 0-atomic, and (m) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathsf{t}} T_2'$, then $\mathbf{T}_1[\,T_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,T_2'\,]$. From (k), by Def. 3.33, we have 2 cases:

- $*$ We consider (n) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (j), (o), and (b), we find a contradiction and conclude
- $*$ We consider (n) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{S}_1[\,\textbf{atomic}\ c_1'\,], \sigma\rangle$, (o) $c_1' \rightsquigarrow c_2$, and (p) $\mathbf{S}_1[\,\textbf{skip}\,] \rightsquigarrow \mathbf{S}_2[\,\textbf{skip}\,]$. From (n), and the semantics, we know (q) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\mathbf{S}_1[\,\textbf{atomic}\ c_1'\,]\,], \sigma\rangle$. From (l) and the semantics, we know (r) $\langle \mathbf{T}_1[\,\mathbf{S}_1[\,\textbf{atomic}\ c_1'\,]\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,\langle\!\langle c_1'\rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_1[\,\textbf{skip}\,])\,], \sigma\rangle$. From (o) and (p), we know (s) $\langle\!\langle c_1'\rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_1[\,\textbf{skip}\,]) \rightsquigarrow_{\mathsf{t}} \langle\!\langle c_2\rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_2[\,\textbf{skip}\,])$. From (s) and (m), we have (t) $\mathbf{T}_1[\,\langle\!\langle c_1'\rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_1[\,\textbf{skip}\,])\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,\langle\!\langle c_2\rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_2[\,\textbf{skip}\,])\,]$. From (t), (h), and (f), using the induction hypothesis, we have (u) $\langle \mathbf{T}_1[\,\langle\!\langle c_1'\rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_1[\,\textbf{skip}\,])\,], \sigma\rangle \Longrightarrow^* \langle \textbf{skip}, \sigma''\rangle$. From (q), (r), (u), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\,\langle\!\langle \textbf{skip}\rangle\!\rangle_{\mathsf{a}} c_2\,]$ and (h) $\kappa = \langle \mathbf{T}_2[\,c_2\,], \sigma\rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$, c_1, and c_1', such that (i) $T_1 = \mathbf{T}_1[\,\langle\!\langle c_1\rangle\!\rangle_{\mathsf{a}} c_1'\,]$, (j) $c_1 \rightsquigarrow \textbf{skip}$, (k) $c_1' \rightsquigarrow c_2$, and (l) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathsf{t}} T_2'$, then $\mathbf{T}_1[\,T_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,T_2'\,]$. From (j), by Def. 3.33, we have 2 cases:

- $*$ We consider (m) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (m), and the semantics, we know (n) $\langle \mathbf{T}_1[\,\langle\!\langle c_1\rangle\!\rangle_{\mathsf{a}} c_1'\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i), (n), and (b), we find a contradiction and conclude
- $*$ We consider (m) $\langle c_1, \sigma\rangle \Downarrow \langle \textbf{skip}, \sigma\rangle$. From (m), and the semantics, we know (n) $\langle \mathbf{T}_1[\,\langle\!\langle c_1\rangle\!\rangle_{\mathsf{a}} c_1'\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\langle\!\langle \textbf{skip}\rangle\!\rangle_{\mathsf{a}} c_1'\,], \sigma\rangle$. From the semantics, we know (o) $\langle \mathbf{T}_1[\,\langle\!\langle \textbf{skip}\rangle\!\rangle_{\mathsf{a}} c_1'\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,c_1'\,], \sigma\rangle$. From (k) and (l), we have (p) $\mathbf{T}_1[\,c_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,c_2\,]$. From (p), (h), and (f), using the induction hypoth-

esis, we have (q) $\langle \mathbf{T}_1[\,c'_1\,], \sigma \rangle \Longrightarrow^* \langle \mathbf{skip}, \sigma'' \rangle$. From (n), (o), (q), and the semantics, we conclude

  - We consider (g) $T_2 = \mathbf{T}_2[\,\langle\!\langle \mathbf{T}'_2[\,c'_2\,], \mathbf{T}''_2[\,c''_2\,] \rangle\!\rangle_{\mathsf{p}} c_2\,]$, (h) $\kappa = \mathsf{race}$, (i) $\langle c'_2, \sigma \rangle \underset{\delta_2}{\longrightarrow}^* \langle c'''_2, \sigma' \rangle$, (j) $\langle c''_2, \sigma' \rangle \underset{\delta'_2}{\longrightarrow}^* \kappa_2$, and (k) $\delta_2 \not\smile \delta'_2$. From (h), and the semantics, we know (f) is false and conclude
  - We consider (g) $T_2 = \mathbf{T}_2[\,\langle\!\langle \mathbf{T}'_2[\,c'_2\,], \mathbf{T}''_2[\,c''_2\,] \rangle\!\rangle_{\mathsf{p}} c_2\,]$, (h) $\kappa = \mathsf{race}$, (i) $\langle c''_2, \sigma \rangle \underset{\delta_2}{\longrightarrow}^* \langle c'''_2, \sigma' \rangle$, (j) $\langle c'_2, \sigma' \rangle \underset{\delta'_2}{\longrightarrow}^* \kappa_2$, and (k) $\delta_2 \not\smile \delta'_2$. From (h), and the semantics, we know (f) is false and conclude

- From $\langle T_2, \sigma \rangle \Longrightarrow^\infty$, by Def. 5.3, we assume (c) $\langle T_2, \sigma \rangle \Longrightarrow^\infty$, and will show that, for all n, there exists κ''_1 such that $\langle T_1, \sigma \rangle \Longrightarrow^n \kappa''_1$. By induction over n, the number of steps of (c), we have 2 cases: If $n = 0$, from (c) and the semantics, we instantiating the goal with $\langle T_1, \sigma \rangle$ and conclude If $n > 0$, from (c) and the semantics, we know there exists κ such that (e) $\langle T_2, \sigma \rangle \Longrightarrow \kappa$ and (f) $\kappa \Longrightarrow^\infty$. From (e), and the semantics, we have 8 cases:

  - We consider (g) $T_2 = \mathbf{T}_2[\,c_2\,]$, (h) $\kappa = \mathsf{abort}$, and (i) $\langle c_2, \sigma \rangle \longrightarrow^* \mathsf{abort}$. From (h), and the semantics, we know (f) is false and conclude
  - We consider (g) $T_2 = \mathbf{T}_2[\,c_2\,]$, (h) $\kappa = \langle \mathbf{T}_2[\,c'_2\,], \sigma' \rangle$, (i) $\langle c_2, \sigma \rangle \Downarrow \langle c'_2, \sigma' \rangle$, and (j) $c_2 \neq c'_2$. From (a) and (g), by Def. 3.23, we know there exists $\mathbf{T}_1$ and c_1 such that (j) $T_1 = \mathbf{T}_1[\,c_1\,]$, (k) $c_1 \rightsquigarrow c_2$, and (l) for all c'_1 and c'_2, if $c'_1 \rightsquigarrow c'_2$, then $\mathbf{T}_1[\,c'_1\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,c'_2\,]$. From (k) and (i), we have 2 cases:
    - ∗ We consider (m) $\langle c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$. From (m), and the semantics, we have (n) $\langle \mathbf{T}_1[\,c_1\,], \sigma \rangle \Longrightarrow \mathsf{abort}$. From (n) and (b), we find a contradiction and conclude
    - ∗ We consider (m) $\langle c_1, \sigma \rangle \Downarrow \langle c'_1, \sigma' \rangle$ and (n) $c'_1 \rightsquigarrow c'_2$. From (n) and (l), we have (o) $\mathbf{T}_1[\,c'_1\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,c'_2\,]$. From (o), (h), and (f), by the induction hypothesis, we have (p) $\langle \mathbf{T}_1[\,c'_1\,], \sigma' \rangle \Longrightarrow^{n-1} \kappa''_1$. We have then 2 cases:

- If (q) $c_1 = c'_1$, from (m), using Remark 3.32, we know (r) $\sigma = \sigma'$. Instantiating the goal with κ''_1, from (q), (r), and (p), we conclude
- If (q) $c_1 \neq c'_1$, from (m) and the semantics, we have (r) $\langle \mathbf{T}_1[\, c_1 \,], \sigma \rangle \Longrightarrow \langle \mathbf{T}_1[\, c'_1 \,], \sigma' \rangle$. Instantiating the goal with κ''_1, from (r), (p), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\, \mathbf{S}_2[\, c_2 \parallel c'_2 \,]\,]$ and (h) $\kappa = \langle \mathbf{T}_2[\, \langle\!\langle c_2, c'_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_2[\, \mathbf{skip} \,])\,], \sigma \rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$ and c_1 such that (i) $T_1 = \mathbf{T}_1[\, c_1 \,]$, (j) $c_1 \rightsquigarrow \mathbf{S}_2[\, c_2 \parallel c'_2 \,]$, and (k) for all T'_1 and T'_2, if $T'_1 \rightsquigarrow_{\mathbf{t}} T'_2$, then $\mathbf{T}_1[\, T'_1 \,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\, T'_2 \,]$. From (j), by Def. 3.33, we have 2 cases:

  * We consider (l) $\langle c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$. From (l), and the semantics, we know (m) $\langle \mathbf{T}_1[\, c_1 \,], \sigma \rangle \Longrightarrow \mathsf{abort}$. From (i), (m), and (b), we find a contradiction and conclude
  * We consider (l) $\langle c_1, \sigma \rangle \Downarrow \langle \mathbf{S}_1[\, c'_1 \parallel c''_1 \,], \sigma \rangle$, (m) $c'_1 \rightsquigarrow c_2$, (n) $c''_1 \rightsquigarrow c'_2$, and (o) $\mathbf{S}_1[\, \mathbf{skip} \,] \rightsquigarrow \mathbf{S}_2[\, \mathbf{skip} \,]$. From (l), and the semantics, we know (p) $\langle \mathbf{T}_1[\, c_1 \,], \sigma \rangle \Longrightarrow^* \langle \mathbf{T}_1[\, \mathbf{S}_1[\, c'_1 \parallel c''_1 \,]\,], \sigma \rangle$. From the semantics, we know (q) $\langle \mathbf{T}_1[\, \mathbf{S}_1[\, c'_1 \parallel c''_1 \,]\,], \sigma \rangle \Longrightarrow \langle \mathbf{T}_1[\, \langle\!\langle c'_1, c''_1 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_1[\, \mathbf{skip} \,])\,], \sigma \rangle$. From (m), (n), and (o), we know (r) $\langle\!\langle c'_1, c''_1 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_1[\, \mathbf{skip} \,]) \rightsquigarrow_{\mathbf{t}} \langle\!\langle c_2, c'_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_2[\, \mathbf{skip} \,])$. From (r) and (k), we have (s) $\mathbf{T}_1[\, \langle\!\langle c'_1, c''_1 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_1[\, \mathbf{skip} \,])\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\, \langle\!\langle c_2, c'_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_2[\, \mathbf{skip} \,])\,]$. From (s), (h), and (f), using the induction hypothesis, we have (t) $\langle \mathbf{T}_1[\, \langle\!\langle c'_1, c''_1 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_1[\, \mathbf{skip} \,])\,], \sigma \rangle \Longrightarrow^{n-1} \kappa''_1$. Instantiating the goal with κ''_1, from (p), (q), (t), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\, \langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathbf{p}} c_2 \,]$ and (h) $\kappa = \langle \mathbf{T}_2[\, c_2 \,], \sigma \rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$, c_1, c'_1, and c''_1, such that (i) $T_1 = \mathbf{T}_1[\, \langle\!\langle c_1, c'_1 \rangle\!\rangle_{\mathbf{p}} c''_1 \,]$, (j) $c_1 \rightsquigarrow \mathbf{skip}$, (k) $c'_1 \rightsquigarrow \mathbf{skip}$, (l) $c''_1 \rightsquigarrow c_2$, and (m) for all T'_1 and T'_2, if $T'_1 \rightsquigarrow_{\mathbf{t}} T'_2$, then $\mathbf{T}_1[\, T'_1 \,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\, T'_2 \,]$. From (j), by Def. 3.33, we have 2 cases:

  * We consider (n) $\langle c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$. From (n), and the semantics, we know

(o) $\langle \mathbf{T}_1[\,\langle\!\langle c_1, c_1' \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma \rangle \Longrightarrow \textsf{abort}$. From (i), (o), and (b), we find a contradiction and conclude

* We consider (n) $\langle c_1, \sigma \rangle \Downarrow \langle \mathbf{skip}, \sigma \rangle$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\,\langle\!\langle c_1, c_1' \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma \rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, c_1' \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma \rangle$. From (k), by Def. 3.33, we have 2 cases:
  - We consider (p) $\langle c_1', \sigma \rangle \longrightarrow^* \textsf{abort}$. From (p), and the semantics, we know (q) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, c_1' \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma \rangle \Longrightarrow \textsf{abort}$. From (i), (o), (q), and (b), we find a contradiction and conclude
  - We consider (p) $\langle c_1', \sigma \rangle \Downarrow \langle \mathbf{skip}, \sigma \rangle$. From (p), and the semantics, we know (q) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, c_1' \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma \rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma \rangle$. From the semantics, we know (r) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathbf{p}} c_1'\,], \sigma \rangle \Longrightarrow \langle \mathbf{T}_1[\,c_1'\,], \sigma \rangle$. From (l) and (m), we have (s) $\mathbf{T}_1[\,c_1'\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,c_2\,]$. From (s), (h), and (f), using the induction hypothesis, we have
    (t) $\langle \mathbf{T}_1[\,c_1'\,], \sigma \rangle \Longrightarrow^* \kappa_1''$. Instantiating the goal with κ_1'', from (o), (q), (r), (t), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\,\mathbf{S}_2[\,\mathbf{atomic}\; c_2\,]\,]$, (h) $\kappa = \langle \mathbf{T}_2[\,\langle\!\langle c_2 \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,], \sigma \rangle$, and (i) $\mathbf{T}_2$ is 0-atomic. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$ and c_1 such that (j) $T_1 = \mathbf{T}_1[\,c_1\,]$, (k) $c_1 \rightsquigarrow \mathbf{S}_2[\,\mathbf{atomic}\; c_2\,]$, (l) $\mathbf{T}_1$ is 0-atomic, and (m) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathbf{t}} T_2'$, then $\mathbf{T}_1[\,T_1'\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,T_2'\,]$. From (k), by Def. 3.33, we have 2 cases:

  * We consider (n) $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\,c_1\,], \sigma \rangle \Longrightarrow \textsf{abort}$. From (j), (o), and (b), we find a contradiction and conclude
  * We consider (n) $\langle c_1, \sigma \rangle \Downarrow \langle \mathbf{S}_1[\,\mathbf{atomic}\; c_1'\,], \sigma \rangle$, (o) $c_1' \rightsquigarrow c_2$, and (p) $\mathbf{S}_1[\,\mathbf{skip}\,] \rightsquigarrow \mathbf{S}_2[\,\mathbf{skip}\,]$. From (n), and the semantics, we know (q) $\langle \mathbf{T}_1[\,c_1\,], \sigma \rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\mathbf{S}_1[\,\mathbf{atomic}\; c_1'\,]\,], \sigma \rangle$. From (l) and the semantics, we know
    (r) $\langle \mathbf{T}_1[\,\mathbf{S}_1[\,\mathbf{atomic}\; c_1'\,]\,], \sigma \rangle \Longrightarrow \langle \mathbf{T}_1[\,\langle\!\langle c_1' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,], \sigma \rangle$. From (o) and (p),

we know (s) $\langle\!\langle c'_1 \rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,]) \rightsquigarrow_{\mathsf{t}} \langle\!\langle c_2 \rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])$. From (s) and (m), we have (t) $\mathbf{T}_1[\,\langle\!\langle c'_1 \rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,\langle\!\langle c_2 \rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,]$. From (t), (h), and (f), using the induction hypothesis, we have (u) $\langle \mathbf{T}_1[\,\langle\!\langle c'_1 \rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,], \sigma\rangle \Longrightarrow^{n-1} \kappa''_1$. Instantiating the goal with κ''_1, from (q), (r), (u), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\,\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} c_2\,]$ and (h) $\kappa = \langle \mathbf{T}_2[\,c_2\,], \sigma\rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$, c_1, and c'_1, such that (i) $T_1 = \mathbf{T}_1[\,\langle\!\langle c_1 \rangle\!\rangle_{\mathsf{a}} c'_1\,]$, (j) $c_1 \rightsquigarrow \mathbf{skip}$, (k) $c'_1 \rightsquigarrow c_2$, and (l) for all T'_1 and T'_2, if $T'_1 \rightsquigarrow_{\mathsf{t}} T'_2$, then $\mathbf{T}_1[\,T'_1\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,T'_2\,]$. From (j), by Def. 3.33, we have 2 cases:

  * We consider (m) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (m), and the semantics, we know (n) $\langle \mathbf{T}_1[\,\langle\!\langle c_1 \rangle\!\rangle_{\mathsf{a}} c'_1\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i), (n), and (b), we find a contradiction and conclude

  * We consider (m) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma\rangle$. From (m), and the semantics, we know (n) $\langle \mathbf{T}_1[\,\langle\!\langle c_1 \rangle\!\rangle_{\mathsf{a}} c'_1\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} c'_1\,], \sigma\rangle$. From the semantics, we know (o) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} c'_1\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,c'_1\,], \sigma\rangle$. From (k) and (l), we have (p) $\mathbf{T}_1[\,c'_1\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,c_2\,]$. From (p), (h), and (f), using the induction hypothesis, we have (q) $\langle \mathbf{T}_1[\,c'_1\,], \sigma\rangle \Longrightarrow^{n-1} \kappa''_1$. Instantiating the goal with κ''_1, from (n), (o), (q), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\,\langle\!\langle \mathbf{T}'_2[\,c'_2\,], \mathbf{T}''_2[\,c''_2\,] \rangle\!\rangle_{\mathsf{p}} c_2\,]$, (h) $\kappa = \mathsf{race}$, (i) $\langle c'_2, \sigma\rangle \xrightarrow[\delta_2]{}^* \langle c'''_2, \sigma'\rangle$, (j) $\langle c''_2, \sigma'\rangle \xrightarrow[\delta'_2]{}^* \kappa_2$, and (k) $\delta_2 \not\subset \delta'_2$. From (h), and the semantics, we know (f) is false and conclude

– We consider (g) $T_2 = \mathbf{T}_2[\,\langle\!\langle \mathbf{T}'_2[\,c'_2\,], \mathbf{T}''_2[\,c''_2\,] \rangle\!\rangle_{\mathsf{p}} c_2\,]$, (h) $\kappa = \mathsf{race}$, (i) $\langle c''_2, \sigma\rangle \xrightarrow[\delta_2]{}^* \langle c'''_2, \sigma'\rangle$, (j) $\langle c'_2, \sigma'\rangle \xrightarrow[\delta'_2]{}^* \kappa_2$, and (k) $\delta_2 \not\subset \delta'_2$. From (h), and the semantics, we know (f) is false and conclude

• From $\langle T_2, \sigma\rangle \Longrightarrow^\infty$, by Def. 5.3, we assume (c) $\langle T_2, \sigma\rangle \Longrightarrow^* \langle \mathbf{T}''_2[\,c''_2\,], \sigma''\rangle$ and (d) $\langle c''_2, \sigma''\rangle \Uparrow$, and will show there exists $\mathbf{T}''_1$ and c''_1 such that $\langle T_1, \sigma\rangle \Longrightarrow^* \langle \mathbf{T}''_1[\,c''_1\,], \sigma''\rangle$ and $\langle c''_1, \sigma''\rangle \Uparrow$. By induction over n, the number of steps of (c), we have 2 cases: If

$n = 0$, from (c) and the semantics, we have (e) $\langle T_2, \sigma\rangle = \langle \mathbf{T}_2''[\,c_2''\,], \sigma''\rangle$. From (a) and (e), by Def. 3.23, we know (f) $T_1 = \mathbf{T}_1''[\,c_1''\,]$ and (g) $c_1'' \rightsquigarrow c_2''$. From (g), by Def. 3.33, we have 2 cases:

- We consider (h) $\langle c_1'', \sigma\rangle \longrightarrow^* \textsf{abort}$. From (h), and the semantics, we know (i) $\langle \mathbf{T}_1''[\,c_1''\,], \sigma\rangle \Longrightarrow \textsf{abort}$. From (i), and (b), we find a contradiction and conclude
- We consider (h) $\langle c_1'', \sigma\rangle \Uparrow$. From (h), and the semantics, we know (i) $\langle T_1, \sigma\rangle \Longrightarrow^0 \langle T_1, \sigma\rangle$. Instantiating the goal with $\mathbf{T}_1''$ and c_1'', from (i) and (h), we conclude

If $n > 0$, from (c) and the semantics, we know there exists κ such that (e) $\langle T_2, \sigma\rangle \Longrightarrow \kappa$ and (f) $\kappa \Longrightarrow^{n-1} \langle \mathbf{T}_2''[\,c_2''\,], \sigma''\rangle$. From (e), and the semantics, we have 8 cases:

- We consider (g) $T_2 = \mathbf{T}_2[\,c_2\,]$, (h) $\kappa = \textsf{abort}$, and (i) $\langle c_2, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (h), and the semantics, we know (f) is false and conclude
- We consider (g) $T_2 = \mathbf{T}_2[\,c_2\,]$, (h) $\kappa = \langle \mathbf{T}_2[\,c_2'\,], \sigma'\rangle$, (i) $\langle c_2, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$, and (j) $c_2 \neq c_2'$. From (a) and (g), by Def. 3.23, we know there exists $\mathbf{T}_1$ and c_1 such that (j) $T_1 = \mathbf{T}_1[\,c_1\,]$, (k) $c_1 \rightsquigarrow c_2$, and (l) for all c_1' and c_2', if $c_1' \rightsquigarrow c_2'$, then $\mathbf{T}_1[\,c_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,c_2'\,]$. From (k) and (i), we have 2 cases:
  - ∗ We consider (m) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (m), and the semantics, we have (n) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow \textsf{abort}$. From (n) and (b), we find a contradiction and conclude
  - ∗ We consider (m) $\langle c_1, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and (n) $c_1' \rightsquigarrow c_2'$. From (n) and (l), we have (o) $\mathbf{T}_1[\,c_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,c_2'\,]$. From (o), (h), and (f), by the induction hypothesis, we have (p) $\langle \mathbf{T}_1[\,c_1'\,], \sigma'\rangle \Longrightarrow^* \langle \mathbf{T}_1''[\,c_1''\,], \sigma''\rangle$ and (q) $\langle c_1'', \sigma''\rangle \Uparrow$. We have then 2 cases:
    - · If (r) $c_1 = c_1'$, from (m), using Remark 3.32, we know (s) $\sigma = \sigma'$. Instantiating the goal with $\mathbf{T}_1''$ and c_1'', from (r), (s), (p), and (q), we conclude
    - · If (r) $c_1 \neq c_1'$, from (m) and the semantics, we have (s) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,c_1'\,], \sigma'\rangle$. Instantiating the goal with $\mathbf{T}_1''$ and c_1'', from (s), (p), (q),

and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\,\mathbf{S}_2[\,c_2 \parallel c'_2\,]\,]$ and (h) $\kappa = \langle \mathbf{T}_2[\,\langle\!\langle c_2, c'_2\rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,], \sigma\rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$ and c_1 such that (i) $T_1 = \mathbf{T}_1[\,c_1\,]$, (j) $c_1 \rightsquigarrow \mathbf{S}_2[\,c_2 \parallel c'_2\,]$, and (k) for all T'_1 and T'_2, if $T'_1 \rightsquigarrow_{\mathbf{t}} T'_2$, then $\mathbf{T}_1[\,T'_1\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,T'_2\,]$. From (j), by Def. 3.33, we have 2 cases:

  * We consider (l) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (l), and the semantics, we know (m) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i), (m), and (b), we find a contradiction and conclude

  * We consider (l) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{S}_1[\,c'_1 \parallel c''_1\,], \sigma\rangle$, (m) $c'_1 \rightsquigarrow c_2$, (n) $c''_1 \rightsquigarrow c'_2$, and (o) $\mathbf{S}_1[\,\mathbf{skip}\,] \rightsquigarrow \mathbf{S}_2[\,\mathbf{skip}\,]$. From (l), and the semantics, we know (p) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\mathbf{S}_1[\,c'_1 \parallel c''_1\,]\,], \sigma\rangle$. From the semantics, we know (q) $\langle \mathbf{T}_1[\,\mathbf{S}_1[\,c'_1 \parallel c''_1\,]\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,\langle\!\langle c'_1, c''_1\rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,], \sigma\rangle$. From (m), (n), and (o), we know (r) $\langle\!\langle c'_1, c''_1\rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_1[\,\mathbf{skip}\,]) \rightsquigarrow_{\mathbf{t}} \langle\!\langle c_2, c'_2\rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_2[\,\mathbf{skip}\,])$. From (r) and (k), we have (s) $\mathbf{T}_1[\,\langle\!\langle c'_1, c''_1\rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,\langle\!\langle c_2, c'_2\rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,]$. From (s), (h), and (f), using the induction hypothesis, we have (t) $\langle \mathbf{T}_1[\,\langle\!\langle c'_1, c''_1\rangle\!\rangle_{\mathbf{p}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}''_1[\,c''_1\,], \sigma''\rangle$ and (u) $\langle c''_1, \sigma''\rangle \Uparrow$. Instantiating the goal with $\mathbf{T}''_1$ and c''_1, from (p), (q), (t), (u), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\,\langle\!\langle \mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathbf{p}} c_2\,]$ and (h) $\kappa = \langle \mathbf{T}_2[\,c_2\,], \sigma\rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$, c_1, c'_1, and c''_1, such that (i) $T_1 = \mathbf{T}_1[\,\langle\!\langle c_1, c'_1\rangle\!\rangle_{\mathbf{p}} c''_1\,]$, (j) $c_1 \rightsquigarrow \mathbf{skip}$, (k) $c'_1 \rightsquigarrow \mathbf{skip}$, (l) $c''_1 \rightsquigarrow c_2$, and (m) for all T'_1 and T'_2, if $T'_1 \rightsquigarrow_{\mathbf{t}} T'_2$, then $\mathbf{T}_1[\,T'_1\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,T'_2\,]$. From (j), by Def. 3.33, we have 2 cases:

  * We consider (n) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\,\langle\!\langle c_1, c'_1\rangle\!\rangle_{\mathbf{p}} c''_1\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i), (o), and (b), we find a contradiction and conclude

  * We consider (n) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma\rangle$. From (n), and the semantics, we know

(o) $\langle \mathbf{T}_1[\,\langle\!\langle c_1, c_1' \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma \rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, c_1' \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma \rangle$. From (k), by Def. 3.33, we have 2 cases:

- We consider (p) $\langle c_1', \sigma \rangle \longrightarrow^* \mathsf{abort}$. From (p), and the semantics, we know (q) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, c_1' \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma \rangle \Longrightarrow \mathsf{abort}$. From (i), (o), (q), and (b), we find a contradiction and conclude
- We consider (p) $\langle c_1', \sigma \rangle \Downarrow \langle \mathbf{skip}, \sigma \rangle$. From (p), and the semantics, we know (q) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, c_1' \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma \rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathbf{p}} c_1''\,], \sigma \rangle$. From the semantics, we know (r) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathbf{p}} c_1'\,], \sigma \rangle \Longrightarrow \langle \mathbf{T}_1[\,c_1'\,], \sigma \rangle$. From (l) and (m), we have (s) $\mathbf{T}_1[\,c_1'\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,c_2\,]$. From (s), (h), and (f), using the induction hypothesis, we have (t) $\langle \mathbf{T}_1[\,c_1'\,], \sigma \rangle \Longrightarrow^* \langle \mathbf{T}_1''[\,c_1''\,], \sigma'' \rangle$ and (u) $\langle c_1'', \sigma'' \rangle \Uparrow$. Instantiating the goal with $\mathbf{T}_1''$ and c_1'', from (o), (q), (r), (t), (u), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\,\mathbf{S}_2[\,\mathbf{atomic}\; c_2\,]\,]$, (h) $\kappa = \langle \mathbf{T}_2[\,\langle\!\langle c_2 \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,], \sigma \rangle$, and (i) $\mathbf{T}_2$ is 0-atomic. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$ and c_1 such that (j) $T_1 = \mathbf{T}_1[\,c_1\,]$, (k) $c_1 \rightsquigarrow \mathbf{S}_2[\,\mathbf{atomic}\; c_2\,]$, (l) $\mathbf{T}_1$ is 0-atomic, and (m) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathbf{t}} T_2'$, then $\mathbf{T}_1[\,T_1'\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,T_2'\,]$. From (k), by Def. 3.33, we have 2 cases:

  * We consider (n) $\langle c_1, \sigma \rangle \longrightarrow^* \mathsf{abort}$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\,c_1\,], \sigma \rangle \Longrightarrow \mathsf{abort}$. From (j), (o), and (b), we find a contradiction and conclude
  * We consider (n) $\langle c_1, \sigma \rangle \Downarrow \langle \mathbf{S}_1[\,\mathbf{atomic}\; c_1'\,], \sigma \rangle$, (o) $c_1' \rightsquigarrow c_2$, and (p) $\mathbf{S}_1[\,\mathbf{skip}\,] \rightsquigarrow \mathbf{S}_2[\,\mathbf{skip}\,]$. From (n), and the semantics, we know (q) $\langle \mathbf{T}_1[\,c_1\,], \sigma \rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\mathbf{S}_1[\,\mathbf{atomic}\; c_1'\,]\,], \sigma \rangle$. From (l) and the semantics, we know (r) $\langle \mathbf{T}_1[\,\mathbf{S}_1[\,\mathbf{atomic}\; c_1'\,]\,], \sigma \rangle \Longrightarrow \langle \mathbf{T}_1[\,\langle\!\langle c_1' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,], \sigma \rangle$. From (o) and (p), we know (s) $\langle\!\langle c_1' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,]) \rightsquigarrow_{\mathbf{t}} \langle\!\langle c_2 \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])$. From (s) and (m), we have (t) $\mathbf{T}_1[\,\langle\!\langle c_1' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,\langle\!\langle c_2 \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,]$. From (t), (h), and

(f), using the induction hypothesis, we have (u) $\langle \mathbf{T}_1[\langle\!\langle c'_1 \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}''_1[\,c''_1\,], \sigma''\rangle$ and (v) $\langle c''_1, \sigma''\rangle \Uparrow$. Instantiating the goal with $\mathbf{T}''_1$ and c''_1, from (q), (r), (u), (v), and the semantics, we conclude

- We consider (g) $T_2 = \mathbf{T}_2[\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathbf{a}} c_2\,]$ and (h) $\kappa = \langle \mathbf{T}_2[\,c_2\,], \sigma\rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$, c_1, and c'_1, such that (i) $T_1 = \mathbf{T}_1[\langle\!\langle c_1 \rangle\!\rangle_{\mathbf{a}} c'_1\,]$, (j) $c_1 \rightsquigarrow \mathbf{skip}$, (k) $c'_1 \rightsquigarrow c_2$, and (l) for all T'_1 and T'_2, if $T'_1 \rightsquigarrow_{\mathbf{t}} T'_2$, then $\mathbf{T}_1[\,T'_1\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,T'_2\,]$. From (j), by Def. 3.33, we have 2 cases:
  - We consider (m) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (m), and the semantics, we know (n) $\langle \mathbf{T}_1[\langle\!\langle c_1 \rangle\!\rangle_{\mathbf{a}} c'_1\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i), (n), and (b), we find a contradiction and conclude
  - We consider (m) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma\rangle$. From (m), and the semantics, we know (n) $\langle \mathbf{T}_1[\langle\!\langle c_1 \rangle\!\rangle_{\mathbf{a}} c'_1\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathbf{a}} c'_1\,], \sigma\rangle$. From the semantics, we know (o) $\langle \mathbf{T}_1[\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathbf{a}} c'_1\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,c'_1\,], \sigma\rangle$. From (k) and (l), we have (p) $\mathbf{T}_1[\,c'_1\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,c_2\,]$. From (p), (h), and (f), using the induction hypothesis, we have (q) $\langle \mathbf{T}_1[\,c'_1\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}''_1[\,c''_1\,], \sigma''\rangle$ and (r) $c''_1 \Uparrow \sigma''$. Instantiating the goal with $\mathbf{T}''_1$ and c''_1, from (n), (o), (q), (r), and the semantics, we conclude
- We consider (g) $T_2 = \mathbf{T}_2[\langle\!\langle \mathbf{T}'_2[\,c'_2\,], \mathbf{T}''_2[\,c''_2\,] \rangle\!\rangle_{\mathbf{p}} c_2\,]$, (h) $\kappa = \mathsf{race}$, (i) $\langle c'_2, \sigma\rangle \xrightarrow[\delta_2]{}^* \langle c'''_2, \sigma'\rangle$, (j) $\langle c''_2, \sigma'\rangle \xrightarrow[\delta'_2]{}^* \kappa_2$, and (k) $\delta_2 \not\smile \delta'_2$. From (h), and the semantics, we know (f) is false and conclude
- We consider (g) $T_2 = \mathbf{T}_2[\langle\!\langle \mathbf{T}'_2[\,c'_2\,], \mathbf{T}''_2[\,c''_2\,] \rangle\!\rangle_{\mathbf{p}} c_2\,]$, (h) $\kappa = \mathsf{race}$, (i) $\langle c''_2, \sigma\rangle \xrightarrow[\delta_2]{}^* \langle c'''_2, \sigma'\rangle$, (j) $\langle c'_2, \sigma'\rangle \xrightarrow[\delta'_2]{}^* \kappa_2$, and (k) $\delta_2 \not\smile \delta'_2$. From (h), and the semantics, we know (f) is false and conclude

- We assume (c) $\langle T_2, \sigma\rangle \Longrightarrow^* \mathsf{race}$. By induction over n, the number of steps of (c), we have 2 cases: If (d) $n = 0$, from (c) and the semantics, we have (e) $\langle T_2, \sigma\rangle = \mathsf{race}$. We know (e) is false and conclude; If (d) $n > 0$, from (c) and the semantics, we know there exists κ such that (e) $\langle T_2, \sigma\rangle \Longrightarrow \kappa$ and (f) $\kappa \Longrightarrow^{n-1}$ race. From (e), and the semantics, we have 8 cases:

- We consider (g) $T_2 = \mathbf{T}_2[\,c_2\,]$, (h) $\kappa = \textsf{abort}$, and (i) $\langle c_2, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (h), and the semantics, we know (f) is false and conclude
- We consider (g) $T_2 = \mathbf{T}_2[\,c_2\,]$, (h) $\kappa = \langle \mathbf{T}_2[\,c_2'\,], \sigma'\rangle$, (i) $\langle c_2, \sigma\rangle \Downarrow \langle c_2', \sigma'\rangle$, and (j) $c_2 \neq c_2'$. From (a) and (g), by Def. 3.23, we know there exists $\mathbf{T}_1$ and c_1 such that (j) $T_1 = \mathbf{T}_1[\,c_1\,]$, (k) $c_1 \rightsquigarrow c_2$, and (l) for all c_1' and c_2', if $c_1' \rightsquigarrow c_2'$, then $\mathbf{T}_1[\,c_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,c_2'\,]$. From (k) and (i), we have 2 cases:
  - We consider (m) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (m), and the semantics, we have (n) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow \textsf{abort}$. From (n) and (b), we find a contradiction and conclude
  - We consider (m) $\langle c_1, \sigma\rangle \Downarrow \langle c_1', \sigma'\rangle$ and (n) $c_1' \rightsquigarrow c_2'$. From (n) and (l), we have (o) $\mathbf{T}_1[\,c_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,c_2'\,]$. From (o), (h), and (f), by the induction hypothesis, we have (p) $\langle \mathbf{T}_1[\,c_1'\,], \sigma'\rangle \Longrightarrow^* \textsf{race}$. We have then 2 cases:
    - If (q) $c_1 = c_1'$, from (m), using Remark 3.32, we know (r) $\sigma = \sigma'$. From (q), (r), and (p), we conclude
    - If (q) $c_1 \neq c_1'$, from (m) and the semantics, we have (r) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,c_1'\,], \sigma'\rangle$. From (r), (p), and the semantics, we conclude
- We consider (g) $T_2 = \mathbf{T}_2[\,\mathbf{S}_2[\,c_2 \parallel c_2'\,]\,]$ and (h) $\kappa = \langle \mathbf{T}_2[\,\langle\!\langle c_2, c_2'\rangle\!\rangle_{\mathsf{p}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,], \sigma\rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$ and c_1 such that (i) $T_1 = \mathbf{T}_1[\,c_1\,]$, (j) $c_1 \rightsquigarrow \mathbf{S}_2[\,c_2 \parallel c_2'\,]$, and (k) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathsf{t}} T_2'$, then $\mathbf{T}_1[\,T_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,T_2'\,]$. From (j), by Def. 3.33, we have 2 cases:
  - We consider (l) $\langle c_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (l), and the semantics, we know (m) $\langle \mathbf{T}_1[\,c_1\,]\sigma, \rangle \Longrightarrow \textsf{abort}$. From (i), (m), and (b), we find a contradiction and conclude
  - We consider (l) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{S}_1[\,c_1' \parallel c_1''\,], \sigma\rangle$, (m) $c_1' \rightsquigarrow c_2$, (n) $c_1'' \rightsquigarrow c_2'$, and (o) $\mathbf{S}_1[\,\mathbf{skip}\,] \rightsquigarrow \mathbf{S}_2[\,\mathbf{skip}\,]$. From (l), and the semantics, we know

(p) $\langle \mathbf{T}_1[\, c_1 \,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\, \mathbf{S}_1[\, c_1' \parallel c_1'' \,]\,], \sigma\rangle$. From the semantics, we know (q) $\langle \mathbf{T}_1[\, \mathbf{S}_1[\, c_1' \parallel c_1'' \,]\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\, \langle\!\langle c_1', c_1''\rangle\!\rangle_{\mathrm{p}}(\mathbf{S}_1[\, \mathbf{skip} \,])\,], \sigma\rangle$. From (m), (n), and (o), we know (r) $\langle\!\langle c_1', c_1''\rangle\!\rangle_{\mathrm{p}}(\mathbf{S}_1[\, \mathbf{skip} \,]) \rightsquigarrow_{\mathrm{t}} \langle\!\langle c_2, c_2'\rangle\!\rangle_{\mathrm{p}}(\mathbf{S}_2[\, \mathbf{skip} \,])$. From (r) and (k), we have (s) $\mathbf{T}_1[\, \langle\!\langle c_1', c_1''\rangle\!\rangle_{\mathrm{p}}(\mathbf{S}_1[\, \mathbf{skip} \,])\,] \rightsquigarrow_{\mathrm{t}} \mathbf{T}_2[\, \langle\!\langle c_2, c_2'\rangle\!\rangle_{\mathrm{p}}(\mathbf{S}_2[\, \mathbf{skip} \,])\,]$. From (s), (h), and (f), using the induction hypothesis, we have (t) $\langle \mathbf{T}_1[\, \langle\!\langle c_1', c_1''\rangle\!\rangle_{\mathrm{p}}(\mathbf{S}_1[\, \mathbf{skip} \,])\,], \sigma\rangle \Longrightarrow^* \mathsf{race}$. From (p), (q), (t), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\, \langle\!\langle \mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathrm{p}} c_2 \,]$ and (h) $\kappa = \langle \mathbf{T}_2[\, c_2 \,], \sigma\rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$, c_1, c_1', and c_1'', such that (i) $T_1 = \mathbf{T}_1[\, \langle\!\langle c_1, c_1'\rangle\!\rangle_{\mathrm{p}} c_1'' \,]$, (j) $c_1 \rightsquigarrow \mathbf{skip}$, (k) $c_1' \rightsquigarrow \mathbf{skip}$, (l) $c_1'' \rightsquigarrow c_2$, and (m) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathrm{t}} T_2'$, then $\mathbf{T}_1[\, T_1' \,] \rightsquigarrow_{\mathrm{t}} \mathbf{T}_2[\, T_2' \,]$. From (j), by Def. 3.33, we have 2 cases:

  * We consider (n) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\, \langle\!\langle c_1, c_1'\rangle\!\rangle_{\mathrm{p}} c_1'' \,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i), (o), and (b), we find a contradiction and conclude

  * We consider (n) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma\rangle$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\, \langle\!\langle c_1, c_1'\rangle\!\rangle_{\mathrm{p}} c_1'' \,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\, \langle\!\langle \mathbf{skip}, c_1'\rangle\!\rangle_{\mathrm{p}} c_1'' \,], \sigma\rangle$. From (k), by Def. 3.33, we have 2 cases:

    · We consider (p) $\langle c_1', \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (p), and the semantics, we know (q) $\langle \mathbf{T}_1[\, \langle\!\langle \mathbf{skip}, c_1'\rangle\!\rangle_{\mathrm{p}} c_1'' \,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i), (o), (q), and (b), we find a contradiction and conclude

    · We consider (p) $\langle c_1', \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma\rangle$. From (p), and the semantics, we know (q) $\langle \mathbf{T}_1[\, \langle\!\langle \mathbf{skip}, c_1'\rangle\!\rangle_{\mathrm{p}} c_1'' \,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\, \langle\!\langle \mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathrm{p}} c_1'' \,], \sigma\rangle$. From the semantics, we know (r) $\langle \mathbf{T}_1[\, \langle\!\langle \mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathrm{p}} c_1' \,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\, c_1' \,], \sigma\rangle$. From (l) and (m), we have (s) $\mathbf{T}_1[\, c_1' \,] \rightsquigarrow_{\mathrm{t}} \mathbf{T}_2[\, c_2 \,]$. From (s), (h), and (f), using the induction hypothesis, we have (t) $\langle \mathbf{T}_1[\, c_1' \,], \sigma\rangle \Longrightarrow^* \mathsf{race}$. From (o), (q), (r), (t), and the semantics, we conclude

- We consider (g) $T_2 = \mathbf{T}_2[\,\mathbf{S}_2[\,\mathbf{atomic}\; c_2\,]\,]$, (h) $\kappa = \langle \mathbf{T}_2[\,\langle\!\langle c_2 \rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,], \sigma\rangle$, and (i) $\mathbf{T}_2$ is 0-atomic. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$ and c_1 such that (j) $T_1 = \mathbf{T}_1[\,c_1\,]$, (k) $c_1 \rightsquigarrow \mathbf{S}_2[\,\mathbf{atomic}\; c_2\,]$, (l) $\mathbf{T}_1$ is 0-atomic, and (m) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathsf{t}} T_2'$, then $\mathbf{T}_1[\,T_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,T_2'\,]$. From (k), by Def. 3.33, we have 2 cases:
  * We consider (n) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (n), and the semantics, we know (o) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (j), (o), and (b), we find a contradiction and conclude
  * We consider (n) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{S}_1[\,\mathbf{atomic}\; c_1'\,], \sigma\rangle$, (o) $c_1' \rightsquigarrow c_2$, and (p) $\mathbf{S}_1[\,\mathbf{skip}\,] \rightsquigarrow \mathbf{S}_2[\,\mathbf{skip}\,]$. From (n), and the semantics, we know (q) $\langle \mathbf{T}_1[\,c_1\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\mathbf{S}_1[\,\mathbf{atomic}\; c_1'\,]\,], \sigma\rangle$. From (l) and the semantics, we know (r) $\langle \mathbf{T}_1[\,\mathbf{S}_1[\,\mathbf{atomic}\; c_1'\,]\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,\langle\!\langle c_1' \rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,], \sigma\rangle$. From (o) and (p), we know (s) $\langle\!\langle c_1' \rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,]) \rightsquigarrow_{\mathsf{t}} \langle\!\langle c_2 \rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])$. From (s) and (m), we have (t) $\mathbf{T}_1[\,\langle\!\langle c_1' \rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,\langle\!\langle c_2 \rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_2[\,\mathbf{skip}\,])\,]$. From (t), (h), and (f), using the induction hypothesis, we have (u) $\langle \mathbf{T}_1[\,\langle\!\langle c_1' \rangle\!\rangle_{\mathsf{a}}(\mathbf{S}_1[\,\mathbf{skip}\,])\,], \sigma\rangle \Longrightarrow^* \mathsf{race}$. From (q), (r), (u), and the semantics, we conclude
- We consider (g) $T_2 = \mathbf{T}_2[\,\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} c_2\,]$ and (h) $\kappa = \langle \mathbf{T}_2[\,c_2\,], \sigma\rangle$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$, c_1, and c_1', such that (i) $T_1 = \mathbf{T}_1[\,\langle\!\langle c_1 \rangle\!\rangle_{\mathsf{a}} c_1'\,]$, (j) $c_1 \rightsquigarrow \mathbf{skip}$, (k) $c_1' \rightsquigarrow c_2$, and (l) for all T_1' and T_2', if $T_1' \rightsquigarrow_{\mathsf{t}} T_2'$, then $\mathbf{T}_1[\,T_1'\,] \rightsquigarrow_{\mathsf{t}} \mathbf{T}_2[\,T_2'\,]$. From (j), by Def. 3.33, we have 2 cases:
  * We consider (m) $\langle c_1, \sigma\rangle \longrightarrow^* \mathsf{abort}$. From (m), and the semantics, we know (n) $\langle \mathbf{T}_1[\,\langle\!\langle c_1 \rangle\!\rangle_{\mathsf{a}} c_1'\,], \sigma\rangle \Longrightarrow \mathsf{abort}$. From (i), (n), and (b), we find a contradiction and conclude
  * We consider (m) $\langle c_1, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma\rangle$. From (m), and the semantics, we know (n) $\langle \mathbf{T}_1[\,\langle\!\langle c_1 \rangle\!\rangle_{\mathsf{a}} c_1'\,], \sigma\rangle \Longrightarrow^* \langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} c_1'\,], \sigma\rangle$. From the semantics, we know (o) $\langle \mathbf{T}_1[\,\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} c_1'\,], \sigma\rangle \Longrightarrow \langle \mathbf{T}_1[\,c_1'\,], \sigma\rangle$. From (k) and (l), we have

(p) $\mathbf{T}_1[\,c'_1\,] \rightsquigarrow_{\mathbf{t}} \mathbf{T}_2[\,c_2\,]$. From (p), (h), and (f), using the induction hypothesis, we have (q) $\langle \mathbf{T}_1[\,c'_1\,], \sigma\rangle \Longrightarrow^* \textsf{race}$. From (n), (o), (q), and the semantics, we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\langle\!\langle \mathbf{T}'_2[\,c'_2\,], \mathbf{T}''_2[\,c''_2\,]\rangle\!\rangle_{\mathbf{p}} c_2\,]$, (h) $\kappa = \textsf{race}$, (i) $\langle c'_2, \sigma\rangle \xrightarrow[\delta_2]{}{}^* \langle c'''_2, \sigma'\rangle$, (j) $\langle c''_2, \sigma'\rangle \xrightarrow[\delta'_2]{}{}^* \kappa_2$, and (k) $\delta_2 \not\overset{\leftharpoonup}{\smile} \delta'_2$. From (a) and (g), by Def. 3.23 and Def. 3.33, we know there exists $\mathbf{T}_1$, $\mathbf{T}'_1$, c'_1, $\mathbf{T}''_1$, c''_1, and c_1, such that (l) $T_1 = \mathbf{T}_1[\langle\!\langle \mathbf{T}'_1[\,c'_1\,], \mathbf{T}''_1[\,c''_1\,]\rangle\!\rangle_{\mathbf{p}} c_1\,]$, (m) $c'_1 \rightsquigarrow c'_2$, and (n) $c''_1 \rightsquigarrow c''_2$. From (i) and (m), by Def. 3.33, we have 2 cases:

  ∗ We consider (o) $\langle c'_1, \sigma\rangle \longrightarrow^* \textsf{abort}$. From (o), and the semantics, we know (p) $\langle \mathbf{T}_1[\langle\!\langle \mathbf{T}'_1[\,c'_1\,], \mathbf{T}''_1[\,c''_1\,]\rangle\!\rangle_{\mathbf{p}} c_1\,], \sigma\rangle \Longrightarrow \textsf{abort}$. From (l), (p), and (b), we find a contradiction and conclude

  ∗ We consider (o) $\langle c'_1, \sigma\rangle \xmapsto[\delta_1]{}{}^* \langle c'''_1, \sigma''\rangle$ and (p) $\delta_2 \subseteq \delta_1$. From (j), (k), (p), and the semantics we know there exists δ''_2 and κ'_2 such that (q) $\langle c''_2, \sigma''\rangle \xrightarrow[\delta''_2]{}{}^* \kappa'_2$ and (r) $\delta_1 \not\overset{\rightharpoonup}{\smile} \delta''_2$. From (q) and (n), by Def. 3.33, we have 2 cases:

    · We consider (s) $\langle c''_1, \sigma''\rangle \xrightarrow[\delta'_1]{}{}^* \textsf{abort}$ and (t) $\delta_1 \overset{\rightharpoonup}{\smile} \delta'_1$. From (t), (o), and (s), using Remark 5.6, we know (u) $\langle c''_1, \sigma\rangle \longrightarrow^* \textsf{abort}$ From (u), and the semantics, we know (v) $\langle \mathbf{T}_1[\langle\!\langle \mathbf{T}'_1[\,c'_1\,], \mathbf{T}''_1[\,c''_1\,]\rangle\!\rangle_{\mathbf{p}} c_1\,], \sigma\rangle \Longrightarrow \textsf{abort}$. From (l), (v), and (b), we find a contradiction and conclude

    · We consider (s) $\langle c''_1, \sigma''\rangle \xmapsto[\delta'_1]{}{}^* \kappa_1$ and (t) $\delta''_2 \subseteq \delta'_1$. From (t) and (r), we know (u) $\delta_1 \not\overset{\rightharpoonup}{\smile} \delta'_1$. From (o), (s), (u), and the semantics, we know (v) $\langle \mathbf{T}_1[\langle\!\langle \mathbf{T}'_1[\,c'_1\,], \mathbf{T}''_1[\,c''_1\,]\rangle\!\rangle_{\mathbf{p}} c_1\,], \sigma\rangle \Longrightarrow \textsf{race}$. From (l) and (v), we conclude

– We consider (g) $T_2 = \mathbf{T}_2[\langle\!\langle \mathbf{T}'_2[\,c'_2\,], \mathbf{T}''_2[\,c''_2\,]\rangle\!\rangle_{\mathbf{p}} c_2\,]$, (h) $\kappa = \textsf{race}$, (i) $\langle c''_2, \sigma\rangle \xrightarrow[\delta_2]{}{}^* \langle c'''_2, \sigma'\rangle$, (j) $\langle c'_2, \sigma'\rangle \xrightarrow[\delta'_2]{}{}^* \kappa_2$, and (k) $\delta_2 \not\overset{\rightharpoonup}{\smile} \delta'_2$. Proof is symmetric to the previous case □

And, finally, we can combine Lemma 5.10 with Lemmas 5.5 and 5.4, to obtain the following corollary:

Corollary 5.11. If $T_1 \rightsquigarrow_{\mathsf{t}} T_2$, $\neg\langle T_1, \sigma\rangle \longmapsto^* \mathsf{abort}$, and $\neg\langle T_1, \sigma\rangle \longmapsto^* \mathsf{race}$, then

1. $\neg\langle T_2, \sigma\rangle \longmapsto^* \mathsf{race}$
2. $\neg\langle T_2, \sigma\rangle \longmapsto^* \mathsf{abort}$
3. If $\langle T_2, \sigma\rangle \longmapsto^* \langle \mathbf{skip}, \sigma'\rangle$, then $\langle T_1, \sigma\rangle \longmapsto^* \langle \mathbf{skip}, \sigma'\rangle$
4. If $\langle T_2, \sigma\rangle \longmapsto^\infty$, then $\langle T_1, \sigma\rangle \longmapsto^\infty$

Proof. We assume (a) $T_1 \rightsquigarrow_{\mathsf{t}} T_2$, (b) $\langle T_1, \sigma\rangle \longmapsto^* \mathsf{abort}$ is false, and (c) $\langle T_1, \sigma\rangle \longmapsto^* \mathsf{race}$ is false. We know (d) $\langle T_1, \sigma\rangle \Longrightarrow^* \mathsf{abort}$ is false

- Let us assume (e) $\langle T_1, \sigma\rangle \Longrightarrow^* \mathsf{abort}$ is true. From (e), using Lemma 5.4 (item 1), we have (d) $\langle T_1, \sigma\rangle \longmapsto^* \mathsf{abort}$. From (b) and (d), we find a contradiction and conclude

We know (e) $\langle T_2, \sigma\rangle \longmapsto^* \mathsf{race}$ is false

- Let us assume (f) $\langle T_2, \sigma\rangle \longmapsto^* \mathsf{race}$ is true. From (f), using Lemma 5.5 (item 1), we have (g) $\langle T_2, \sigma\rangle \Longrightarrow^* \mathsf{race}$. From (a), (d), and (g), using Lemma 5.10 (item 4), we have (h) $\langle T_1, \sigma\rangle \Longrightarrow^* \mathsf{race}$. From (h), using Lemma 5.4 (item 1), we have (i) $\langle T_1, \sigma\rangle \longmapsto^* \mathsf{race}$. From (c) and (i), we find a contradiction conclude

Then we consider the 4 cases:

- From (e), we conclude
- Let us assume (f) $\langle T_2, \sigma\rangle \longmapsto^* \mathsf{abort}$ is true. From (a), and (d), using Lemma 5.10 (item 1), we know (g) $\langle T_2, \sigma\rangle \Longrightarrow^* \mathsf{abort}$ is false. From (e), and (f), using Lemma 5.5 (item 2.a), we know (h) $\langle T_2, \sigma\rangle \Longrightarrow^* \mathsf{abort}$. From (g) and (h), we find a contradiction and conclude
- If (f) $\langle T_2, \sigma\rangle \longmapsto^* \langle \mathbf{skip}, \sigma'\rangle$, then we need to show that $\langle T_1, \sigma\rangle \longmapsto^* \langle \mathbf{skip}, \sigma'\rangle$. From (e) and (f), using Lemma 5.5 (item 2.b), we have (g) $\langle T_2, \sigma\rangle \Longrightarrow^* \langle \mathbf{skip}, \sigma'\rangle$. From (a), (d) and (g), using Lemma 5.10 (item 2), we know (h) $\langle T_1, \sigma\rangle \Longrightarrow^* \langle \mathbf{skip}, \sigma'\rangle$. From (h), using Lemma 5.4 (item 1), we conclude

- If (f) $\langle T_2, \sigma\rangle \longmapsto^{\infty}$, then we need to show that $\langle T_1, \sigma\rangle \longmapsto^{\infty}$. From (e) and (f), using Lemma 5.5 (item 2.c), we have (g) $\langle T_2, \sigma\rangle \Longrightarrow^{\infty}$. From (a), (d) and (g), using Lemma 5.10 (item 3), we know (h) $\langle T_1, \sigma\rangle \Longrightarrow^{\infty}$. From (h), using Lemma 5.4 (item 2), we conclude $\square$

5.3 Proof of the DRF-guarantee

We can now prove the DRF-guarantee of the relaxed $\rightsquigarrow$-parameterized semantics presented in Sec. 3.10. Based on Def.5.2, Theorem 5.12 says that a race-free program configuration has the same behavior in the relaxed semantics as it has in the interleaved semantics:

1. if it is safe in the interleaved semantics, then it is safe in the relaxed semantics;
2. if it is safe in the interleaved semantics, then it will reach a final state through the relaxed semantics that is reachable through the interleaved semantics.

Theorem 5.12 (DRF-guarantee)**.** The $\rightsquigarrow$ relation provides the DRF-guarantee

Proof. From Def. 5.2, given (a) $\kappa \longmapsto^{*} \textsf{abort}$ is false, (b) $\kappa \longmapsto^{*} \textsf{race}$ is false, we have 2 cases:

- Let us assume (c) $[\Lambda]\ \kappa \longmapsto^{*} \textsf{abort}$ is true. From (a) and (b), and the semantics, we know $\kappa = \langle T, \sigma\rangle$. From (c), using Lemma 3.46, we know exists T' such that (d) $T \rightsquigarrow_{\mathsf{t}} T'$ and (e) $\langle T', \sigma\rangle \longmapsto^{*} \textsf{abort}$. From (d), (a), and (b), using Corollary 5.11 (item 2), we know (f) $\langle T', \sigma\rangle \longmapsto^{*} \textsf{abort}$ is false. From (e) and (f), we find a contradiction and conclude
- If (c) $[\Lambda]\ \kappa \longmapsto^{*} \langle \mathbf{skip}, \sigma\rangle$, then we need to show that $\kappa \longmapsto^{*} \langle \mathbf{skip}, \sigma\rangle$. From (a) and (b), and the semantics, we know $\kappa = \langle T, \sigma'\rangle$. From (c), using Lemma 3.46, we know exists T' such that (d) $T \rightsquigarrow_{\mathsf{t}} T'$ and (e) $\langle T', \sigma'\rangle \longmapsto^{*} \langle \mathbf{skip}, \sigma\rangle$. From (d), (a), (b), and (e), using Corollary 5.11 (item 3), we conclude $\square$

Chapter 6

Concurrent Separation Logic

In this chapter, we present a logic for concurrent program verification similar to Concurrent Separation Logic (CSL). We prove its soundness with regard to the interleaved semantics of Sec. 3.7. Then, using the DRF-guarantee results of Chapter 5, we establish its soundness with regard to the $\rightsquigarrow$-parameterized relaxed semantics of Sec. 3.10.

CSL keeps an invariant about the shared memory. When no thread is accessing the shared memory, it is guaranteed that the invariant holds. When a thread is accessing the shared memory — which the logic enforces by mutual exclusion — the invariant does not need to be preserved, but it has to be reinstated once the operation ceases. An interesting aspect about CSL is that it allows ownership transfer to happen during shared memory access. When a thread starts a shared memory access, it gets a hold on the complete shared memory. When it finishes, it does not necessarily need to release the shared memory with the exact footprint. It may be increased or decreased. That is the ownership transfer that happens. This is because CSL only requires the shared-memory invariant to be preserved, and memories with different sizes may very well satisfy the requirement. For instance, if we have a list in shared memory, we may have an invariant that describes precisely the structure of the list, but does not specify its length. Therefore during shared-memory access any thread could insert or remove elements in the list, as long as the list structure is preserved. This ownership transfer of list nodes is dynamic, during program execution, even though the invariant that describes the list structure is static.

There are two important aspects of our semantics that needed special treatment when developing this logic and its soundness:

1. The dynamic semantics of the language execute atomic blocks in small steps. Therefore shared memory modification is not performed in a single-shot, and it can interleave with non-atomic code;

2. Memory allocation and deallocation have to be protected by an atomic block in order to establish race freedom. This is enforced, syntactically, by the inference rules.

Note also that standard CSL, such as the one presented here, does not support sharing read-only memory, that will be addressed in Chapter 7.

6.1 Assertion Language

We define assertions as sets of states as show in Fig. 6.1. There is no special syntax for writing them, we phrase them using set operators borrowed from the meta-logic (e.g. $\cap$ for conjunction, $\cup$ for disjunction, $\subseteq$ for implication, an so forth). We also provide support for assertion formulae, to explicitly characterize assertions that have a free metavariable of polymorphic type α.

$$\begin{array}{rrcl} (\mathit{Assertion}) & P, Q, I & \subseteq & \mathit{State} \\ (\mathit{AssertionFormula}) & \mathcal{P}, \mathcal{Q}, \mathcal{I} & \subseteq & \alpha \to \mathit{Assertion} \end{array}$$

Figure 6.1: Assertions and assertion formulae

Common operations on assertions not captured by the standard set operators are defined by a shallow embedding in the meta-logic. They are presented in Fig. 6.2.

To lift conditions into sets of states we use the notation $\lfloor b \rfloor$, it defines the set of states where b evaluates to true. To get the set of states where b evaluates to false, we usually write $\lfloor \neg b \rfloor$. Note that this is very different from $\neg \lfloor b \rfloor$, which gets the set of states where b

$$
\begin{aligned}
\lfloor b \rfloor &\overset{\text{def}}{=} \{\sigma \mid [\![b]\!]_\sigma = \mathit{true}\} \\
P_1 {*} P_2 &\overset{\text{def}}{=} \{\sigma_1 \uplus \sigma_2 \mid \sigma_1 \in P_1 \wedge \sigma_2 \in P_2\} \\
\mathsf{Emp} &\overset{\text{def}}{=} \{\varnothing\} \\
\ell \mapsto i &\overset{\text{def}}{=} \{\{\ell \rightsquigarrow i\}\}
\end{aligned}
$$

Figure 6.2: Auxiliary assertion definitions

evaluates to false or it is undefined. Therefore to get the set of states where b evaluation is defined we usually write $\lfloor b \rfloor \cup \lfloor \neg b \rfloor$.

In Fig. 6.2, we also define the separating conjunction, the $*$ from standard separation logic. $P_1 * P_2$ defines the set of states that can be partitioned into two sub-states, the first belonging to P_1, and the second belonging to P_2. $\uplus$ represents the disjoint union operator for partial functions. We define Emp as the singleton set containing the empty state. Similarly, we define $\ell \mapsto i$ as the singleton set containing the singleton state where ℓ maps to i. We write $\ell \mapsto \_$ as a shorthand for $\exists i. \ell \mapsto i$.

We also need to define precise assertions. An assertion P is precise if for any state, there is at most one sub-state that belongs to P.

Definition 6.1. An assertion P is precise if, and only if, for all σ, $\sigma_1 \subseteq \sigma$, and $\sigma_2 \subseteq \sigma$, such that $\sigma_1 \in P$ and $\sigma_2 \in P$, we have $\sigma_1 = \sigma_2$

6.2 Inference Rules

In Fig. 6.3, we present the set of inference rules for CSL. Informally, the judgment $I \vdash \{P\}\, c\, \{Q\}$ says that the state can be split implicitly into a shared part and a private part. The private part can be accessed only by c. P and Q are pre- and post-conditions for the private state. The shared part can be accessed by both c and its environment, but only when within atomic blocks. Accesses to the shared state must preserve its invariant I. Most of the rules are standard Hoare Logic rules adorned with an invariant I, which is not by used the rule. Therefore, we will only explain those rules that we think are peculiar to CSL.

$$\frac{}{\mathsf{Emp} \vdash \{Q \circ [\![\nu := e]\!]\}\, \nu := e\, \{Q\}}\ (\textsc{assignment}) \qquad \frac{}{\mathsf{Emp} \vdash \{Q \circ [\![a]\!]\}\, \langle a \rangle\, \{Q\}}\ (\textsc{action})$$

$$\frac{I \vdash \{P\}\, c_1\, \{P'\} \quad I \vdash \{P'\}\, c_2\, \{Q\}}{I \vdash \{P\}\, c_1; c_2\, \{Q\}}\ (\textsc{sequential}) \qquad \frac{}{\mathsf{Emp} \vdash \{P\}\, \textbf{skip}\, \{P\}}\ (\textsc{skip})$$

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad I \vdash \{P \cap \lfloor b \rfloor\}\, c_1\, \{Q\} \quad I \vdash \{P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q\}}{I \vdash \{P\}\, \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\, \{Q\}}\ (\textsc{conditional})$$

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad I \vdash \{P \cap \lfloor b \rfloor\}\, c\, \{P\}}{I \vdash \{P\}\, \textbf{while}\ b\ \textbf{do}\ c\, \{P \cap \lfloor \neg b \rfloor\}}\ (\textsc{loop})$$

$$\frac{I \vdash \{P_1\}\, c_1\, \{Q_1\} \quad I \vdash \{P_2\}\, c_2\, \{Q_2\}}{I \vdash \{P_1 * P_2\}\, c_1 \parallel c_2\, \{Q_1 * Q_2\}}\ (\textsc{parallel}) \qquad \frac{\mathsf{Emp} \vdash \{I * P\}\, c\, \{I * Q\}}{I \vdash \{P\}\, \textbf{atomic}\ c\, \{Q\}}\ (\textsc{atomic})$$

$$\frac{P \subseteq P' \quad I \vdash \{P'\}\, c\, \{Q'\} \quad Q' \subseteq Q}{I \vdash \{P\}\, c\, \{Q\}}\ (\textsc{consequence})$$

$$\frac{\forall x.\ I \vdash \{\mathcal{P}(x)\}\, c\, \{\mathcal{Q}(x)\}}{I \vdash \{\exists x.\ \mathcal{P}(x)\}\, c\, \{\exists x.\ \mathcal{Q}(x)\}}\ (\textsc{existential})$$

$$\frac{I \vdash \{P\}\, c\, \{Q\}}{I' * I \vdash \{P\}\, c\, \{Q\}}\ (\textsc{frame}) \qquad \frac{I' * I \vdash \{P\}\, c\, \{Q\}}{I \vdash \{I' * P\}\, c\, \{I' * Q\}}\ (\textsc{resource})$$

$$\frac{I \vdash \{P_1\}\, c\, \{Q_1\} \quad I \vdash \{P_2\}\, c\, \{Q_2\} \quad I \text{ is precise}}{I \vdash \{P_1 \cap P_2\}\, c\, \{Q_1 \cap Q_2\}}\ (\textsc{conjunction})$$

$$\frac{I \vdash \{P_1\}\, c\, \{Q_1\} \quad I \vdash \{P_2\}\, c\, \{Q_2\}}{I \vdash \{P_1 \cup P_2\}\, c\, \{Q_1 \cup Q_2\}}\ (\textsc{disjunction})$$

Figure 6.3: CSL inference rules

The first two rules, ASSIGNMENT and ACTION, are general rules to verify actions. We use the notation $Q \circ [\![a]\!]$ to specify the weakest pre-condition of a, given post-condition Q.

Definition 6.2. $Q \circ [\![a]\!]$ is a set of states where, for each state σ, we have:

1. exists σ' such that $(\sigma, \sigma') \in [\![a]\!]$
2. for all σ', such that $(\sigma, \sigma') \in [\![a]\!]$, we have $\sigma' \in Q$

Specific rules that are not based explicitly on the definition of $[\![-]\!]$ can be derived based on this rule. For instance, we can derive the following rule where p and i are integers:

$$\frac{}{\mathsf{Emp} \vdash \{p \mapsto \_\}\ [p] := i\ \{p \mapsto i\}}$$

The ACTION rule is similar, but specialized for synchronized actions (we write $\langle a \rangle$ which is syntactic sugar for **atomic** a). This rule is designed to be used for memory allocation and deallocation, given that those operations must be placed inside an atomic block to avoid races (according to the definition of race from Chapter 3). As one can deduct, CSL enforces, syntactically, that all memory allocation and deallocation is protected by an atomic block.

The PARALLEL rule is used to verify parallel composition. It allows it to be done modularly by splitting the private memory into two parts that must be rejoined once the execution finishes. An associated rule is the RESOURCE rule that demotes shared memory to private memory. From these two we can derive a fancier version of the parallel rule that creates extra memory for threads to share:

$$\frac{I' * I \vdash \{P_1\}\ c_1\ \{Q_1\} \quad I' * I \vdash \{P_2\}\ c_2\ \{Q_2\}}{I \vdash \{I' * P_1 * P_2\}\ c_1 \parallel c_2\ \{I' * Q_1 * Q_2\}}$$

Another important rule is the ATOMIC rule. It allows the exclusive access to the shared memory during the execution of the atomic block. During the verification of an atomic block the shared invariant becomes Emp therefore the actual invariant I does not need to be maintained. It must be restored back when the atomic block execution completes.

We also have a frame rule for the shared memory FRAME. It allows abstracting away extra memory when it is not needed. A more traditional frame rule, for the private memory, can be derived directly from this rule and the RESOURCE rule:

$$\frac{I \vdash \{P'\}\, c\, \{Q'\}}{I \vdash \{I' * P\}\, c\, \{I' * Q\}}$$

Finally, we would like to point out that the only rule that requires I to be precise is the CONJUNCTION rule. This is needed for proving soundness, in particular Lemma 6.29. A fancier version of the CONJUNCTION rule can also be incorporated, this is discussed in Sec. 6.4.

6.3 Soundness

In the following sections, we present the soundness proof of CSL regarding all three concurrent semantics from Chapter 3. The soundness proof to be presented uses a semantic approach with indexing.

6.3.1 With Regard to the Interleaved Semantics

In this section, we present the soundness proof with regard to the interleaved semantics from Sec. 3.7. The proof is structured around the following definition:

Definition 6.3. $I \models \langle T, \sigma \rangle \rhd_0 Q$ always holds; $I \models \langle T, \sigma \rangle \rhd_{n+1} Q$ holds if, and only if, the following are true:

1. T is either 0- or 1-atomic

2. $\neg \langle T, \sigma \rangle \longmapsto \mathsf{abort}$

3. $\neg \langle T, \sigma \rangle \longmapsto \mathsf{race}$

4. If $\langle T, \sigma \rangle \longmapsto \langle T', \sigma' \rangle$, then

   (a) If both T and T' are 1-atomic, then $I \models \langle T', \sigma' \rangle \rhd_n Q$

   (b) If both T and T' are 0-atomic, then $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $I \models \langle T', \sigma' \rangle \rhd_n Q$

   (c) If T is 0-atomic and T' is 1-atomic, then for all σ_1 and σ_2, such that $\sigma_2 = \sigma_1 \uplus \sigma'$ and $\sigma_1 \in I$, we have $I \models \langle T', \sigma_2 \rangle \rhd_n Q$

   (d) If T is 1-atomic and T' is 0-atomic, then exists σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in I$, and $I \models \langle T', \sigma_2 \rangle \rhd_n Q$

5. If $T = \mathbf{skip}$, then $\sigma \in Q$

We define $I \models \langle T, \sigma \rangle \rhd Q$ as $\forall n.\ I \models \langle T, \sigma \rangle \rhd_n Q$.

The triple $I \models \langle T, \sigma \rangle \rhd Q$ ensures that each step performed by a program configuration has at most one ongoing atomic block execution (item 1), does not abort (item 2), and is not at a race condition (item 3). Furthermore, if it reaches a final configuration $\langle \mathbf{skip}, \sigma' \rangle$, then σ' must satisfy post-condition Q (item 5). This definition also manages proper access to private memory (item 4). If the current configuration has an ongoing atomic block execution (item (a)), then it already has a hold of the shared memory, and it can perform the step with out constraints. If the current configuration does not have an ongoing atomic block execution (item (b)), then no memory allocation or deallocation must happen to avoid a race-condition with the environment, which may have an ongoing atomic block execution performing memory allocation or deallocation. This constraint is enforced by the condition $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$. If the current program configuration is starting to execute a top-level atomic block (item (c)), then it must get a hold on the shared memory, assuming it satisfies I. If the current program configuration is completing the execution of a top-level atomic block (item (d)), then it must return the shared memory ensuring that it satisfies I.

We present the soundness proof in three sections, following the order:

1. Auxiliary lemmas for CSL triples, as in Def. 6.3

2. Semantic rules, each corresponding to a syntactic rule from Fig. 6.3

3. Top level soundness theorem

Auxiliary lemmas. Given that programs may diverge, and since $I \models \langle T, \sigma \rangle \rhd Q$ is defined in terms of itself, we used indexing to ensure this definition is well-founded. Lemma 6.4 allows greater flexibility when dealing with indexing.

Lemma 6.4. If $I \models \langle T, \sigma \rangle \rhd_{n_1} Q$, and $n_2 \leq n_1$, then $I \models \langle T, \sigma \rangle \rhd_{n_2} Q$

Proof. By induction over n_1. If $n_1 = 0$, then $n_2 = 0$ as well, by Def. 6.3, we conclude. If $n_1 > 0$, by Def. 6.3, assuming (a) $I \models \langle T, \sigma \rangle \rhd_{n_1} Q$ and (b) $n_2 \leq n_1$, we have 5 cases:

- From (a), by Def. 6.3 (item 1), we know that T is 0- or 1-atomic
- From (a), by Def. 6.3 (item 2), we know that $\langle T, \sigma \rangle \longmapsto \textsf{abort}$ is false
- From (a), by Def. 6.3 (item 3), we know that $\langle T, \sigma \rangle \longmapsto \textsf{race}$ is false
- If (c) $\langle T, \sigma \rangle \longmapsto \langle T', \sigma' \rangle$, then we have 4 cases:
  - If both T and T' are 1-atomic, we need to show that $I \models \langle T', \sigma' \rangle \rhd_{(n_2-1)} Q$. From (a) and (c), by Def. 6.3 (item 4.a), we know that (d) $I \models \langle T', \sigma' \rangle \rhd_{(n_1-1)} Q$. Trivially, from (b), we know that (e) $n_2 - 1 \leq n_1 - 1$. From (d) and (e), by the induction hypothesis, we conclude
  - If both T and T' are 0-atomic, we need to show that $\textsf{dom}(\sigma) = \textsf{dom}(\sigma')$ and $I \models \langle T', \sigma' \rangle \rhd_{(n_2-1)} Q$. From (a) and (c), by Def. 6.3 (item 4.b), we know already that $\textsf{dom}(\sigma) = \textsf{dom}(\sigma')$, and also that (d) $I \models \langle T', \sigma' \rangle \rhd_{(n_1-1)} Q$. Trivially, from (b), we know that (e) $n_2 - 1 \leq n_1 - 1$. From (d) and (e), by the induction hypothesis, we conclude
  - If T is 0-atomic and T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (d) $\sigma_2 = \sigma_1 \uplus \sigma'$ and (e) $\sigma_1 \in I$, we have $I \models \langle T', \sigma_2 \rangle \rhd_{(n_2-1)} Q$. From (a), (c), (d), and (e), by Def. 6.3 (item 4.c), we know that (f) $I \models \langle T', \sigma_2 \rangle \rhd_{(n_1-1)} Q$.

Trivially, from (b), we know that (g) $n_2 - 1 \leq n_1 - 1$. From (f) and (g), by the induction hypothesis, we conclude

– If T is 1-atomic and T' is 0-atomic, we need to show that exists σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in I$, and $I \models \langle T', \sigma_2 \rangle \rhd_{(n_2-1)} Q$. From (a) and (c), by Def. 6.3 (item 4.d), we know there exists σ'_1 and σ'_2, such that (d) $\sigma' = \sigma'_1 \uplus \sigma'_2$, (e) $\sigma'_1 \in I$, and (f) $I \models \langle T', \sigma'_2 \rangle \rhd_{(n_1-1)} Q$. Trivially, from (b), we know that (g) $n_2 - 1 \leq n_1 - 1$. From (f) and (g), by the induction hypothesis, we have (h) $I \models \langle T', \sigma'_2 \rangle \rhd_{(n_2-1)} Q$. Instantiating the goal with σ'_1 and σ'_2, from (d), (e), and (h), we conclude

- We assume (c) $T = \textbf{skip}$. From (a) and (c), by Def. 6.3 (item 5), we know that $\sigma \in Q$ and conclude □

The following lemma allows the weakening of Q in a CSL triple.

Lemma 6.5. If $I \models \langle T, \sigma \rangle \rhd_n Q$, and $Q \subseteq Q'$, then $I \models \langle T, \sigma \rangle \rhd_n Q'$

Proof. By induction over n. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, assuming (a) $I \models \langle T, \sigma \rangle \rhd_n Q$ and (b) $Q \subseteq Q'$, we have 5 cases:

- From (a), by Def. 6.3 (item 1), we know that T is 0- or 1-atomic
- From (a), by Def. 6.3 (item 2), we know that $\langle T, \sigma \rangle \longmapsto \mathsf{abort}$ is false
- From (a), by Def. 6.3 (item 3), we know that $\langle T, \sigma \rangle \longmapsto \mathsf{race}$ is false
- If (c) $\langle T, \sigma \rangle \longmapsto \langle T', \sigma' \rangle$, then we have 4 cases:

  – If both T and T' are 1-atomic, we need to show that $I \models \langle T', \sigma' \rangle \rhd_{n-1} Q'$. From (a) and (c), by Def. 6.3 (item 4.a), we know that (d) $I \models \langle T', \sigma' \rangle \rhd_{n-1} Q$. From (b) and (d), by the induction hypothesis, we conclude

  – If both T and T' are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $I \models \langle T', \sigma' \rangle \rhd_{n-1} Q'$. From (a) and (c), by Def. 6.3 (item 4.b), we know already that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, and also that (d) $I \models \langle T', \sigma' \rangle \rhd_{n-1} Q$. From (b) and (d), by the induction hypothesis, we conclude

  – If T is 0-atomic and T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (d) $\sigma_2 = \sigma_1 \uplus \sigma'$ and (e) $\sigma_1 \in I$, we have $I \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q'$. From (a), (c), (d), and (e), by Def. 6.3 (item 4.c), we know that (f) $I \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q$. From (b) and (f), by the induction hypothesis, we conclude

  – If T is 1-atomic and T' is 0-atomic, we need to show that exists σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in I$, and $I \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q'$. From (a) and (c), by Def. 6.3 (item 4.d), we know there exists σ_1' and σ_2', such that (d) $\sigma' = \sigma_1' \uplus \sigma_2'$, (e) $\sigma_1' \in I$, and (f) $I \models \langle T', \sigma_2' \rangle \rhd_{n-1} Q$. From (b) and (f), by the induction hypothesis, we have (g) $I \models \langle T', \sigma_2' \rangle \rhd_{n-1} Q'$. Instantiating the goal with σ_1' and σ_2', from (d), (e), and (g), we conclude

- We assume (c) $T = \textbf{skip}$. From (a) and (c), by Def. 6.3 (item 5), we know that (d) $\sigma \in Q$. From (b) and (d), we know that $\sigma \in Q'$ and conclude □

We can construct a CSL triple from **skip** using the following lemma.

Lemma 6.6. If $\sigma \in Q$, then $I \models \langle \textbf{skip}, \sigma \rangle \rhd_n Q$

Proof. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, we have 5 cases:

- By Def. 3.1, we know that **skip** is 0-atomic
- From the semantics, we know that $\langle \textbf{skip}, \sigma \rangle \longmapsto \mathsf{abort}$ is false
- From the semantics, we know that $\langle \textbf{skip}, \sigma \rangle \longmapsto \mathsf{race}$ is false
- From the semantics, we know that $\langle \textbf{skip}, \sigma \rangle \longmapsto \langle T', \sigma' \rangle$ is false
- Since $\textbf{skip} = \textbf{skip}$, and $\sigma \in Q$, we conclude □

The following lemma is used for sequential composition of triples.

Lemma 6.7. If $I \models \langle T, \sigma \rangle \rhd_n P$, and, for all $\sigma' \in P$, we have $I \models \langle c', \sigma' \rangle \rhd_n Q$, then

1. If $T = c$, then $I \models \langle c; c', \sigma \rangle \rhd_n Q$
2. If $T = \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$, and $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$ is 0- or 1-atomic, then $I \models \langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} (c; c'), \sigma \rangle \rhd_n Q$
3. If $T = \langle\!\langle T' \rangle\!\rangle_{\mathbf{a}} c$, then $I \models \langle \langle\!\langle T' \rangle\!\rangle_{\mathbf{a}} (c; c'), \sigma \rangle \rhd_n Q$

Proof. By induction over n. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, we assume (a) $I \models \langle T, \sigma \rangle \rhd_n P$ and (b) for all $\sigma' \in P$, we have $I \models \langle c', \sigma' \rangle \rhd_n Q$. We establish (c) for all $\sigma' \in P$, we have $I \models \langle c', \sigma' \rangle \rhd_{n-1} Q$:

- We assume $\sigma' \in P$, from (b), using Lemma 6.4, we conclude

Then we consider 3 cases:

- If $T = c$, then we need to show that $I \models \langle c; c', \sigma \rangle \rhd_n Q$. By Def. 6.3, we have 5 cases:
  - By Def. 3.1, we know that $c; c'$ is 0-atomic
  - From (a), by Def. 6.3 (item 2), we know that (d) $\langle c, \sigma \rangle \longmapsto \mathsf{abort}$ is false. From (d), and the semantics, we know that $\langle c; c', \sigma \rangle \longmapsto \mathsf{abort}$ is also false
  - From the semantics, we know that $\langle c; c', \sigma \rangle \longmapsto \mathsf{race}$ is false
  - If (d) $\langle c; c', \sigma \rangle \longmapsto \langle T', \sigma' \rangle$, from the semantics, we have 4 cases:
    * If $c = \mathbf{skip}$, $T' = c'$, and $\sigma' = \sigma$, since both $\mathbf{skip}; c'$ and c' are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, which holds trivially, and $I \models \langle c', \sigma' \rangle \rhd_{n-1} Q$. From (a), by Def. 6.3 (item 5), we know that (e) $\sigma \in P$. From (e) and (c), we conclude
    * If $T' = c''; c'$, and (e) $\langle c, \sigma \rangle \longmapsto \langle c'', \sigma' \rangle$, given that both $c; c'$ and $c''; c'$ are 0-atomic, we need to show $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $I \models \langle c''; c', \sigma' \rangle \rhd_{n-1} Q$. From (a) and (e), by Def. 6.3 (item 4.b), we have (f) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and (g) $I \models \langle c'', \sigma' \rangle \rhd_{n-1} P$. From (g) and (c), using the induction hypothesis (item 1), we have (h) $I \models \langle c''; c', \sigma' \rangle \rhd_{n-1} Q$. From (f) and (h), we conclude

    * If $c = \mathbf{S}[\, c_1 \parallel c_2 \,]$, $T' = \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,]; c')$, and $\sigma' = \sigma$, since both $c; c'$ and $\langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,]; c')$ are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, which holds trivially, and $I \models \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,]; c'), \sigma \rangle \triangleright_{n-1} Q$. From the semantics, we know that (e) $\langle c, \sigma \rangle \longmapsto \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,]), \sigma \rangle$. From (a) and (e), by Def. 6.3 (item 4.b), we know that (f) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma)$ and (g) $I \models \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,]), \sigma \rangle \triangleright_{n-1} P$. From (g) and (c), using the induction hypothesis (item 2), we conclude
    * If $c = \mathbf{S}[\,\mathbf{atomic}\ c''\,]$, $T' = \langle\!\langle c'' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\mathbf{skip}\,]; c')$, and $\sigma' = \sigma$, since $c; c'$ is 0-atomic and $\langle\!\langle c'' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\mathbf{skip}\,]; c')$ is 1-atomic, we need to show that for all σ_1 and σ_2, such that (e) $\sigma_2 = \sigma_1 \uplus \sigma$, and (f) $\sigma_1 \in I$, we have $I \models \langle T', \sigma_2 \rangle \triangleright_{n-1} Q$. From the semantics, we know that (g) $\langle c, \sigma \rangle \longmapsto \langle \langle\!\langle c'' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\mathbf{skip}\,]), \sigma \rangle$. From (a), (g), (e), and (f), by Def. 6.3 (item 4.c), we have
(h) $I \models \langle \langle\!\langle c'' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\mathbf{skip}\,]), \sigma_2 \rangle \triangleright_{n-1} P$. From (h) and (c), using the induction hypothesis (item 3), we conclude
  - We know that $c; c' \neq \mathbf{skip}$

- If $T = \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$, and (d) $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$ is 0- or 1-atomic, then we need to show that $I \models \langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}(c; c'), \sigma \rangle \triangleright_n Q$. By Def. 6.3, we have 5 cases:
  - From (d), by Def. 3.1, we know that $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}(c; c')$ is 0- or 1-atomic
  - From (a), by Def. 6.3 (item 2), we know that (e) $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c, \sigma \rangle \longmapsto \mathsf{abort}$ is false. From (e), and the semantics, we know that $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}(c; c'), \sigma \rangle \longmapsto \mathsf{abort}$ is also false
  - From (a), by Def. 6.3 (item 3), we know that (e) $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c, \sigma \rangle \longmapsto \mathsf{race}$ is false. From (e), and the semantics, we know that $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}(c; c'), \sigma \rangle \longmapsto \mathsf{race}$ is also false
  - If (e) $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}(c; c'), \sigma \rangle \longmapsto \langle T', \sigma' \rangle$, from the semantics, we have 3 cases:
    * If $T_1 = \mathbf{skip}$, $T_2 = \mathbf{skip}$, $T' = c; c'$, and $\sigma' = \sigma$, since both $\langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathbf{p}}(c; c')$

and $c; c'$ are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, which holds trivially, and $I \models \langle c; c', \sigma\rangle \rhd_{n-1} Q$. From the semantics, we know that (f) $\langle \langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathrm{p}} c, \sigma\rangle \longmapsto \langle c, \sigma\rangle$. From (a) and (f), by Def. 6.3 (item 4.b), we know that (g) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma)$ and (h) $I \models \langle c, \sigma\rangle \rhd_{n-1} P$. From (h), (c), by the induction hypothesis (item 1), we conclude

* If $T' = \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}}(c; c')$, and (f) $\langle T_1, \sigma\rangle \longmapsto \langle T_1', \sigma'\rangle$, we have 4 cases:
  - If both T_1 and T_1' are 1-atomic, we have to show that $I \models \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}}(c; c'), \sigma'\rangle \rhd_{n-1} Q$. From (f), and the semantics, we know (g) $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathrm{p}} c, \sigma\rangle \longmapsto \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}} c, \sigma'\rangle$. From (a) and (g), by Def. 6.3 (item 4.a), we know (h) $I \models \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}} c, \sigma'\rangle \rhd_{n-1} P$. From (h) and (c), using the induction hypothesis (item 2), we conclude
  - If both T_1 and T_1' are 0-atomic, we have to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $I \models \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}}(c; c'), \sigma'\rangle \rhd_{n-1} Q$. From (f), and the semantics, we know (g) $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathrm{p}} c, \sigma\rangle \longmapsto \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}} c, \sigma'\rangle$. From (a) and (g), by Def. 6.3 (item 4.b), we know (h) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and (i) $I \models \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}} c, \sigma'\rangle \rhd_{n-1} P$. From (i) and (c), using the induction hypothesis (item 2), we know (j) $I \models \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}}(c; c'), \sigma'\rangle \rhd_{n-1} Q$. From (h) and (j), we conclude
  - If T_1 is 0-atomic and T_1' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (g) $\sigma_2 = \sigma_1 \uplus \sigma'$ and (h) $\sigma_1 \in I$, we have $I \models \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}}(c; c'), \sigma_2\rangle \rhd_{n-1} Q$. From (f), and the semantics, we know (i) $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathrm{p}} c, \sigma\rangle \longmapsto \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}} c, \sigma'\rangle$. From (a), (i), (g), and (h), by Def. 6.3 (item 4.c), we know that (j) $I \models \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}} c, \sigma_2\rangle \rhd_{n-1} P$. From (j) and (c), using the induction hypothesis (item 2), we conclude
  - If T_1 is 1-atomic and T_1' is 0-atomic, we need to show that exists σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in I$, and $I \models \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}}(c; c'), \sigma_2\rangle \rhd_{n-1} Q$. From (f), and the semantics, we know

(g) $\langle\langle\!\langle T_1, T_2\rangle\!\rangle_{\mathsf{p}} c, \sigma\rangle \longmapsto \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}} c, \sigma'\rangle$. From (a) and (g), by Def. 6.3 (item 4.d), we know there exists σ_1' and σ_2' such that (i) $\sigma' = \sigma_1' \uplus \sigma_2'$, (j) $\sigma_1' \in I$, and (k) $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}} c, \sigma_2'\rangle \rhd_{n-1} P$. From (k) and (c), using the induction hypothesis (item 2), we know (l) $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}} (c; c'), \sigma_2'\rangle \rhd_{n-1} Q$. Instantiating the goal with σ_1' and σ_2', from (i), (j), and (l), we conclude

   * If $T' = \langle\!\langle T_1, T_2'\rangle\!\rangle_{\mathsf{p}} (c; c')$, and (f) $\langle T_2, \sigma\rangle \longmapsto \langle T_2', \sigma'\rangle$, the proof is symmetric to the previous case

 – We know that $\langle\!\langle T_1, T_2\rangle\!\rangle_{\mathsf{p}} (c; c') \neq \mathbf{skip}$

• If $T = \langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} c$, then we need to show that $I \models \langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} (c; c'), \sigma\rangle \rhd_n Q$. By Def. 6.3, we have 5 cases:

 – By Def. 3.1, we know that $\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} (c; c')$ is 1-atomic

 – From (a), by Def. 6.3 (item 2), we know that (d) $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} c, \sigma\rangle \longmapsto \mathsf{abort}$ is false. From (d), and the semantics, we know that $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} (c; c'), \sigma\rangle \longmapsto \mathsf{abort}$ is also false

 – From (a), by Def. 6.3 (item 3), we know that (d) $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} c, \sigma\rangle \longmapsto \mathsf{race}$ is false. From (d), and the semantics, we know that $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} (c; c'), \sigma\rangle \longmapsto \mathsf{race}$ is also false

 – If (d) $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} (c; c'), \sigma\rangle \longmapsto \langle T'', \sigma'\rangle$, from the semantics, we have 2 cases:

   * If $T' = \mathbf{skip}$, $T'' = c; c'$, and $\sigma' = \sigma$, since $\langle\!\langle \mathbf{skip}\rangle\!\rangle_{\mathsf{a}} (c; c')$ is 1-atomic and $c; c'$ is 0-atomic, we need to show that exists σ_1 and σ_2 such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in I$, and $I \models \langle c; c', \sigma_2\rangle \rhd_{n-1} Q$. From the semantics, we know that (e) $\langle\langle\!\langle \mathbf{skip}\rangle\!\rangle_{\mathsf{a}} c, \sigma\rangle \longmapsto \langle c, \sigma\rangle$. From (a) and (e), by Def. 6.3 (item 4.d), we know that exists σ_1' and σ_2' such that (f) $\sigma = \sigma_1' \uplus \sigma_2'$, (g) $\sigma_1' \in I$, and (h) $I \models \langle c, \sigma_2'\rangle \rhd_{n-1} P$. From (h), (c), by the induction hypothesis (item 1), we have (i) $I \models \langle c; c', \sigma_2'\rangle \rhd_{n-1} Q$. Instantiating the goal with σ_1' and σ_2', from (f), (g), and (i), we conclude

    * If $T'' = \langle\!\langle T''' \rangle\!\rangle_{\mathsf{a}}(c; c')$, and (e) $\langle T', \sigma\rangle \longmapsto \langle T''', \sigma'\rangle$, given that both $\langle\!\langle T' \rangle\!\rangle_{\mathsf{a}}(c; c')$ and $\langle\!\langle T''' \rangle\!\rangle_{\mathsf{a}}(c; c')$ are 1-atomic, we need to show $I \models \langle \langle\!\langle T''' \rangle\!\rangle_{\mathsf{a}}(c; c'), \sigma'\rangle \rhd_{n-1} Q$. From (e), and the semantics, we know (f) $\langle \langle\!\langle T' \rangle\!\rangle_{\mathsf{a}} c, \sigma\rangle \longmapsto \langle \langle\!\langle T''' \rangle\!\rangle_{\mathsf{a}} c, \sigma'\rangle$. From (a) and (f), by Def. 6.3 (item 4.a), then (g) $I \models \langle \langle\!\langle T''' \rangle\!\rangle_{\mathsf{a}} c, \sigma'\rangle \rhd_{n-1} P$. From (g) and (c), using the induction hypothesis (item 3), we conclude

  – We know that $\langle\!\langle T' \rangle\!\rangle_{\mathsf{a}}(c; c') \neq \textbf{skip}$ □

Conditional commands can be introduced using the following lemma.

Lemma 6.8. If $I \models \langle c, \sigma\rangle \rhd_n Q$, then

1. If $\sigma \in \lfloor b \rfloor$, then $I \models \langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \sigma\rangle \rhd_n Q$
2. If $\sigma \in \lfloor \neg b \rfloor$, then $I \models \langle \textbf{if } b \textbf{ then } c' \textbf{ else } c, \sigma\rangle \rhd_n Q$

Proof. From assumption (a) $I \models \langle c, \sigma\rangle \rhd_n Q$ we have 2 cases:

- We assume (b) $\sigma \in \lfloor b \rfloor$. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, we have 5 cases:
  – By Def. 3.1, we know that (c) $\textbf{if } b \textbf{ then } c \textbf{ else } c'$ is 0-atomic
  – From the semantics, we know that $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \sigma\rangle \longmapsto \mathsf{abort}$ is false if $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \sigma\rangle \longrightarrow \mathsf{abort}$ is false. From the sequential semantics, we know $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \sigma\rangle \longrightarrow \mathsf{abort}$ is false if there exists z such that $[\![b]\!]_\sigma = z$. From (b), we know $[\![b]\!]_\sigma = \textit{true}$ and conclude
  – From the semantics, we know that $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \sigma\rangle \longmapsto \mathsf{race}$ is false
  – From the semantics, given (b), we know that $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \sigma\rangle \longmapsto \langle c, \sigma\rangle$. From (c), and since c is 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma)$ and $I \models \langle c, \sigma\rangle \rhd_{n-1} Q$. From (a), using Lemma 6.4, we conclude
  – We know that $\textbf{if } b \textbf{ then } c \textbf{ else } c' \neq \textbf{skip}$
- We assume (b) $\sigma \in \lfloor \neg b \rfloor$. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, we have 5 cases:

  - By Def. 3.1, we know that (c) **if** b **then** c' **else** c is 0-atomic
  - From the semantics, we know that $\langle$**if** b **then** c' **else** $c, \sigma\rangle \longmapsto \mathsf{abort}$ is false if $\langle$**if** b **then** c' **else** $c, \sigma\rangle \longrightarrow \mathsf{abort}$ is false. From the sequential semantics, we know $\langle$**if** b **then** c' **else** $c, \sigma\rangle \longrightarrow \mathsf{abort}$ is false if there exists z such that $[\![b]\!]_\sigma = z$. From (b), we know $[\![b]\!]_\sigma = \mathit{false}$ and conclude
  - From the semantics, we know that $\langle$**if** b **then** c' **else** $c, \sigma\rangle \longmapsto \mathsf{race}$ is false
  - From the semantics, given (b), we know that $\langle$**if** b **then** c' **else** $c, \sigma\rangle \longmapsto \langle c, \sigma\rangle$. From (c), and since c is 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma)$ and $I \models \langle c, \sigma\rangle \rhd_{n-1} Q$. From (a), using Lemma 6.4, we conclude
  - We know that **if** b **then** c' **else** $c \neq$ **skip** □

A loop command can be introduced using the following lemma.

Lemma 6.9. If $P \subseteq \lfloor b\rfloor \cup \lfloor \neg b\rfloor$, $\sigma \in P$, and, for all $\sigma' \in P \cap \lfloor b\rfloor$, we have $I \models \langle c, \sigma'\rangle \rhd_n P$, then $I \models \langle$**while** b **do** $c, \sigma\rangle \rhd_n (P \cap \lfloor \neg b\rfloor)$

Proof. By induction over n. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, assuming (a) $P \subseteq \lfloor b\rfloor \cup \lfloor \neg b\rfloor$, (b) $\sigma \in P$, and, (c) for all $\sigma' \in P \cap \lfloor b\rfloor$, we have $I \models \langle c, \sigma'\rangle \rhd_n P$, we have 5 cases:

- By Def. 3.1, we know that (d) **while** b **do** c is 0-atomic
- From the semantics, we know that $\langle$**while** b **do** $c, \sigma\rangle \longmapsto \mathsf{abort}$ is false
- From the semantics, we know that $\langle$**while** b **do** $c, \sigma\rangle \longmapsto \mathsf{race}$ is false
- From the semantics, $\langle$**while** b **do** $c, \sigma\rangle \longmapsto \langle$**if** b **then** $(c;$ **while** b **do** $c)$ **else skip**$, \sigma\rangle$. From (d), and since **if** b **then** $(c;$ **while** b **do** $c)$ **else skip** is 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma)$ and $I \models \langle$**if** b **then** $(c;$ **while** b **do** $c)$ **else skip**$, \sigma\rangle \rhd_{n-1} (P \cap \lfloor \neg b\rfloor)$. From (a) and (b), we know (e) $\sigma \in \lfloor b\rfloor \cup \lfloor \neg b\rfloor$. From (e), we have two cases:
  - We assume (f) $\sigma \in \lfloor b\rfloor$. From (b) and (f), we know (g) $\sigma \in P \cap \lfloor b\rfloor$. We establish (h) for all $\sigma' \in P \cap \lfloor b\rfloor$, we have $I \models \langle c, \sigma'\rangle \rhd_{n-1} P$:

    * We assume $\sigma' \in P \cap \lfloor b \rfloor$, from (c), using Lemma 6.4, we conclude

  From (g) and (h), we know (i) $I \models \langle c, \sigma \rangle \triangleright_{n-1} P$. We establish (j) for all $\sigma' \in P$, we have $I \models \langle \textbf{while}\ b\ \textbf{do}\ c, \sigma' \rangle \triangleright_{n-1} (P \cap \lfloor \neg b \rfloor)$:

    * We assume $\sigma' \in P$, with (a) and (h), using the induction hypothesis, we conclude

  From (i) and (j), using Lemma 6.7 (item 1), we get (k) $I \models \langle c; \textbf{while}\ b\ \textbf{do}\ c, \sigma \rangle \triangleright_{n-1} (P \cap \lfloor \neg b \rfloor)$. From (f) and (k), using Lemma 6.8 (item 1), we conclude

  – We assume (f) $\sigma \in \lfloor \neg b \rfloor$. From (b) and (f), we know (g) $\sigma \in P \cap \lfloor \neg b \rfloor$. From (g), using Lemma 6.6, we know (h) $I \models \langle \textbf{skip}, \sigma \rangle \triangleright_{n-1} (P \cap \lfloor \neg b \rfloor)$. From (f) and (h), using Lemma 6.8 (item 2), we conclude

- We know that $\textbf{while}\ b\ \textbf{do}\ c \neq \textbf{skip}$ □

The following lemma is used for parallel composition of triples.

Lemma 6.10. If $I \models \langle T_1, \sigma_1 \rangle \triangleright_n Q_1$, $I \models \langle T_2, \sigma_2 \rangle \triangleright_n Q_2$, and $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathrm{p}}\textbf{skip}$ is 0- or 1-atomic, then $I \models \langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathrm{p}}\textbf{skip}, \sigma_1 \uplus \sigma_2 \rangle \triangleright_n (Q_1 * Q_2)$

Proof. By induction over n. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, assuming (a) $I \models \langle T_1, \sigma_1 \rangle \triangleright_n Q_1$, (b) $I \models \langle T_2, \sigma_2 \rangle \triangleright_n Q_2$, and (c) $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathrm{p}}\textbf{skip}$ is 0- or 1-atomic, we have 5 cases:

- From (c), we conclude

- From the semantics, we know that $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathrm{p}}\textbf{skip}, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \textsf{abort}$ if either:

  – $\langle T_1, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \textsf{abort}$. From (a), by Def. 6.3 (item 2), we know (d) $\langle T_1, \sigma_1 \rangle \longmapsto \textsf{abort}$ is false. From (d), using Lemma 3.22 (item 1), we conclude

  – or, $\langle T_2, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \textsf{abort}$. From (b), by Def. 6.3 (item 2), we know (e) $\langle T_2, \sigma_2 \rangle \longmapsto \textsf{abort}$ is false. From (e), using Lemma 3.22 (item 1), we conclude

- From the semantics, we know that $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathrm{p}}\textbf{skip}, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \textsf{race}$ if either:

– $\langle T_1, \sigma_1 \uplus \sigma_2\rangle \longmapsto \textsf{race}$. From (a), by Def. 6.3 (item 3), we know (f) $\langle T_1, \sigma_1\rangle \longmapsto \textsf{race}$ is false. From (d) and (f), using Lemma 3.22 (item 2), we conclude

– or, $\langle T_2, \sigma_1 \uplus \sigma_2\rangle \longmapsto \textsf{race}$. From (b), by Def. 6.3 (item 3), we know (g) $\langle T_2, \sigma_2\rangle \longmapsto \textsf{race}$ is false. From (e) and (g), using Lemma 3.22 (item 2), we conclude

– or, $T_1 = \mathbf{T}_1[\,c_1\,]$, $T_2 = \mathbf{T}_2[\,c_2\,]$, $\langle c_1, \sigma_1 \uplus \sigma_2\rangle \underset{\delta_1}{\longrightarrow} \langle c_1', \sigma'\rangle$, $\langle c_2, \sigma'\rangle \underset{\delta_2}{\longrightarrow} \kappa$ and $\delta_1 \not\smile \delta_2$. By contradiction, we will assume (f) $\langle c_1, \sigma_1 \uplus \sigma_2\rangle \underset{\delta_1}{\longrightarrow} \langle c_1', \sigma'\rangle$, and (g) $\langle c_2, \sigma'\rangle \underset{\delta_2}{\longrightarrow} \kappa$ in order to obtain $\delta_1 \overset{\rightharpoonup}{\smile} \delta_2$. From the semantics, and (d), we know (h) $\langle c1, \sigma_1\rangle \longrightarrow \textsf{abort}$ is false. From (h) and (f), using Lemma 3.16 (item 2), we know there exists σ_1' such that (i) $\sigma' = \sigma_1' \uplus \sigma_2$ and (j) $\langle c1, \sigma_1\rangle \underset{\delta_1}{\longrightarrow} \langle c_1', \sigma_1'\rangle$. From the semantics, and (e), we know (k) $\langle c2, \sigma_2\rangle \longrightarrow \textsf{abort}$ is false. From (k), using Lemma 3.16 (item 1), we know (l) $\langle c2, \sigma_1' \uplus \sigma_2\rangle \longrightarrow \textsf{abort}$ is false. From (g), (i), and (l), we know there exists c_2' and σ'' such that (m) $\kappa = \langle c_2', \sigma''\rangle$. From (k), (g) and (m), using Lemma 3.16 (item 2), we know there exists σ_2' such that (n) $\sigma'' = \sigma_1' \uplus \sigma_2'$ and (o) $\langle c_2, \sigma_2\rangle \underset{\delta_2}{\longrightarrow} \langle c_2', \sigma_2'\rangle$. From (j), using Remark 3.10 (item 2), by Def. 3.4, we know that (p) $\delta_1 \subseteq (\emptyset, \textsf{dom}(\sigma_1) \cup \textsf{dom}(\sigma_1'))$. From (o), using Remark 3.10 (item 2), by Def. 3.4, we know that (q) $\delta_2 \subseteq (\emptyset, \textsf{dom}(\sigma_2) \cup \textsf{dom}(\sigma_2'))$. Then, we have 2 cases:

  * If $\mathbf{T}_1[\,c_1\,]$ is 0-atomic, then from (a) and (j), by Def. 6.3 (item 4.b), we know that (r) $\textsf{dom}(\sigma_1) = \textsf{dom}(\sigma_1')$. From (p) and (r), we know that (s) $\delta_1 \subseteq (\emptyset, \textsf{dom}(\sigma_1'))$. From (i), (n), we know that (t) $\textsf{dom}(\sigma_1') \cap (\textsf{dom}(\sigma_2) \cup \textsf{dom}(\sigma_2')) = \emptyset$. From (t), (s), and (q), we know that $\delta_1.ws \cap (\delta_2.rs \cup \delta_2.ws) = \emptyset$ and conclude

  * If $\mathbf{T}_1[\,c_1\,]$ is 1-atomic, from (c) we know that (r) $\mathbf{T}_2[\,c_2\,]$ is 0-atomic. From (b), (o), and (r), by Def. 6.3 (item 4.b), we know that (s) $\textsf{dom}(\sigma_2) = \textsf{dom}(\sigma_2')$. From (q) and (s), we know that (t) $\delta_2 \subseteq (\emptyset, \textsf{dom}(\sigma_2))$. From (i), we know that (u) $(\textsf{dom}(\sigma_1) \cup \textsf{dom}(\sigma_1')) \cap \textsf{dom}(\sigma_2) = \emptyset$. From (u), (p), and (t), we know that $\delta_1.ws \cap (\delta_2.rs \cup \delta_2.ws) = \emptyset$ and conclude

- or, $T_1 = \mathbf{T}_1[\, c_1 \,]$, $T_2 = \mathbf{T}_2[\, c_2 \,]$, $\langle c_2, \sigma_1 \uplus \sigma_2 \rangle \xrightarrow[\delta_2]{} \langle c_2', \sigma' \rangle$, $\langle c_1, \sigma' \rangle \xrightarrow[\delta_1]{} \kappa$ and $\delta_2 \not\smile \delta_1$. The proof is symmetric to the previous case

- From the semantics, if (f) $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \langle T', \sigma' \rangle$, then either:

  - $T_1 = \mathbf{skip}$, $T_2 = \mathbf{skip}$, $T' = \mathbf{skip}$, and $\sigma' = \sigma_1 \uplus \sigma_2$. Since both $\langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}$ and $\mathbf{skip}$ are 0-atomic, we need to show that $\mathsf{dom}(\sigma') = \mathsf{dom}(\sigma_1 \uplus \sigma_2)$, which holds trivially, and that $I \models \langle \mathbf{skip}, \sigma' \rangle \rhd_{n-1} (Q_1 {*} Q_2)$. From (a), by Def. 6.3 (item 5), we know (g) $\sigma_1 \in Q_1$. From (b), by Def. 6.3 (item 5), we know (h) $\sigma_2 \in Q_2$. From (g) and (h), we know (i) $\sigma' \in Q_1 * Q_2$. From (i), using Lemma 6.6, we conclude

  - or, $T' = \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}$ and (g) $\langle T_1, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \langle T_1', \sigma' \rangle$. From (d) and (g), using Lemma 3.22 (item 3), we know that exists σ_1' such that $\sigma' = \sigma_1' \uplus \sigma_2$ and (h) $\langle T_1, \sigma_1 \rangle \longmapsto \langle T_1', \sigma_1' \rangle$. From (c), and (f), using Remark 3.17, we have 5 cases:

    * If both T_1 and T_1' are 1-atomic, and T_2 is 0-atomic, we have to show that $I \models \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \sigma_1' \uplus \sigma_2 \rangle \rhd_{n-1} (Q_1 {*} Q_2)$. From (a) and (h), by Def. 6.3 (item 4.a), we know that (i) $I \models \langle T_1', \sigma_1' \rangle \rhd_{n-1} Q_1$. From (b), using Lemma 6.4, we know that (j) $I \models \langle T_2, \sigma_2 \rangle \rhd_{n-1} Q_2$. From (i) and (j), using the induction hypothesis, we conclude

    * If both T_1 and T_1' are 0-atomic, and T_2 is 1-atomic, we have to show that $I \models \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \sigma_1' \uplus \sigma_2 \rangle \rhd_{n-1} (Q_1 * Q_2)$. From (a) and (h), by Def. 6.3 (item 4.b), we know that (i) $\mathsf{dom}(\sigma_1) = \mathsf{dom}(\sigma_1')$ and (j) $I \models \langle T_1', \sigma_1' \rangle \rhd_{n-1} Q_1$. From (b), using Lemma 6.4, we know that (k) $I \models \langle T_2, \sigma_2 \rangle \rhd_{n-1} Q_2$. From (j) and (k), using the induction hypothesis, we conclude

    * If all T_1, T_1', and T_2 are 0-atomic, we have to show that $\mathsf{dom}(\sigma_1 \uplus \sigma_2) = \mathsf{dom}(\sigma_1' \uplus \sigma_2)$ and $I \models \langle \langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \sigma_1' \uplus \sigma_2 \rangle \rhd_{n-1} (Q_1 * Q_2)$. From (a) and (h), by Def. 6.3 (item 4.b), we know that (i) $\mathsf{dom}(\sigma_1) = \mathsf{dom}(\sigma_1')$ and (j) $I \models \langle T_1', \sigma_1' \rangle \rhd_{n-1} Q_1$. From (b), using Lemma 6.4, we know that (k) $I \models \langle T_2, \sigma_2 \rangle \rhd_{n-1} Q_2$. From (j) and (k), using the induction hypothesis, we

know (l) $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathrm{p}}\mathbf{skip}, \sigma_1' \uplus \sigma_2\rangle \rhd_{n-1} (Q_1{*}Q_2)$. From (i), we know that (m) $\mathsf{dom}(\sigma_1 \uplus \sigma_2) = \mathsf{dom}(\sigma_1' \uplus \sigma_2)$. From (l) and (m), we conclude

  * If both T_1 and T_2 are 0-atomic, and T_1' is 1-atomic, we have to show that for all σ_1'' and σ_2'', such that (i) $\sigma_2'' = \sigma_1'' \uplus \sigma'$ and (j) $\sigma_1'' \in I$, we have $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathrm{p}}\mathbf{skip}, \sigma_2''\rangle \rhd_{n-1} (Q_1 {*} Q_2)$. From (a), (j) and (h), by Def. 6.3 (item 4.c), we know that (k) $I \models \langle T_1', \sigma_1''{\uplus}\sigma_1'\rangle \rhd_{n-1} Q_1$. From (b), using Lemma 6.4, we know that (l) $I \models \langle T_2, \sigma_2\rangle \rhd_{n-1} Q_2$. From (k) and (l), by the induction hypothesis, we conclude
  * If both T_1' and T_2 are 0-atomic, and T_1 is 1-atomic, we have to show that exists σ_1'' and σ_2'', such that $\sigma' = \sigma_1''{\uplus}\sigma_2''$, $\sigma_1'' \in I$, and $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathrm{p}}\mathbf{skip}, \sigma_2''\rangle \rhd_{n-1} (Q_1{*}Q_2)$. From (a) and (h), by Def. 6.3 (item 4.d), we know there exists σ_1''' and σ_2''' such that (i) $\sigma_1' = \sigma_1''' \uplus \sigma_2'''$, (j) $\sigma_1''' \in I$, and (k) $I \models \langle T_1', \sigma_2'''\rangle \rhd_{n-1} Q_1$. From (b), using Lemma 6.4, we know that (l) $I \models \langle T_2, \sigma_2\rangle \rhd_{n-1} Q_2$. From (k), and (l), by the induction hypothesis, we have (m) $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathrm{p}}\mathbf{skip}, \sigma_2''' \uplus \sigma_2\rangle \rhd_{n-1} (Q_1{*}Q_2)$. Instantiating the goal with σ_1''' and $\sigma_2''' \uplus \sigma_2$, from (j), and (m), we conclude

– or, $T' = \langle\!\langle T_1, T_2'\rangle\!\rangle_{\mathrm{p}}\mathbf{skip}$ and $\langle T_2, \sigma_1 \uplus \sigma_2\rangle \longmapsto \langle T_2', \sigma'\rangle$. The proof is symmetric to the previous case

- $\langle\!\langle T_1, T_2\rangle\!\rangle_{\mathrm{p}}\mathbf{skip} \neq \mathbf{skip}$ □

An atomic block can be introduced using the following lemma.

Lemma 6.11. If $\mathsf{Emp} \models \langle T, \sigma\rangle \rhd_n (I{*}Q)$, then $I \models \langle\langle\!\langle T\rangle\!\rangle_{\mathrm{a}}\mathbf{skip}, \sigma\rangle \rhd_n Q$

Proof. By induction over n. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, assuming (a) $\mathsf{Emp} \models \langle T, \sigma\rangle \rhd_n (I{*}Q)$, we have 5 cases:

- By Def. 3.1, we know that (b) $\langle\!\langle T\rangle\!\rangle_{\mathrm{a}}\mathbf{skip}$ is 1-atomic
- From the semantics, if we assume $\langle\langle\!\langle T\rangle\!\rangle_{\mathrm{a}}\mathbf{skip}, \sigma\rangle \longmapsto \mathsf{abort}$, then we know that (c) $\langle T, \sigma\rangle \longmapsto \mathsf{abort}$. From (a), by Def. 6.3 (item 2), we know that (c) is false

- From the semantics, if we assume $\langle \langle\!\langle T \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \sigma\rangle \longmapsto \textsf{race}$, then we know that (c) $\langle T, \sigma\rangle \longmapsto \textsf{race}$. From (a), by Def. 6.3 (item 3), we know that (c) is false
- If (c) $\langle \langle\!\langle T \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \sigma\rangle \longmapsto \langle T', \sigma'\rangle$, given (b) and from the semantics, we have 2 cases:
  - We have $T' = \langle\!\langle T'' \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}$, which is 1-atomic, (d) $\langle T, \sigma\rangle \longmapsto \langle T'', \sigma'\rangle$, and we need to show that $I \models \langle \langle\!\langle T'' \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \sigma'\rangle \rhd_{n-1} Q$. From (a) and (d), by Def. 6.3 (items 4.a through 4.b), we know that (e) $\textsf{Emp} \models \langle T'', \sigma'\rangle \rhd_{n-1} (I * Q)$. From (e), by the induction hypothesis, we conclude
  - We have (d) $T = \mathbf{skip}$, $T' = \mathbf{skip}$ which is 0-atomic, $\sigma = \sigma'$, and we need to show that exists σ_1 and σ_2, such that $\sigma = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in I$, and $I \models \langle \mathbf{skip}, \sigma_2\rangle \rhd_{n-1} Q$. From (a) and (d), by Def. 6.3 (item 5), we know that (e) $\sigma \in I * Q$. From (e), we know that there exists σ_1' and σ_2' such that (f) $\sigma = \sigma_1' \uplus \sigma_2'$, (g) $\sigma_1' \in I$, and (h) $\sigma_2' \in Q$. From (h), using Lemma 6.6, we know (i) $I \models \langle \mathbf{skip}, \sigma_2'\rangle \rhd_{n-1} Q$. Instantiating the goal with σ_1' and σ_2', from (f), (g), and (h), we conclude
- We know that $\langle\!\langle T \rangle\!\rangle_{\mathsf{a}} \mathbf{skip} \neq \mathbf{skip}$ □

The following lemma is used for framing a triple into a larger shared memory.

Lemma 6.12. If $I \models \langle T, \sigma\rangle \rhd_n Q$, then

1. If T is 0-atomic, then $I' * I \models \langle T, \sigma\rangle \rhd_n Q$
2. If T is 1-atomic, and $\sigma' \in I'$, then $I' * I \models \langle T, \sigma' \uplus \sigma\rangle \rhd_n Q$

Proof. By induction over n. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, assuming (a) $I \models \langle T, \sigma\rangle \rhd_n Q$, we have 2 cases:

- If (b) T is 0-atomic, we need to show $I' * I \models \langle T, \sigma\rangle \rhd_n Q$. By Def. 6.3, we have 5 cases:
  - From (b), we know that T is 0-atomic
  - From (a), by Def. 6.3 (item 2), we know that $\langle T, \sigma\rangle \longmapsto \textsf{abort}$ is false
  - From (a), by Def. 6.3 (item 3), we know that $\langle T, \sigma\rangle \longmapsto \textsf{race}$ is false

  - If (c) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'\rangle$, given (b) the we have 2 cases:
    - If T' is 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $I' * I \models \langle T', \sigma'\rangle \rhd_{n-1} Q$. From (a) and (c), by Def. 6.3 (item 4.b), we know that (d) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and (e) $I \models \langle T', \sigma'\rangle \rhd_{n-1} Q$. From (e), by the induction hypothesis (item 1), we know that (f) $I' * I \models \langle T', \sigma'\rangle \rhd_{n-1} Q$. From (d) and (f) we conclude
    - If T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (d) $\sigma_2 = \sigma_1 \uplus \sigma'$ and (e) $\sigma_1 \in I' * I$, we have $I' * I \models \langle T', \sigma_2\rangle \rhd_{n-1} Q$. From (e), we know exists σ_1' and σ_1'', such that $\sigma_1 = \sigma_1' \uplus \sigma_1''$, (f) $\sigma_1' \in I'$, and (g) $\sigma_1'' \in I$. From (a), (c), and (g), by Def. 6.3 (item 4.c), we know that (h) $I \models \langle T', \sigma_1'' \uplus \sigma'\rangle \rhd_{n-1} Q$. From (h) and (f), by the induction hypothesis (item 2), we conclude
  - We assume (c) $T = \textbf{skip}$. From (a) and (c), by Def. 6.3 (item 5), we know $\sigma \in Q$ and conclude

- If (b) T is 1-atomic, and (c) $\sigma' \in I'$, we need to show $I' * I \models \langle T, \sigma' \uplus \sigma\rangle \rhd_n Q$. By Def. 6.3, we have 5 cases:
  - From (b), we know that T is 1-atomic
  - From (a), by Def. 6.3 (item 2), we know that (d) $\langle T, \sigma\rangle \longmapsto \mathsf{abort}$ is false. From (d), using Lemma 3.22 (item 1), we conclude
  - From (a), by Def. 6.3 (item 3), we know that (e) $\langle T, \sigma\rangle \longmapsto \mathsf{race}$ is false. From (d) and (e), using Lemma 3.22 (item 2), we conclude
  - We know (f) $\langle T, \sigma' \uplus \sigma\rangle \longmapsto \langle T', \sigma''\rangle$. From (d) and (f), using Lemma 3.22 (item 3), we know exists σ''' such that (g) $\sigma'' = \sigma' \uplus \sigma'''$ and (h) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'''\rangle$. Then we have 2 cases:
    - If T' is 1-atomic, we need to show that $I' * I \models \langle T', \sigma' \uplus \sigma'''\rangle \rhd_{n-1} Q$. From (a) and (h), by Def. 6.3 (item 4.a), we know that (i) $I \models \langle T', \sigma'''\rangle \rhd_{n-1} Q$. From

(i) and (c), by the induction hypothesis (item 2), we conclude

  * If T' is 0-atomic, we need to show that exists σ_1 and σ_2, such that $\sigma'' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in I' * I$, and $I' * I \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q$. From (a) and (h), by Def. 6.3 (item 4.d), we know there exists σ'_1 and σ'_2, such that (i) $\sigma''' = \sigma'_1 \uplus \sigma'_2$, (j) $\sigma'_1 \in I$, and (k) $I \models \langle T', \sigma'_2 \rangle \rhd_{n-1} Q$. From (k), by the induction hypothesis (item 1), we have (l) $I' * I \models \langle T', \sigma'_2 \rangle \rhd_{n-1} Q$. From (c), (g), (i), and (j), we have (m) $\sigma' \uplus \sigma'_1 \in I' * I$. Instantiating the goal with $\sigma' \uplus \sigma'_1$ and σ'_2, from (g), (i), (m), and (l), we conclude

- From (b), we know that $T \neq$ **skip** □

The following lemma is used to transfer a resource from shared to private in a triple.

Lemma 6.13. If $I' * I \models \langle T, \sigma \rangle \rhd_n Q$, then

1. If T is 0-atomic, and $\sigma' \in I'$, then $I \models \langle T, \sigma' \uplus \sigma \rangle \rhd_n (I' * Q)$
2. If T is 1-atomic, then $I \models \langle T, \sigma \rangle \rhd_n (I' * Q)$

Proof. By induction over n. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, assuming (a) $I' * I \models \langle T, \sigma \rangle \rhd_n Q$, we have 2 cases:

- If (b) T is 0-atomic, and (c) $\sigma' \in I'$, we need to show $I \models \langle T, \sigma' \uplus \sigma \rangle \rhd_n (I' * Q)$. By Def. 6.3, we have 5 cases:
  - From (b), we know that T is 0-atomic
  - From (a), by Def. 6.3 (item 2), we know that (d) $\langle T, \sigma \rangle \longmapsto \textsf{abort}$ is false. From (d), using Lemma 3.22 (item 1), we conclude
  - From (a), by Def. 6.3 (item 3), we know that (e) $\langle T, \sigma \rangle \longmapsto \textsf{race}$ is false. From (d) and (e), using Lemma 3.22 (item 2), we conclude
  - We know (f) $\langle T, \sigma' \uplus \sigma \rangle \longmapsto \langle T', \sigma'' \rangle$. From (d) and (f), using Lemma 3.22 (item 3), we know exists σ''' such that (g) $\sigma'' = \sigma' \uplus \sigma'''$ and (h) $\langle T, \sigma \rangle \longmapsto \langle T', \sigma''' \rangle$. Then we have 2 cases:

    - If T' is 0-atomic, we need to show that $\mathsf{dom}(\sigma' \uplus \sigma) = \mathsf{dom}(\sigma' \uplus \sigma''')$ and $I \models \langle T', \sigma' \uplus \sigma''' \rangle \rhd_{n-1} (I' {*} Q)$. From (a) and (h), by Def. 6.3 (item 4.b), we know that (i) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma''')$ and (j) $I' {*} I \models \langle T', \sigma''' \rangle \rhd_{n-1} Q$. From (j) and (c), by the induction hypothesis (item 1), we know (k) $I \models \langle T', \sigma' \uplus \sigma''' \rangle \rhd_{n-1} (I' {*} Q)$. From (i) and (k), we conclude
    - If T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (i) $\sigma_2 = \sigma_1 \uplus \sigma' \uplus \sigma'''$ and (j) $\sigma_1 \in I$, we have $I \models \langle T', \sigma_2 \rangle \rhd_{n-1} (I' {*} Q)$. From (c) and (j), we know (k) $\sigma' \uplus \sigma_1 \in I' {*} I$. From (a), (h), (i), and (k), by Def. 6.3 (item 4.c), we know that (l) $I' {*} I \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q$. From (l), by the induction hypothesis (item 2), we conclude
  - We assume (d) $T = \textbf{skip}$. From (a) and (d), by Def. 6.3 (item 5), we know that (e) $\sigma \in Q$. From (c) and (e) we know that $\sigma' \uplus \sigma \in I' {*} Q$ and conclude

- If (b) T is 1-atomic, we need to show $I \models \langle T, \sigma \rangle \rhd_n (I' {*} Q)$. By Def. 6.3, we have 5 cases:
  - From (b), we know that T is 1-atomic
  - From (a), by Def. 6.3 (item 2), we know that $\langle T, \sigma \rangle \longmapsto \mathsf{abort}$ is false
  - From (a), by Def. 6.3 (item 3), we know that $\langle T, \sigma \rangle \longmapsto \mathsf{race}$ is false
  - If (c) $\langle T, \sigma \rangle \longmapsto \langle T', \sigma' \rangle$, given (b) the we have 2 cases:
    - If T' is 1-atomic, we need to show that $I \models \langle T', \sigma' \rangle \rhd_{n-1} (I' {*} Q)$. From (a) and (c), by Def. 6.3 (item 4.a), we know that (d) $I' {*} I \models \langle T', \sigma' \rangle \rhd_{n-1} Q$. From (d), by the induction hypothesis (item 2), we conclude
    - If T' is 0-atomic, we need to show that exists σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in I$, and $I \models \langle T', \sigma_2 \rangle \rhd_{n-1} (I' {*} Q)$. From (a) and (c), by Def. 6.3 (item 4.d), we know there exists σ_1' and σ_2', such that (d) $\sigma' = \sigma_1' \uplus \sigma_2'$, (e) $\sigma_1' \in I' {*} I$, and (f) $I' {*} I \models \langle T', \sigma_2' \rangle \rhd_{n-1} Q$. From (e) we know exists σ_1'' and σ_1''' such that (g) $\sigma_1' = \sigma_1'' \uplus \sigma_1'''$, (h) $\sigma_1'' \in I'$, and (i) $\sigma_1''' \in I$. From (f) and (h), by

the induction hypothesis (item 1), we have (j) $I \models \langle T', \sigma_1'' \uplus \sigma_2' \rangle \rhd_{n-1} (I' * Q)$. Instantiating the goal with σ_1''' and $\sigma_1'' \uplus \sigma_2'$, from (d), (g), (i), and (j), we conclude

– From (b), we know that $T \neq$ **skip** $\square$

The following lemma is used for the conjunction of triples.

Lemma 6.14. If $I \models \langle T, \sigma \rangle \rhd_n Q_1$, $I \models \langle T, \sigma \rangle \rhd_n Q_2$, and I is precise, then $I \models \langle T, \sigma \rangle \rhd_n (Q_1 \cap Q_2)$

Proof. By induction over n. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, assuming (a) $I \models \langle T, \sigma \rangle \rhd_n Q_1$ and (b) $I \models \langle T, \sigma \rangle \rhd_n Q_2$, we have 5 cases:

- From (a), by Def. 6.3 (item 1), we know that T is 0- or 1-atomic
- From (a), by Def. 6.3 (item 2), we know that $\langle T, \sigma \rangle \longmapsto \mathsf{abort}$ is false
- From (a), by Def. 6.3 (item 3), we know that $\langle T, \sigma \rangle \longmapsto \mathsf{race}$ is false
- If (c) $\langle T, \sigma \rangle \longmapsto \langle T', \sigma' \rangle$, then we have 4 cases:
  – If both T and T' are 1-atomic, we need to show that $I \models \langle T', \sigma' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (c), by Def. 6.3 (item 4.a), we know that (d) $I \models \langle T', \sigma' \rangle \rhd_{n-1} Q_1$. From (b) and (c), by Def. 6.3 (item 4.a), we know that (e) $I \models \langle T', \sigma' \rangle \rhd_{n-1} Q_2$. From (d) and (e), by the induction hypothesis, we conclude

  – If both T and T' are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $I \models \langle T', \sigma' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (c), by Def. 6.3 (item 4.b), we know already that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, and also that (d) $I \models \langle T', \sigma' \rangle \rhd_{n-1} Q_1$. From (b) and (c), by Def. 6.3 (item 4.b), we also know that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and (e) $I \models \langle T', \sigma' \rangle \rhd_{n-1} Q_2$. From (d) and (e), by the induction hypothesis, we conclude

  - If T is 0-atomic and T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (d) $\sigma_2 = \sigma_1 \uplus \sigma'$ and (e) $\sigma_1 \in I$, we have $I \models \langle T', \sigma_2 \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a), (c), (d), and (e), by Def. 6.3 (item 4.c), we know that (f) $I \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q_1$. From (b), (c), (d), and (e), by Def. 6.3 (item 4.c), we know that (g) $I \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q_2$. From (f) and (g), by the induction hypothesis, we conclude
  - If T is 1-atomic and T' is 0-atomic, we need to show that exists σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in I$, and $I \models \langle T', \sigma_2 \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (c), by Def. 6.3 (item 4.d), we know there exists σ_1' and σ_2', such that (d) $\sigma' = \sigma_1' \uplus \sigma_2'$, (e) $\sigma_1' \in I$, and (f) $I \models \langle T', \sigma_2' \rangle \rhd_{n-1} Q_1$. From (b) and (c), by Def. 6.3 (item 4.d), we know there exists σ_1'' and σ_2'', such that (g) $\sigma' = \sigma_1'' \uplus \sigma_2''$, (h) $\sigma_1'' \in I$, and (i) $I \models \langle T', \sigma_2'' \rangle \rhd_{n-1} Q_2$. From (d), (e), (g), and (h), given that I is precise, we know that $\sigma_1' = \sigma_1''$ and $\sigma_2' = \sigma_2''$. From (f) and (i), by the induction hypothesis, we have (j) $I \models \langle T', \sigma_2' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. Instantiating the goal with σ_1' and σ_2', from (d), (e), and (j), we conclude

- We assume (c) $T = \textbf{skip}$. From (a) and (c), by Def. 6.3 (item 5), we know that (d) $\sigma \in Q_1$. From (b) and (c), by Def. 6.3 (item 5), we know that (e) $\sigma \in Q_2$. From (d) and (e), we know that $\sigma \in Q_1 \cap Q_2$ and conclude □

Semantics rules. The quadruple $I \models \{P\}\, c\, \{Q\}$ is the semantic correspondent to $I \vdash \{P\}\, c\, \{Q\}$. It is defined in terms of $I \models \langle T, \sigma \rangle \rhd Q$ as show below:

Definition 6.15. $I \models \{P\}\, c\, \{Q\}$, if and only if, for all σ, such that $\sigma \in P$, we have $I \models \langle c, \sigma \rangle \rhd Q$

From Def. 6.15, we can prove Lemma 6.16 which states more explicitly the properties guaranteed by the semantic quadruple: safety, race-freedom, and partial correctness (items 1, 2, and 3, respectively).

Lemma 6.16. If $I \models \{P\}\, c\, \{Q\}$, then for all σ, such that $\sigma \in I * P$, we have:

1. $\neg\langle c, \sigma\rangle \longmapsto^* \mathsf{abort}$

2. $\neg\langle c, \sigma\rangle \longmapsto^* \mathsf{race}$

3. If $\langle c, \sigma\rangle \longmapsto^* \langle \mathbf{skip}, \sigma'\rangle$, then $\sigma' \in I * Q$

Proof. From $I \models \{P\}\, c\, \{Q\}$, using Lemma 6.28[1], we obtain (a) $\mathsf{Emp} \models \{I{*}P\}\, c\, \{I{*}Q\}$. Given (a), and $\sigma \in I * P$, from Def. 6.15, we obtain (b) $\mathsf{Emp} \models \langle c, \sigma\rangle \rhd (I * Q)$. We then generalize the proof from command c to any 0- or 1-atomic thread tree T. Now we can consider each one of the goals:

- For goal 1, we need to show that for all n, (c) $\langle T, \sigma\rangle \longmapsto^n \mathsf{abort}$ is false. From (b), we obtain (d) $\mathsf{Emp} \models \langle T, \sigma\rangle \rhd_n (I * Q)$, which is our sole assumption. By induction over n, we have two cases. The base case, where $n = 0$, is trivial as $\langle T, \sigma\rangle \neq \mathsf{abort}$. In the inductive case, where $n > 0$, we know there exists a configuration κ, such that (e) $\langle T, \sigma\rangle \longmapsto \kappa$ and (f) $\kappa \longmapsto^{n-1} \mathsf{abort}$. From (d), given that $n > 0$, we know, from items 2 and 3 of Def. 6.3, that by exclusion there must exists T' and σ', such that $\kappa = \langle T', \sigma'\rangle$; therefore we obtain (g) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'\rangle$. From (g), using Remark 3.17, we know that T' is either 0- or 1-atomic. Given (g), and (d), from items 4.a through 4.d of Def. 6.3, we know that (h) $\mathsf{Emp} \models \langle T', \sigma'\rangle \rhd_{n-1} (I * Q)$. From (h), and (f), using the induction hypothesis, we know that (i) $\langle T', \sigma'\rangle \longmapsto^{n-1} \mathsf{abort}$ is false. From (g), and (i), we know $\langle T, \sigma\rangle \longmapsto^{n-1} \mathsf{abort}$ is false, which was our goal.

- For goal 2, we need to show that for all n, (c) $\langle T, \sigma\rangle \longmapsto^n \mathsf{race}$ is false. From (b), we obtain (d) $\mathsf{Emp} \models \langle T, \sigma\rangle \rhd_n (I * Q)$, which is our sole assumption. By induction over n, we have two cases. The base case, where $n = 0$, is trivial as $\langle T, \sigma\rangle \neq \mathsf{race}$. In the inductive case, where $n > 0$, we know there exists a configuration κ, such that (e) $\langle T, \sigma\rangle \longmapsto \kappa$ and (f) $\kappa \longmapsto^{n-1} \mathsf{race}$. From (d), given that $n > 0$, we know, from items 2 and 3 of Def. 6.3, that by exclusion there must exists T' and σ', such that $\kappa = \langle T', \sigma'\rangle$; therefore we obtain (g) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'\rangle$. From (g), using Remark 3.17,

[1]Although Lemma 6.28 is defined later on the text, there is no circularity.

we know that T' is either 0- or 1-atomic. Given (g), and (d), from items 4.a through 4.d of Def. 6.3, we know that (h) $\mathsf{Emp} \models \langle T', \sigma' \rangle \rhd_{n-1} (I * Q)$. From (h), and (f), using the induction hypothesis, we know that (i) $\langle T', \sigma' \rangle \longmapsto^{n-1} \mathsf{race}$ is false. From (g), and (i), we know $\langle T, \sigma \rangle \longmapsto^{n-1} \mathsf{race}$ is false, which was our goal.

- For goal 3, we need to show that for all n, if (c) $\langle T, \sigma \rangle \longmapsto^{n} \langle \mathbf{skip}, \sigma' \rangle$, then $\sigma' \in I * Q$. From (b), we obtain (d) $\mathsf{Emp} \models \langle T, \sigma \rangle \rhd_n (I * Q)$, which is our sole assumption. By induction over n, we have two cases. In base case, where $n = 0$, we know that $T = \mathbf{skip}$ and $\sigma' = \sigma$; given (d), from item 5 of Def. 6.3, we obtain the goal $\sigma' \in I * Q$. In the inductive case, where $n > 0$, we know there exists a configuration κ, such that (e) $\langle T, \sigma \rangle \longmapsto \kappa$ and (f) $\kappa \longmapsto^{n-1} \langle \mathbf{skip}, \sigma' \rangle$. From (d), given that $n > 0$, we know, from items 2 and 3 of Def. 6.3, that by exclusion there must exists T' and σ'', such that $\kappa = \langle T', \sigma'' \rangle$; therefore we obtain (g) $\langle T, \sigma \rangle \longmapsto \langle T', \sigma'' \rangle$. From (g), using Remark 3.17, we know that T' is either 0- or 1-atomic. Given (g), and (d), from items 4.a through 4.d of Def. 6.3, we know that (h) $\mathsf{Emp} \models \langle T', \sigma'' \rangle \rhd_{n-1} (I * Q)$. From (h), and (f), using the induction hypothesis, we obtain the goal $\sigma' \in I * Q$. $\square$

In the following sequence of lemmas, Lemma 6.17 through Lemma 6.30, we show the correspondence between the CSL rules from Fig. 6.3, and their semantic equivalent using definition Def. 6.15.

Lemma 6.17.

$$\frac{}{\mathsf{Emp} \models \{Q \circ [\![\nu := e]\!]\}\ \nu := e\ \{Q\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in Q \circ [\![\nu := e]\!]$, and we need to show that $\mathsf{Emp} \models \langle \nu := e, \sigma \rangle \rhd_n Q$. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, we have 5 cases:

- By Def. 3.1, we know that (b) $\nu := e$ is 0-atomic

- From the semantics, we know that $\langle \nu := e, \sigma \rangle \longmapsto \mathsf{abort}$ is false if $\langle \nu := e, \sigma \rangle \longrightarrow \mathsf{abort}$

is false. From the sequential semantics, we know $\langle \nu := e, \sigma\rangle \longrightarrow \mathsf{abort}$ is false if there exists σ' such that $(\sigma, \sigma') \in [\![a]\!]$. From (a), by Def. 6.2 (item 1), we conclude

- From the semantics, we know that $\langle \nu := e, \sigma\rangle \longmapsto \mathsf{race}$ is false
- From the semantics, we know that $\langle \nu := e, \sigma\rangle \longmapsto \langle \mathbf{skip}, \sigma'\rangle$, where (c) $\langle \nu := e, \sigma\rangle \longrightarrow \langle \mathbf{skip}, \sigma'\rangle$. From (b), and since **skip** is 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $\mathsf{Emp} \models \langle \mathbf{skip}, \sigma'\rangle \rhd_{n-1} Q$. From the sequential semantics, and (c), we know (d) $(\sigma, \sigma') \in [\![\nu := e]\!]$. From (d), using Remark 3.13, we know (e) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$. From (a) and (d), by Def. 6.2 (item 2), we know that (f) $\sigma' \in Q$. From (f), and Lemma 6.6, we know (g) $\mathsf{Emp} \models \langle \mathbf{skip}, \sigma'\rangle \rhd_{n-1} Q$. From (e) and (g) we conclude
- We know that $(\nu := e) \neq \mathbf{skip}$ □

Lemma 6.18.

$$\frac{}{\mathsf{Emp} \models \{Q \circ [\![a]\!]\}\ \langle a\rangle\ \{Q\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in Q \circ [\![a]\!]$, and we need to show that $\mathsf{Emp} \models \langle \mathbf{atomic}\ a, \sigma\rangle \rhd_n Q$. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, we have 5 cases:

- By Def. 3.1, we know that (b) **atomic** a is 0-atomic
- From the semantics, we know that $\langle \mathbf{atomic}\ a, \sigma\rangle \longmapsto \mathsf{abort}$ is false
- From the semantics, we know that $\langle \mathbf{atomic}\ a, \sigma\rangle \longmapsto \mathsf{race}$ is false
- From the semantics, we know that $\langle \mathbf{atomic}\ a, \sigma\rangle \longmapsto \langle \langle\!\langle a\rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \sigma\rangle$, given that $\bullet$ is 0-atomic. From (b), and since $\langle\!\langle a\rangle\!\rangle_{\mathsf{a}} \mathbf{skip}$ is 1-atomic, we need to show that for all σ_1 and σ_2, such that (c) $\sigma_2 = \sigma_1 \uplus \sigma$ and (d) $\sigma_1 \in \mathsf{Emp}$, $\mathsf{Emp} \models \langle \langle\!\langle a\rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \sigma_2\rangle \rhd_{n-1} Q$. From (d), we know $\sigma_1 = \emptyset$, therefore, from (c), we know $\sigma_2 = \sigma$. If $n = 1$, by Def. 6.3, we conclude. If $n > 1$, by Def. 6.3, we have 5 cases:
  - By Def. 3.1, we know that (e) $\langle\!\langle a\rangle\!\rangle_{\mathsf{a}} \mathbf{skip}$ is 1-atomic

- From the semantics, we know that $\langle\langle\!\langle a\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma\rangle \longmapsto \mathsf{abort}$ is false if $\langle\langle\!\langle a\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma\rangle \longrightarrow \mathsf{abort}$ is false. From the sequential semantics, we know $\langle\langle\!\langle a\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma\rangle \longrightarrow \mathsf{abort}$ is false if there exists σ' such that $(\sigma, \sigma') \in [\![a]\!]$. From (a), by Def. 6.2 (item 1), we conclude
- From the semantics, we know that $\langle\langle\!\langle a\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma\rangle \longmapsto \mathsf{race}$ is false
- From the semantics, we know that $\langle\langle\!\langle a\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma\rangle \longmapsto \langle\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma'\rangle$, where (f) $\langle a, \sigma\rangle \longrightarrow \langle\mathbf{skip}, \sigma'\rangle$. From (e), and since $\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}$ is 1-atomic, we need to show that $\mathsf{Emp} \models \langle\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma'\rangle \rhd_{(n-2)} Q$. If $n = 2$, by Def. 6.3, we conclude. If $n > 2$, by Def. 6.3, we have 5 cases:
  - By Def. 3.1, we know that (g) $\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}$ is 1-atomic
  - From the semantics, we know that $\langle\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma'\rangle \longmapsto \mathsf{abort}$ is false
  - From the semantics, we know that $\langle\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma'\rangle \longmapsto \mathsf{race}$ is false
  - From the semantics, we know that $\langle\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma'\rangle \longmapsto \langle\mathbf{skip}, \sigma'\rangle$. From (g), and since $\mathbf{skip}$ is 0-atomic, we need to show there exists σ'_1 and σ'_2 such that $\sigma' = \sigma'_1 \uplus \sigma'_2$, $\sigma'_1 \in \mathsf{Emp}$, and $\mathsf{Emp} \models \langle\mathbf{skip}, \sigma'_2\rangle \rhd_{(n-3)} Q$. We instantiate σ'_1 as $\emptyset$ and σ'_2 as σ', as we know that $\sigma' = \emptyset \uplus \sigma'$ and $\emptyset \in \mathsf{Emp}$; it remains to show that $\mathsf{Emp} \models \langle\mathbf{skip}, \sigma'\rangle \rhd_{(n-3)} Q$. From the sequential semantics, and (f), we know (h) $(\sigma, \sigma') \in [\![a]\!]$. From (a) and (h), by Def. 6.2 (item 2), we know that (i) $\sigma' \in Q$. From (i), and Lemma 6.6, we conclude
  - We know that $\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip} \neq \mathbf{skip}$
- We know that $\langle\!\langle a\rangle\!\rangle_{\mathsf{a}}\mathbf{skip} \neq \mathbf{skip}$

• We know that **atomic** $a \neq$ **skip** □

Lemma 6.19.

$$\frac{I \models \{P\}\, c_1\, \{P'\} \quad I \models \{P'\}\, c_2\, \{Q\}}{I \models \{P\}\, c_1; c_2\, \{Q\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $I \models \langle c_1; c_2, \sigma \rangle \rhd_n Q$. From (a), and $I \models \{P\}\, c_1\, \{P'\}$, by Def. 6.15, we know that (b) $I \models \langle c_1, \sigma \rangle \rhd_n P'$. From $I \models \{P'\}\, c_2\, \{Q\}$, by Def. 6.15, we know that (c) for all $\sigma' \in P'$, we have $I \models \langle c_2, \sigma' \rangle \rhd_n Q$. From (b) and (c), using Lemma 6.7 (item 1), we conclude □

Lemma 6.20.

$$\frac{}{\mathsf{Emp} \models \{P\}\, \textbf{skip}\, \{P\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $\mathsf{Emp} \models \langle \textbf{skip}, \sigma \rangle \rhd_n P$. From (a), using Lemma 6.6, we conclude □

Lemma 6.21.

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad I \models \{P \cap \lfloor b \rfloor\}\, c_1\, \{Q\} \quad I \models \{P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q\}}{I \models \{P\}\, \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\, \{Q\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $I \models \langle \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2, \sigma \rangle \rhd_n Q$. From (a), and $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, we know (b) $\sigma \in \lfloor b \rfloor \cup \lfloor \neg b \rfloor$. From (b) we can consider two cases:

- If (c) $\sigma \in \lfloor b \rfloor$, with (a), we know (d) $\sigma \in P \cap \lfloor b \rfloor$. From (d), and $I \models \{P \cap \lfloor b \rfloor\}\, c_1\, \{Q\}$, by Def. 6.15, we know that (e) $I \models \langle c_1, \sigma \rangle \rhd_n Q$. From (e) and (c), using Lemma 6.8 (item 1), we conclude

- If (c) $\sigma \in \lfloor \neg b \rfloor$, with (a), we know (d) $\sigma \in P \cap \lfloor \neg b \rfloor$. From (d), and $I \models \{P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q\}$, by Def. 6.15, we know that (e) $I \models \langle c_2, \sigma \rangle \rhd_n Q$. From (e) and (c), using Lemma 6.8 (item 2), we conclude □

Lemma 6.22.

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad I \models \{P \cap \lfloor b \rfloor\}\, c\, \{P\}}{I \models \{P\}\, \textbf{while}\ b\ \textbf{do}\ c\, \{P \cap \lfloor \neg b \rfloor\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $I \models \langle \textbf{while}\ b\ \textbf{do}\ c, \sigma \rangle \rhd_n (P \cap \lfloor \neg b \rfloor)$. From $I \models \{P \cap \lfloor b \rfloor\}\, c\, \{P\}$, by Def. 6.15, we know that (b) for all $\sigma' \in P \cap \lfloor b \rfloor$, we have $I \models \langle c, \sigma' \rangle \triangleright_n P$. From (a), (b), and $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, using Lemma 6.9, we conclude □

Lemma 6.23.

$$\frac{I \models \{P_1\}\, c_1\, \{Q_1\} \quad I \models \{P_2\}\, c_2\, \{Q_2\}}{I \models \{P_1 * P_2\}\, c_1 \parallel c_2\, \{Q_1 * Q_2\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in P_1 * P_2$, and we need to show that $I \models \langle c_1 \parallel c_2, \sigma \rangle \triangleright_n (Q_1 * Q_2)$. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, we have 5 cases:

- By Def. 3.1, we know that (b) $c_1 \parallel c_2$ is 0-atomic
- From the semantics, we know that $\langle c_1 \parallel c_2, \sigma \rangle \longmapsto \textsf{abort}$ is false
- From the semantics, we know that $\langle c_1 \parallel c_2, \sigma \rangle \longmapsto \textsf{race}$ is false
- From the semantics, we know that $\langle c_1 \parallel c_2, \sigma \rangle \longmapsto \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \sigma \rangle$. By Def. 3.1, we know (c) $\langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}$ is 0-atomic. From (b) and (c), we need to show that $\sigma = \sigma$, which is trivial, and that $I \models \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \sigma \rangle \triangleright_{n-1} (Q_1 * Q_2)$. From (a), we know there exists σ_1 and σ_2, such that $\sigma = \sigma_1 \uplus \sigma_2$, (d) $\sigma_1 \in P_1$, and (e) $\sigma_2 \in P_2$. From (d), and $I \models \{P_1\}\, c_1\, \{Q_1\}$, by Def. 6.15, we know that (f) $I \models \langle c_1, \sigma_1 \rangle \triangleright_{n-1} Q_1$. From (e), and $I \models \{P_2\}\, c_2\, \{Q_2\}$, by Def. 6.15, we know that (g) $I \models \langle c_2, \sigma_2 \rangle \triangleright_{n-1} Q_2$. From (f), (g), and (c), using Lemma 6.10, we conclude
- We know that $c_1 \parallel c_2 \neq \mathbf{skip}$ □

Lemma 6.24.

$$\frac{\textsf{Emp} \models \{I * P\}\, c\, \{I * Q\}}{I \models \{P\}\, \mathbf{atomic}\; c\, \{Q\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $I \models \langle \mathbf{atomic}\; c, \sigma \rangle \triangleright_n Q$. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, we have 5 cases:

- By Def. 3.1, we know that (b) **atomic** c is 0-atomic
- From the semantics, we know that $\langle \textbf{atomic}\ c, \sigma\rangle \longmapsto \textsf{abort}$ is false
- From the semantics, we know that $\langle \textbf{atomic}\ c, \sigma\rangle \longmapsto \textsf{race}$ is false
- From the semantics, we know that $\langle \textbf{atomic}\ c, \sigma\rangle \longmapsto \langle \langle\!\langle c\rangle\!\rangle_{\textsf{a}}\textbf{skip}, \sigma\rangle$, given that $\bullet$ is 0-atomic. From (b), and since $\langle\!\langle c\rangle\!\rangle_{\textsf{a}}\textbf{skip}$ is 1-atomic, we need to show that for all σ_1 and σ_2, such that (c) $\sigma_2 = \sigma_1 \uplus \sigma$ and (d) $\sigma_1 \in I$, $I \models \langle \langle\!\langle c\rangle\!\rangle_{\textsf{a}}\textbf{skip}, \sigma_2\rangle \rhd_{n-1} Q$. From (a), (c), and (d), we know (e) $\sigma_2 \in I * P$. From (e), and $\textsf{Emp} \models \{I * P\}\, c\, \{I * Q\}$, by Def. 6.15, we know that (f) $\textsf{Emp} \models \langle c, \sigma_2\rangle \rhd_{n-1} (I * Q)$. From (f), using Lemma 6.11, we conclude
- We know that **atomic** $c \neq$ **skip** $\square$

Lemma 6.25.

$$\frac{P \subseteq P' \quad I \models \{P'\}\, c\, \{Q'\} \quad Q' \subseteq Q}{I \models \{P\}\, c\, \{Q\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $I \models \langle c, \sigma\rangle \rhd_n Q$. From (a), and $P \subseteq P'$, we get (b) $\sigma \in P'$. From (b), and $I \models \{P'\}\, c\, \{Q'\}$, by Def. 6.15, we know that (c) $I \models \langle c, \sigma\rangle \rhd_n Q'$. From (c), and $Q' \subseteq Q$, using Lemma 6.5, we conclude $\square$

Lemma 6.26.

$$\frac{\forall x.\ I \models \{\mathcal{P}(x)\}\, c\, \{\mathcal{Q}(x)\}}{I \models \{\exists x.\ \mathcal{P}(x)\}\, c\, \{\exists x.\ \mathcal{Q}(x)\}}$$

Proof. From Def. 6.15, we assume there exist σ, n, and x, such that (a) $\sigma \in \mathcal{P}(x)$, and we need to show that $I \models \langle c, \sigma\rangle \rhd_n (\exists x.\ \mathcal{Q}(x))$. From (a), and $\forall x.\ I \models \{\mathcal{P}(x)\}\, c\, \{\mathcal{Q}(x)\}$, by Def. 6.15, we know that (b) $I \models \langle c, \sigma\rangle \rhd_n (\mathcal{Q}(x))$. We also know that (c) $(\mathcal{Q}(x)) \subseteq (\exists x.\ \mathcal{Q}(x))$. From (b) and (c), using Lemma 6.5, we conclude $\square$

Lemma 6.27.

$$\frac{I \models \{P\}\, c\, \{Q\}}{I' {*} I \models \{P\}\, c\, \{Q\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $I' {*} I \models \langle c, \sigma\rangle \rhd_n Q$. From (a), and $I \models \{P\}\, c\, \{Q\}$, by Def. 6.15, we know that (b) $I \models \langle c, \sigma\rangle \rhd_n Q$. We also know that (c) c is 0-atomic. From (b), and (c), using Lemma 6.12 (item 1), we conclude □

Lemma 6.28.

$$\frac{I' {*} I \models \{P\}\, c\, \{Q\}}{I \models \{I' {*} P\}\, c\, \{I' {*} Q\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in I' {*} P$, and we need to show that $I \models \langle c, \sigma\rangle \rhd_n (I' {*} Q)$. From (a) we know that there exists σ_1 and σ_2 such that $\sigma = \sigma_1 \uplus \sigma_2$, (b) $\sigma_1 \in I'$, and (c) $\sigma_2 \in P$. From (c), and $I' {*} I \models \{P\}\, c\, \{Q\}$, by Def. 6.15, we know that (d) $I' {*} I \models \langle c, \sigma_2\rangle \rhd_n Q$. We also know that (e) c is 0-atomic. From (d), (e), and (b), using Lemma 6.13 (item 1), we conclude □

Lemma 6.29.

$$\frac{I \models \{P_1\}\, c\, \{Q_1\} \quad I \models \{P_2\}\, c\, \{Q_2\} \quad I \text{ is precise}}{I \models \{P_1 \cap P_2\}\, c\, \{Q_1 \cap Q_2\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in P_1 \cap P_2$, and we need to show that $I \models \langle c, \sigma\rangle \rhd_n (Q_1 \cap Q_2)$. From (a) we know that (b) $\sigma \in P_1$ and (c) $\sigma \in P_2$. From (b), and $I \models \{P_1\}\, c\, \{Q_1\}$, by Def. 6.15, we know that (d) $I \models \langle c, \sigma\rangle \rhd_n Q_1$. From (c), and $I \models \{P_2\}\, c\, \{Q_2\}$, by Def. 6.15, we know that (e) $I \models \langle c, \sigma\rangle \rhd_n Q_2$. From (d), (e), and knowing that I is precise, using Lemma 6.14, we conclude □

Lemma 6.30.

$$\frac{I \models \{P_1\}\, c\, \{Q_1\} \quad I \models \{P_2\}\, c\, \{Q_2\}}{I \models \{P_1 \cup P_2\}\, c\, \{Q_1 \cup Q_2\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in P_1 \cup P_2$, and we need to show that $I \models \langle c, \sigma \rangle \rhd_n (Q_1 \cup Q_2)$. From (a) we know that either (b) $\sigma \in P_1$ or (c) $\sigma \in P_2$. If we assume (b), then from $I \models \{P_1\}\, c\, \{Q_1\}$, by Def. 6.15, we know that (d) $I \models \langle c, \sigma \rangle \rhd_n Q_1$. We also know that (e) $Q_1 \subseteq Q_1 \cup Q_2$. From (d) and (e), using Lemma 6.5, we conclude. Similarly, if we assume (c), then from $I \models \{P_2\}\, c\, \{Q_2\}$, by Def. 6.15, we know that (f) $I \models \langle c, \sigma \rangle \rhd_n Q_2$. We also know that (g) $Q_2 \subseteq Q_1 \cup Q_2$. From (f) and (g), using Lemma 6.5, we conclude □

Soundness theorem. The proof structure of Theorem 6.31 is similar for all CSL rules. It uses all lemmas from Lemma 6.17 to Lemma 6.30, one for each corresponding CSL rule. The proof structure is modular, if an extra rule is added to Fig. 6.3, we just need to prove an extra lemma for its correspondent semantic rule.

Theorem 6.31. If $I \vdash \{P\}\, c\, \{Q\}$, then $I \models \{P\}\, c\, \{Q\}$

Proof. By strong induction over the derivation tree depth of (a) $I \vdash \{P\}\, c\, \{Q\}$. After inversion of (a), we have one case for each rule:

- ASSIGNMENT: we know $I = \mathsf{Emp}$, $P = Q \circ [\![\nu := e]\!]$, and $c = (\nu := e)$, using Lemma 6.17, we conclude
- ACTION: we know $I = \mathsf{Emp}$, $P = Q \circ [\![a]\!]$, and $c = \langle a \rangle$, using Lemma 6.18, we conclude.
- SEQUENTIAL: we know $c = (c_1; c_2)$ and that there exists P' such that (a) $I \vdash \{P\}\, c_1\, \{P'\}$ and (b) $I \vdash \{P'\}\, c_2\, \{Q\}$. From (a), using the induction hypothesis, we obtain (c) $I \models \{P\}\, c_1\, \{P'\}$. From (b), using the induction hypothesis, we obtain (d) $I \models \{P'\}\, c_2\, \{Q\}$. From (c), and (d), using Lemma 6.19, we conclude
- SKIP: we know $I = \mathsf{Emp}$, $P = Q$, and $c = \textbf{skip}$, using Lemma 6.20, we conclude.
- CONDITIONAL: we know $c = (\textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2)$ and that (a) $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, (b) $I \vdash \{P \cap \lfloor b \rfloor\}\, c_1\, \{Q\}$, and (c) $I \vdash \{P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q\}$. From (b), using the induction

hypothesis, we obtain (d) $I \models \{P \cap \lfloor b \rfloor\}\, c_1\, \{Q\}$. From (c), using the induction hypothesis, we obtain (e) $I \models \{P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q\}$. From (a), (d), and (e), using Lemma 6.21, we conclude

- LOOP: we know $c = (\textbf{while}\ b\ \textbf{do}\ c)$, $Q = (P \cap \lfloor \neg b \rfloor)$, (a) $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, and (b) $I \vdash \{P \cap \lfloor b \rfloor\}\, c\, \{P\}$. From (b), using the induction hypothesis, we obtain (c) $I \models \{P \cap \lfloor b \rfloor\}\, c\, \{P\}$. From (a), and (c), using Lemma 6.22, we conclude

- PARALLEL: we know $P = (P_1 * P_2)$, $c = (c_1 \parallel c_2)$, $Q = (Q_1 * Q_2)$, and that (a) $I \vdash \{P_1\}\, c_1\, \{Q_1\}$ and (b) $I \vdash \{P_2\}\, c_2\, \{Q_2\}$. From (a), using the induction hypothesis, we obtain (c) $I \models \{P_1\}\, c_1\, \{Q_1\}$. From (b), using the induction hypothesis, we obtain (d) $I \models \{P_2\}\, c_2\, \{Q_2\}$. From (c), and (d), using Lemma 6.23, we conclude

- ATOMIC: we know $c = (\textbf{atomic}\ c)$, and (a) $\mathsf{Emp} \vdash \{I * P\}\, c\, \{I * Q\}$. From (a), using the induction hypothesis, we obtain (b) $\mathsf{Emp} \models \{I * P\}\, c\, \{I * Q\}$. From (b), using Lemma 6.24, we conclude

- CONSEQUENCE: we know that there exists P' and Q', such that (a) $P \subseteq P'$, (b) $Q' \subseteq Q$, and (c) $I \vdash \{P'\}\, c\, \{Q'\}$. From (c), using the induction hypothesis, we obtain (d) $I \models \{P'\}\, c\, \{Q'\}$. From (a), (b), and (d), using Lemma 6.25, we conclude

- EXISTENTIAL: we know that $P = (\exists x.\ \mathcal{P}(x))$, $Q = (\exists x.\ \mathcal{Q}(x))$, and (a) $\forall x.\ I \vdash \{\mathcal{P}(x)\}\, c\, \{\mathcal{Q}(x)\}$. From (a), using the induction hypothesis, we obtain (b) $\forall x.\ I \models \{\mathcal{P}(x)\}\, c\, \{\mathcal{Q}(x)\}$. From (b), using Lemma 6.26, we conclude

- FRAME: we know $I = (I'' * I')$, and (a) $I' \vdash \{P\}\, c\, \{Q\}$. From (a), using the induction hypothesis, we obtain (b) $I' \models \{P\}\, c\, \{Q\}$. From (b), using Lemma 6.27, we conclude

- RESOURCE: we know $P = (I' * P')$, $Q = (I' * Q')$, and (a) $I' * I \vdash \{P'\}\, c\, \{Q'\}$. From (a), using the induction hypothesis, we obtain (b) $I' * I \models \{P'\}\, c\, \{Q'\}$. From (b), using Lemma 6.28, we conclude

- CONJUNCTION: we know $P = (P_1 \cap P_2)$, $Q = (Q_1 \cap Q_2)$, and that (a) $I \vdash \{P_1\}\, c\, \{Q_1\}$, (b) $I \vdash \{P_2\}\, c\, \{Q_2\}$, and (c) I is precise. From (a), using the induction hypothesis, we obtain (d) $I \models \{P_1\}\, c\, \{Q_1\}$. From (b), using the induction hypothesis, we obtain (e) $I \models \{P_2\}\, c\, \{Q_2\}$. From (d), (e), and (c), using Lemma 6.29, we conclude

- DISJUNCTION: we know $P = (P_1 \cup P_2)$, $Q = (Q_1 \cup Q_2)$, and that (a) $I \vdash \{P_1\}\, c\, \{Q_1\}$ and (b) $I \vdash \{P_2\}\, c\, \{Q_2\}$. From (a), using the induction hypothesis, we obtain (c) $I \models \{P_1\}\, c\, \{Q_1\}$. From (b), using the induction hypothesis, we obtain (d) $I \models \{P_2\}\, c\, \{Q_2\}$. From (c), and (d), using Lemma 6.30, we conclude $\square$

6.3.2 With Regard to the Parameterized Semantics

In this section, we proof the soundness of CSL with regard to the parameterized semantics of Sec. 3.8. First, we need to define the semantic meaning of a Λ-parameterized CSL quadruple.

Definition 6.32. $I \models_{[\Lambda]} \{P\}\, c\, \{Q\}$, if and only if, for all σ, such that $\sigma \in I * P$, we have:

1. $\neg [\Lambda]\ \langle c, \sigma \rangle \longmapsto^* \mathsf{abort}$
2. If $[\Lambda]\ \langle c, \sigma \rangle \longmapsto^* \langle \mathbf{skip}, \sigma' \rangle$, then $\sigma' \in I * Q$

This definition is straightforward. The $I \models_{[\Lambda]} \{P\}\, c\, \{Q\}$ quadruple ensures that for any state satisfying the pre-condition $I * P$ will not abort, and, if it the execution completes, the final state will satisfy $I * Q$. Given this definition, we can phrase and prove the soundness theorem below:

Theorem 6.33. If $I \vdash \{P\}\, c\, \{Q\}$, and Λ provides the DRF-guarantee, then $I \models_{[\Lambda]} \{P\}\, c\, \{Q\}$

Proof. From $I \vdash \{P\}\, c\, \{Q\}$, using Theorem 6.31, we obtain (a) $I \models \{P\}\, c\, \{Q\}$. From Def. 6.32, we can prove the goal if, by assuming there is a state σ, such that (b) $\sigma \in I * P$, we can establish the following two conditions:

(c) $\neg [\Lambda]\ \langle c, \sigma \rangle \longmapsto^* \mathsf{abort}$

(d) If $[\Lambda]\ \langle c, \sigma \rangle \longmapsto^* \langle \textbf{skip}, \sigma' \rangle$, then $\sigma' \in I * Q$

From (a), and (b), using Lemma 6.16, we know that:

(e) $\neg \langle c, \sigma \rangle \longmapsto^* \textsf{abort}$

(f) $\neg \langle c, \sigma \rangle \longmapsto^* \textsf{race}$

(g) If $\langle c, \sigma \rangle \longmapsto^* \langle \textbf{skip}, \sigma' \rangle$, then $\sigma' \in I * Q$

Since Λ provides the DRF-guarantee, from (e), and (f), based on Def. 5.2, we establish (c) and we know

(h) If $[\Lambda]\ \langle c, \sigma \rangle \longmapsto^* \langle \textbf{skip}, \sigma' \rangle$, then $\langle c, \sigma \rangle \longmapsto^* \langle \textbf{skip}, \sigma' \rangle$

From (h), and (g), we can establish (d) $\square$

6.3.3 With Regard to the Relaxed Semantics

The proof of soundness regarding the $\rightsquigarrow$-parameterized relaxed semantics of Sec. 3.10 is straightforward.

Theorem 6.34. If $I \vdash \{P\}\, c\, \{Q\}$, then $I \models_{[\rightsquigarrow]} \{P\}\, c\, \{Q\}$

Proof. From Theorem 5.12, we know that (a) $\rightsquigarrow$ provides the DRF-guarantee. From $I \vdash \{P\}\, c\, \{Q\}$, and (a), using Theorem 6.33, we prove our goal $I \models_{[\rightsquigarrow]} \{P\}\, c\, \{Q\}$ $\square$

6.4 Extension Rules

In section Sec. 6.2, we have presented the standard CSL rules. Here we present some extensions.

In Fig. 6.3, we have presented the following CONJUNCTION rule

$$\frac{I \vdash \{P_1\}\, c\, \{Q_1\} \quad I \vdash \{P_2\}\, c\, \{Q_2\} \quad I \text{ is precise}}{I \vdash \{P_1 \cap P_2\}\, c\, \{Q_1 \cap Q_2\}}$$

which has the constraint that I is precise. We can generalize this rule to the following

$$\frac{I_1 \vdash \{P_1\}\, c\, \{Q_1\} \quad I_2 \vdash \{P_2\}\, c\, \{Q_2\} \quad I_1 \text{ and } I_2 \text{ coincide}}{I_1 {\cap} I_2 \vdash \{P_1 {\cap} P_2\}\, c\, \{Q_1 {\cap} Q_2\}}$$

which has a generalized notion of precision: *coincidence*.

Definition 6.35. Two assertions P_1 and P_2 coincide if, and only if, for all σ, $\sigma_1 \subseteq \sigma$, and $\sigma_2 \subseteq \sigma$, such that $\sigma_1 \in P_1$ and $\sigma_2 \in P_2$, we have $\sigma_1 = \sigma_2$

Comparing the precision, by Def. 6.1, with Def. 6.35, we observe that an assertion is precise then it coincides with itself, an vice-versa.

Remark 6.36. An assertion I is precise if, and only if, I and I coincide

Remark 6.37. If $(P_1 {\cup} P_2)$ is precise, then P_1 and P_2 coincide

Remark 6.38. If P_1 and P_2 coincide, then $(P_1 {\cap} P_2)$ is precise

In order to prove the soundness of this generalized rule, we need the following two lemmas:

Lemma 6.39. If $I_1 \models \langle T, \sigma\rangle \rhd_n Q_1$, $I_2 \models \langle T, \sigma\rangle \rhd_n Q_2$, and I_1 and I_2 coincide, then $I_1 {\cap} I_2 \models \langle T, \sigma\rangle \rhd_n (Q_1 {\cap} Q_2)$

Proof. By induction over n. If $n = 0$, by Def. 6.3, we conclude. If $n > 0$, by Def. 6.3, assuming (a) $I_1 \models \langle T, \sigma\rangle \rhd_n Q_1$ and (b) $I_2 \models \langle T, \sigma\rangle \rhd_n Q_2$, we have 5 cases:

- From (a), by Def. 6.3 (item 1), we know that T is 0- or 1-atomic
- From (a), by Def. 6.3 (item 2), we know that $\langle T, \sigma\rangle \longmapsto \textsf{abort}$ is false
- From (a), by Def. 6.3 (item 3), we know that $\langle T, \sigma\rangle \longmapsto \textsf{race}$ is false
- If (c) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'\rangle$, then we have 4 cases:

- If both T and T' are 1-atomic, we need to show that $I_1 \cap I_2 \models \langle T', \sigma' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (c), by Def. 6.3 (item 4.a), we know that (d) $I_1 \models \langle T', \sigma' \rangle \rhd_{n-1} Q_1$. From (b) and (c), by Def. 6.3 (item 4.a), we know that (e) $I_2 \models \langle T', \sigma' \rangle \rhd_{n-1} Q_2$. From (d) and (e), by the induction hypothesis, we conclude
- If both T and T' are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $I_1 \cap I_2 \models \langle T', \sigma' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (c), by Def. 6.3 (item 4.b), we know already that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, and also that (d) $I_1 \models \langle T', \sigma' \rangle \rhd_{n-1} Q_1$. From (b) and (c), by Def. 6.3 (item 4.b), we also know that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and (e) $I_2 \models \langle T', \sigma' \rangle \rhd_{n-1} Q_2$. From (d) and (e), by the induction hypothesis, we conclude
- If T is 0-atomic and T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (d) $\sigma_2 = \sigma_1 \uplus \sigma'$ and (e) $\sigma_1 \in I_1 \cap I_2$, we have $I_1 \cap I_2 \models \langle T', \sigma_2 \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (e), we know that (f) $\sigma_1 \in I_1$ and (g) $\sigma_1 \in I_2$. From (a), (c), (d), and (f), by Def. 6.3 (item 4.c), we know that (h) $I_1 \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q_1$. From (b), (c), (d), and (g), by Def. 6.3 (item 4.c), we know that (i) $I_2 \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q_2$. From (h) and (i), by the induction hypothesis, we conclude
- If T is 1-atomic and T' is 0-atomic, we need to show that exists σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in I_1 \cap I_2$, and $I_1 \cap I_2 \models \langle T', \sigma_2 \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (c), by Def. 6.3 (item 4.d), we know there exists σ_1' and σ_2', such that (d) $\sigma' = \sigma_1' \uplus \sigma_2'$, (e) $\sigma_1' \in I_1$, and (f) $I_1 \models \langle T', \sigma_2' \rangle \rhd_{n-1} Q_1$. From (b) and (c), by Def. 6.3 (item 4.d), we know there exists σ_1'' and σ_2'', such that (g) $\sigma' = \sigma_1'' \uplus \sigma_2''$, (h) $\sigma_1'' \in I_2$, and (i) $I_2 \models \langle T', \sigma_2'' \rangle \rhd_{n-1} Q_2$. From (d), (e), (g), and (h), given that I_1 and I_2 coincide, we know that $\sigma_1' = \sigma_1''$ and $\sigma_2' = \sigma_2''$. From (f) and (i), by the induction hypothesis, we have (j) $I_1 \cap I_2 \models \langle T', \sigma_2' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. Instantiating the goal with σ_1' and σ_2', from (d), (e), and (j), we conclude

• We assume (c) $T = \mathbf{skip}$. From (a) and (c), by Def. 6.3 (item 5), we know that (d) $\sigma \in Q_1$. From (b) and (c), by Def. 6.3 (item 5), we know that (e) $\sigma \in Q_2$. From (d) and (e), we know that $\sigma \in Q_1 \cap Q_2$ and conclude $\square$

Lemma 6.40.

$$\frac{I_1 \models \{P_1\}\, c\, \{Q_1\} \quad I_2 \models \{P_2\}\, c\, \{Q_2\} \quad I_1 \text{ and } I_2 \text{ coincide}}{I_1 \cap I_2 \models \{P_1 \cap P_2\}\, c\, \{Q_1 \cap Q_2\}}$$

Proof. From Def. 6.15, we assume there exist σ and n, such that (a) $\sigma \in P_1 \cap P_2$, and we need to show that $I_1 \cap I_2 \models \langle c, \sigma\rangle \rhd_n (Q_1 \cap Q_2)$. From (a) we know that (b) $\sigma \in P_1$ and (c) $\sigma \in P_2$. From (b), and $I_1 \models \{P_1\}\, c\, \{Q_1\}$, by Def. 6.15, we know that (d) $I_1 \models \langle c, \sigma\rangle \rhd_n Q_1$. From (c), and $I_2 \models \{P_2\}\, c\, \{Q_2\}$, by Def. 6.15, we know that (e) $I_2 \models \langle c, \sigma\rangle \rhd_n Q_2$. From (d), (e), and knowing that I_1 and I_2 coincide, using Lemma 6.39, we conclude $\square$

6.5 Verification Examples

In this section, we verify some sample code using the logic presented in this chapter. Since our language does not support procedures, we use macros to organize the code when necessary, knowing that they are actually inlined. These macros only take program variables as argument. Since the language also does not support local variables, when we write **local** v it means that v is implicitly passed along as macro parameter.

6.5.1 Compare-And-Swap

Our first example of verification using CSL is the *compare-and-swap* (CAS) operation, widely available in hardware to perform atomic changes to memory. The code for CAS is presented in Fig. 6.4.

We verify CAS using the following polymorphic invariant $(\exists v.\, p \mapsto v * \mathcal{I}(v))$ where $\mathcal{I}(v)$ is supposed to describe the contents of the shared memory when the cell at location p has value v. In the proof below, p and $\mathcal{I}$ are free meta-variables.

```
CAS(location, oldvalue, newvalue, result)
  atomic
    if [location] = oldvalue then
      [location] := newvalue;
      result := 1
    else
      result := 0
```

Figure 6.4: Compare-and-swap

```
$\forall p, v_1, v_2, \mathcal{I}.$ CAS(loc, old, new, res)
$(\exists v.\ p \mapsto v * \mathcal{I}(v)) \vdash$
  $\{\texttt{loc} \mapsto p * \texttt{old} \mapsto v_1 * \texttt{new} \mapsto v_2 * \texttt{res} \mapsto \_ * \mathcal{I}(v_2)\}$
  atomic
    $\{(\exists v.\ p \mapsto v * \mathcal{I}(v)) * \texttt{loc} \mapsto p * \texttt{old} \mapsto v_1 * \texttt{new} \mapsto v_2 * \texttt{res} \mapsto \_ * \mathcal{I}(v_2)\}$
    if [loc] = old then
      $\{p \mapsto v_1 * \mathcal{I}(v_1) * \texttt{loc} \mapsto p * \texttt{old} \mapsto v_1 * \texttt{new} \mapsto v_2 * \texttt{res} \mapsto \_ * \mathcal{I}(v_2)\}$
      [loc] := new;
      $\{p \mapsto v_2 * \mathcal{I}(v_1) * \texttt{loc} \mapsto p * \texttt{old} \mapsto v_1 * \texttt{new} \mapsto v_2 * \texttt{res} \mapsto \_ * \mathcal{I}(v_2)\}$
      res := 1
      $\{p \mapsto v_2 * \mathcal{I}(v_1) * \texttt{loc} \mapsto p * \texttt{old} \mapsto v_1 * \texttt{new} \mapsto v_2 * \texttt{res} \mapsto 1 * \mathcal{I}(v_2)\}$
      $\{(\exists v.\ p \mapsto v * \mathcal{I}(v)) * \texttt{loc} \mapsto p * \texttt{old} \mapsto v_1 * \texttt{new} \mapsto v_2 * \texttt{res} \mapsto 1 * \mathcal{I}(v_1)\}$
    else
      res := 0
      $\{(\exists v.\ p \mapsto v * \mathcal{I}(v)) * \texttt{loc} \mapsto p * \texttt{old} \mapsto v_1 * \texttt{new} \mapsto v_2 * \texttt{res} \mapsto 0 * \mathcal{I}(v_2)\}$
  $\{\texttt{loc} \mapsto p * \texttt{old} \mapsto v_1 * \texttt{new} \mapsto v_2 * (\texttt{res} \mapsto 1 * \mathcal{I}(v_1)) \cup (\texttt{res} \mapsto 0 * \mathcal{I}(v_2))\}$
```

Note that this polymorphic proof for CAS can be used whenever the invariant for different values v_1 and v_1 are separated, i.e. $\mathcal{I}(v_1) * \mathcal{I}(v_2)$. Since we use CAS usually to acquire and release memory, one of them is typically Emp.

6.5.2 Single-Entry Mutex

Our next example is a simple, single entry mutex with three operations: lock, trylock, and unlock. The code is shown in Fig. 6.5.

In this implementation, lock busy waits until the mutex is 0, and then sets is to 1, which works as a mutex acquire. Unlock simply releases the mutex by setting it to 0. Trylock simply tests if the acquire is free, if so it sets it to 1, otherwise do nothing. We use CAS to implement the trylock code.

```
LOCK(mutex)                          TRYLOCK(mutex, result)
  atomic                             local t0, t1
    wait [mutex] = 0;                  t0 := 0;
    [mutex] := 1                       t1 := 1;
                                       CAS(mutex, t0, t1, result)
UNLOCK(mutex)
  atomic [mutex] := 0
```

Figure 6.5: Single-entry mutex

In order to write specifications about mutexes, we describe a mutex at location p, protecting shared memory invariant I, as the following definition:

$$\mathsf{mux}(p, I) \overset{\text{def}}{=} (p \mapsto 0 * I) \cup p \mapsto 1$$

Naturally, all three operations assume a mutex is placed in shared memory.

We present below the proof for lock and unlock. Unlock does not test the mutex before proceeding, therefore we require I to be precise in order to verify it. This is because we know that if I is precise, then $I * I$ is true only if, and only if, $I = \mathsf{Emp}$.

```
∀p, I. LOCK(mux)                         ∀p, I. UNLOCK(mux)
mux(p, I) ⊢                              mux(p, I) ⊢
  {mux ↦ p}                                {I * mux ↦ p}
  atomic                                   atomic
    {mux(p, I) * mux ↦ p}                    {mux(p, I) * I * mux ↦ p}
    wait [mux] = 0;                          {((p ↦ 0 * I) ∪ p ↦ 1) * I * mux ↦ p}
    {p ↦ 0 * I * mux ↦ p}                    {((p ↦ 0 * I * I) ∪ (p ↦ 1 * I)) * mux ↦ p}
    [mux] := 1                               {((p ↦ 0 ∩ I = Emp) ∪ (p ↦ 1 * I)) * mux ↦ p}
    {p ↦ 1 * I * mux ↦ p}                    [mux] := 0
    {mux(p, I) * I * mux ↦ p}                {p ↦ 0 * I * mux ↦ p}
  {I * mux ↦ p}                              {mux(p, I) * mux ↦ p}
                                           {mux ↦ p}
```

We also show the verification of trylock based on the already verified code for CAS, from Sec. 6.5.1.

```
$\forall p, I.$ TRYLOCK(mux, res)
**local** t0, t1
mux$(p, I) \vdash$
    $\{$mux$\mapsto p *$res$\mapsto \_ *$t0$\mapsto \_ *$t1$\mapsto \_\}$
    t0$:= 0$;
    $\{$mux$\mapsto p *$res$\mapsto \_ *$t0$\mapsto 0 *$t1$\mapsto \_\}$
    t1$:= 1$;
    $\{$mux$\mapsto p *$res$\mapsto \_ *$t0$\mapsto 0 *$t1$\mapsto 1\}$
    $[p, 0, 1, \lambda v.\ (v = 0 * I) \cup (v = 1 * \mathsf{Emp})]$ CAS(mux, t0, t1, res)
    $\{$mux$\mapsto p * (($res$\mapsto 1 * I) \cup$res$\mapsto 0) *$t0$\mapsto \_ *$t1$\mapsto \_\}$
```

Alternatively, we could also implement the lock operation using CAS, in this case the busy wait loop is outside of the atomic block, as shown below:

```
$\forall p, I.$ LOCK(mux)
**local** t0, t1, res
mux$(p, I) \vdash$
    $\{$mux$\mapsto p *$t0$\mapsto \_ *$t1$\mapsto \_ *$res$\mapsto \_\}$
    t0$:= 0$;
    $\{$mux$\mapsto p *$t0$\mapsto 0 *$t1$\mapsto \_ *$res$\mapsto \_\}$
    t1$:= 1$;
    **repeat**
        $\{$mux$\mapsto p *$t0$\mapsto 0 *$t1$\mapsto 1 *$res$\mapsto \_\}$
        $[p, 0, 1, \lambda v.\ (v = 0 * I) \cup (v = 1 * \mathsf{Emp})]$ CAS(mux, t0, t1, res)
        $\{$mux$\mapsto p *$t0$\mapsto \_ *$t1$\mapsto \_ * (($res$\mapsto 1 * I) \cup$res$\mapsto 0)\}$
    **until** res $= 1$
    $\{$mux$\mapsto p * I *$t0$\mapsto \_ *$t1$\mapsto \_ *$res$\mapsto \_\}$
```

Chapter 7

CSL with Partial Permissions

In this chapter, we present a version of CSL that allows verifying programs with concurrent shared read-only operations. This is done using by assigning permissions to memory locations. These permissions specify whether the location can be written or not. The logic enforces that threads only write to a memory location if it has permission to. Just like standard CSL, the logic will allow adjusting permissions during shared-memory access. It is just a generalization of ownership transfer.

In order to maintain permissions of memory cells, we extend the definition of program states. As show in Fig. 7.1, a state $\bar{\sigma}$ is a map from memory location to a pair of value and permission. We also define Π which is a finite map from memory location to permission. A permission π is fractional, it must be a rational number greater than 0 and less than equal to 1. A permission of value 1 is a full permission. A permission of value less than 1 is a partial permission. A full permission allows read and write accesses. A partial permission only allows read accesses.

$$\begin{array}{rrcl}
(\mathit{PermState}) & \bar{\sigma} & \in & \mathit{Location} \rightharpoonup_{\text{fin}} (\mathit{Integer} \times \mathit{Permission}) \\
(\mathit{Permissions}) & \Pi & \in & \mathit{Location} \rightharpoonup_{\text{fin}} \mathit{Permission} \\
(\mathit{Permission}) & \pi & \in & (0, 1]
\end{array}$$

Figure 7.1: Program state with permissions

We use fractional permissions, instead of simply binary (read or write) permissions, in order to allow a more flexible splitting the state without losing track of permissions. Differently from simply partitioning the state into two sub-states of non-overlapping domains, we allow some overlapping as long as their combined permissions add up to the original permissions. The definition of disjoin union for states with permissions is shown below.

Definition 7.1. The disjoint union $\bar{\sigma} = \bar{\sigma}_1 \uplus \bar{\sigma}_2$, is defined if, and only if, $\mathsf{dom}(\bar{\sigma}_1) \subseteq \mathsf{dom}(\bar{\sigma})$, $\mathsf{dom}(\bar{\sigma}_2) \subseteq \mathsf{dom}(\bar{\sigma})$, and for all $\ell \in \mathsf{dom}(\bar{\sigma})$, we have either

- $\bar{\sigma}(\ell) = \bar{\sigma}_1(\ell)$ and $\ell \notin \mathsf{dom}(\bar{\sigma}_2)$
- or, $\bar{\sigma}(\ell) = \bar{\sigma}_2(\ell)$ and $\ell \notin \mathsf{dom}(\bar{\sigma}_1)$
- or, $\bar{\sigma}_1(\ell) = (i, \pi_1)$, $\bar{\sigma}_2(\ell) = (i, \pi_2)$, and $\bar{\sigma}(\ell) = (i, \pi_1 + \pi_2)$ (by definition $\pi_1 + \pi_2 \in (0, 1]$)

We define $\bar{\sigma}_1 \subseteq \bar{\sigma}_2$ as $\exists \bar{\sigma}'_1.\ \bar{\sigma}_2 = \bar{\sigma}_1 \uplus \bar{\sigma}'_1$

We also define a special operation to map states without permissions to states with permissions, and vice versa. We use Π to record all the locations with partial permissions.

Definition 7.2. The operation $\sigma = \bar{\sigma}|_{\Pi}$, is defined if, and only if, $\mathsf{dom}(\sigma) = \mathsf{dom}(\bar{\sigma})$, $\mathsf{dom}(\Pi) \subseteq \mathsf{dom}(\bar{\sigma})$, and for all $\ell \in \mathsf{dom}(\bar{\sigma})$, we have

- If $\bar{\sigma}(\ell) = (i, 1)$, then $\sigma(\ell) = i$ and $\ell \notin \mathsf{dom}(\Pi)$
- If $\bar{\sigma}(\ell) = (i, \pi)$, where $\pi < 1$, then $\sigma(\ell) = i$ and $\Pi(\ell) = \pi$

We also define a function to extract the largest footprint compatible with a states with permissions.

Definition 7.3. The function $\nabla(\bar{\sigma})$ is defined as $(\mathsf{dom}(\Pi), \mathsf{dom}(\sigma) \backslash \mathsf{dom}(\Pi))$, where $\sigma = \bar{\sigma}|_{\Pi}$

The remaining sections have an structure very similar to Chapter 6, we will point out what are the key differences as they appear.

7.1 Assertion Language

The assertion language for CSL with permissions is similar to the assertion language of CSL (from Sec. 6.1). However, instead of sets of states, we use sets of states with permissions. This is shown in Fig. 7.2.

$$\begin{array}{rrcl} (\mathit{Assertion}) & P, Q, I & \subseteq & \mathit{PermState} \\ (\mathit{AssertionFormula}) & \mathcal{P}, \mathcal{Q} & \subseteq & \alpha \to \mathit{Assertion} \end{array}$$

Figure 7.2: Assertions and assertion formulae

Similarly, in Fig. 7.3, we provide auxiliary definitions for the assertion language. We extend the syntax of $\ell \mapsto i$ to carry the permission for the location $\ell \overset{\pi}{\mapsto} i$. The default permission is 1. Note also that the separating conjunction can be used to split permissions, e.g. $l \mapsto v = (l \overset{.5}{\mapsto} v * l \overset{.5}{\mapsto} v)$.

$$\begin{array}{rcl} \lfloor b \rfloor & \stackrel{\text{def}}{=} & \{\bar{\sigma} \mid [\![b]\!]_{(\bar{\sigma}|_\Pi)} = \mathit{true}\} \\ P_1 * P_2 & \stackrel{\text{def}}{=} & \{\bar{\sigma}_1 \uplus \bar{\sigma}_2 \mid \bar{\sigma}_1 \in P_1 \wedge \bar{\sigma}_2 \in P_2\} \\ \mathsf{Emp} & \stackrel{\text{def}}{=} & \{\emptyset\} \\ \ell \overset{\pi}{\mapsto} i & \stackrel{\text{def}}{=} & \{\{\ell \rightsquigarrow (i, \pi)\}\} \\ \ell \mapsto i & \stackrel{\text{def}}{=} & \ell \overset{1}{\mapsto} i \end{array}$$

Figure 7.3: Auxiliary assertion definitions

We also need to define precision for this assertion language.

Definition 7.4. An assertion P is precise if, and only if, for all $\bar{\sigma}$, $\bar{\sigma}_1 \subseteq \bar{\sigma}$, and $\bar{\sigma}_2 \subseteq \bar{\sigma}$, such that $\bar{\sigma}_1 \in P$ and $\bar{\sigma}_2 \in P$, we have $\bar{\sigma}_1 = \bar{\sigma}_2$

7.2 Inference Rules

The inference rules for CSL with permissions are presented in Fig. 7.4. These rules are exactly the same as the ones presented in Fig. 6.3, however, all definitions are overloaded. As in CSL, the judgment $I \vdash \{P\}\, c\, \{Q\}$ for says that the state can be split implicitly into a shared part and a private part. The private part can be accessed only by c. The shared

$$\frac{}{\mathsf{Emp} \vdash \{Q \circ [\![\nu := e]\!]\}\, \nu := e\, \{Q\}}\ (\textsc{assignment}) \qquad \frac{}{\mathsf{Emp} \vdash \{Q \circ [\![a]\!]\}\, \langle a \rangle\, \{Q\}}\ (\textsc{action})$$

$$\frac{I \vdash \{P\}\, c_1\, \{P'\} \quad I \vdash \{P'\}\, c_2\, \{Q\}}{I \vdash \{P\}\, c_1; c_2\, \{Q\}}\ (\textsc{sequential}) \qquad \frac{}{\mathsf{Emp} \vdash \{P\}\, \textbf{skip}\, \{P\}}\ (\textsc{skip})$$

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad I \vdash \{P \cap \lfloor b \rfloor\}\, c_1\, \{Q\} \quad I \vdash \{P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q\}}{I \vdash \{P\}\, \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\, \{Q\}}\ (\textsc{conditional})$$

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad I \vdash \{P \cap \lfloor b \rfloor\}\, c\, \{P\}}{I \vdash \{P\}\, \textbf{while}\ b\ \textbf{do}\ c\, \{P \cap \lfloor \neg b \rfloor\}}\ (\textsc{loop})$$

$$\frac{I \vdash \{P_1\}\, c_1\, \{Q_1\} \quad I \vdash \{P_2\}\, c_2\, \{Q_2\}}{I \vdash \{P_1 * P_2\}\, c_1 \parallel c_2\, \{Q_1 * Q_2\}}\ (\textsc{parallel}) \qquad \frac{\mathsf{Emp} \vdash \{I * P\}\, c\, \{I * Q\}}{I \vdash \{P\}\, \textbf{atomic}\ c\, \{Q\}}\ (\textsc{atomic})$$

$$\frac{P \subseteq P' \quad I \vdash \{P'\}\, c\, \{Q'\} \quad Q' \subseteq Q}{I \vdash \{P\}\, c\, \{Q\}}\ (\textsc{consequence})$$

$$\frac{\forall x.\ I \vdash \{\mathcal{P}(x)\}\, c\, \{\mathcal{Q}(x)\}}{I \vdash \{\exists x.\ \mathcal{P}(x)\}\, c\, \{\exists x.\ \mathcal{Q}(x)\}}\ (\textsc{existential})$$

$$\frac{I \vdash \{P\}\, c\, \{Q\}}{I' * I \vdash \{P\}\, c\, \{Q\}}\ (\textsc{frame}) \qquad \frac{I' * I \vdash \{P\}\, c\, \{Q\}}{I \vdash \{I' * P\}\, c\, \{I' * Q\}}\ (\textsc{resource})$$

$$\frac{I \vdash \{P_1\}\, c\, \{Q_1\} \quad I \vdash \{P_2\}\, c\, \{Q_2\} \quad I \text{ is precise}}{I \vdash \{P_1 \cap P_2\}\, c\, \{Q_1 \cap Q_2\}}\ (\textsc{conjunction})$$

$$\frac{I \vdash \{P_1\}\, c\, \{Q_1\} \quad I \vdash \{P_2\}\, c\, \{Q_2\}}{I \vdash \{P_1 \cup P_2\}\, c\, \{Q_1 \cup Q_2\}}\ (\textsc{disjunction})$$

Figure 7.4: CSL with partial permissions

part can be accessed by both c and its environment. Accesses to the shared state must preserve its invariant I. Furthermore, since assertions carry permissions, we can have shared read-only memory as part of P's and Q's. The logic will prevent those locations from being modified by either c or its environment. Using the PARALLEL, ATOMIC, and RESOURCE rules, one can transfer manage memory with permissions just like in CSL. The fractions help keeping accurate track of permissions throughout these transfers.

Below, we present an alternate definition of $Q \circ [\![a]\!]$, it checks whether the action a can be executed without violating the permissions (item 1), and if it does, the new state $\bar{\sigma}'$ has the same read permissions as $\bar{\sigma}$, and it must satisfy Q. As one may observe, the dynamic semantics does not know about permissions. Permission accounting is done solely in the logic, statically.

Definition 7.5. $Q \circ [\![a]\!]$ is a set of states with permissions where, for each state $\bar{\sigma}$, we have:

1. $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$

2. for all $\bar{\sigma}'$, such that $(\bar{\sigma}|_{\Pi}, \bar{\sigma}'|_{\Pi}) \in [\![a]\!]$, we have $\bar{\sigma}' \in Q$

Remark 7.6 establishes that Def. 7.5 is not weaker than Def. 6.2.

Remark 7.6. If $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$, then

1. exists σ' such that $(\bar{\sigma}|_{\Pi}, \sigma') \in [\![a]\!]$

2. for all σ', such that $(\bar{\sigma}|_{\Pi}, \sigma') \in [\![a]\!]$, exists $\bar{\sigma}'$ such that $\sigma' = \bar{\sigma}'|_{\Pi}$

Remark 7.7. If $\Delta^{a}_{(\bar{\sigma}_1|_{\Pi_1})} \subseteq \nabla(\bar{\sigma}_1)$, then $\Delta^{a}_{(\bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1 \uplus \Pi_2})} \subseteq \nabla(\bar{\sigma}_1 \uplus \bar{\sigma}_2)$

7.3 Soundness

In this section, we present the soundness of CSL with partial permissions with regard to the semantics of Chapter 3. It follows a similar structure of Sec. 6.3

7.3.1 With Regard to the Interleaved Semantics

In this section, we present the soundness proof with regard to the interleaved semantics from Sec. 3.7. The proof is structured around the following definition:

Definition 7.8. $I \models \langle T, \bar{\sigma} \rangle \rhd_0 Q$ always holds; $I \models \langle T, \bar{\sigma} \rangle \rhd_{n+1} Q$ holds if, and only if, the following are true:

1. T is either 0- or 1-atomic

2. $\neg\langle T, \bar{\sigma}|_{\Pi} \rangle \longmapsto \textsf{abort}$

3. $\neg\langle T, \bar{\sigma}|_{\Pi} \rangle \longmapsto \textsf{race}$

4. If $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle T', \bar{\sigma}'|_{\Pi}\rangle$, then

   (a) If both T and T' are 1-atomic, then $I \models \langle T', \bar{\sigma}'\rangle \rhd_n Q$

   (b) If both T and T' are 0-atomic, then $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$ and $I \models \langle T', \bar{\sigma}'\rangle \rhd_n Q$

   (c) If T is 0-atomic and T' is 1-atomic, then for all $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that $\bar{\sigma}_2 = \bar{\sigma}_1 \uplus \bar{\sigma}'$ and $\bar{\sigma}_1 \in I$, we have $I \models \langle T', \bar{\sigma}_2\rangle \rhd_n Q$

   (d) If T is 1-atomic and T' is 0-atomic, then exists $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that $\bar{\sigma}' = \bar{\sigma}_1 \uplus \bar{\sigma}_2$, $\bar{\sigma}_1 \in I$, and $I \models \langle T', \bar{\sigma}_2\rangle \rhd_n Q$

5. If $T = \mathbf{skip}$, then $\bar{\sigma} \in Q$

6. If $T = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$, then $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$

We define $I \models \langle T, \bar{\sigma}\rangle \rhd Q$ as $\forall n.\ I \models \langle T, \bar{\sigma}\rangle \rhd_n Q$.

The triple $I \models \langle T, \bar{\sigma}\rangle \rhd Q$ ensures that each step performed by a program configuration has at most one ongoing atomic block execution (item 1), does not abort (item 2), and is not at a race condition (item 3). Furthermore, if it reaches a final configuration $\langle \mathbf{skip}, \bar{\sigma}'\rangle$, then $\bar{\sigma}'$ must satisfy post-condition Q (item 5). This definition also manages proper access to private memory (item 4). If the current configuration has an ongoing atomic block execution (item (a)), then it already has a hold of the shared memory, and it can perform the step with out constraints. If the current configuration does not have an ongoing atomic block execution (item (b)), then no memory allocation or deallocation must happen to avoid a race-condition with the environment, which may have an ongoing atomic block execution performing memory allocation or deallocation. This constraint is enforced by the condition $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$. If the current program configuration is starting to execute a top-level atomic block (item (c)), then it must get a hold on the shared memory, assuming it satisfies I. If the current program configuration is completing the execution of a top-level atomic block (item (d)), then it must return the shared memory ensuring that it satisfies I. Notice that this definition strips the permissions from the state ($\bar{\sigma}|_{\Pi}$) whenever it refers to the dynamic semantics ($\longmapsto$). This seems natural, as the permissions are logical and not carried out by the execution. Nevertheless, we must enforce, additionally, that read-only locations are not modified by the dynamic semantics, which could only happen through the execution of an action a. Therefore, we check whether the footprint of any a, that is in the imminence of being executed, is smaller than the largest compatible footprint of the current state $\bar{\sigma}$ (item 6).

We present the soundness proof in three sections, following the order:

1. Auxiliary lemmas for CSL triples, as in Def. 7.8

2. Semantic rules, each corresponding to a syntactic rule from Fig. 7.4

3. Top level soundness theorem

Auxiliary lemmas. Given that programs may diverge, and since $I \models \langle T, \bar{\sigma} \rangle \triangleright Q$ is defined in terms of itself, we used indexing to ensure this definition is well-founded. Lemma 7.9 allows greater flexibility when dealing with indexing.

Lemma 7.9. If $I \models \langle T, \bar{\sigma} \rangle \triangleright_{n_1} Q$, and $n_2 \leq n_1$, then $I \models \langle T, \bar{\sigma} \rangle \triangleright_{n_2} Q$

Proof. By induction over n_1. If $n_1 = 0$, then $n_2 = 0$ as well, by Def. 7.8, we conclude. If $n_1 > 0$, by Def. 7.8, assuming (a) $I \models \langle T, \bar{\sigma} \rangle \triangleright_{n_1} Q$ and (b) $n_2 \leq n_1$, we have 6 cases:

- From (a), by Def. 7.8 (item 1), we know that T is 0- or 1-atomic

- From (a), by Def. 7.8 (item 2), we know that $\langle T, \bar{\sigma}|_\Pi \rangle \longmapsto \mathsf{abort}$ is false

- From (a), by Def. 7.8 (item 3), we know that $\langle T, \bar{\sigma}|_\Pi \rangle \longmapsto \mathsf{race}$ is false

- If (c) $\langle T, \bar{\sigma}|_\Pi \rangle \longmapsto \langle T', \bar{\sigma}'|_\Pi \rangle$, then we have 4 cases:

  – If both T and T' are 1-atomic, we need to show that $I \models \langle T', \bar{\sigma}' \rangle \triangleright_{(n_2-1)} Q$. From (a) and (c), by Def. 7.8 (item 4.a), we know that (d) $I \models \langle T', \bar{\sigma}' \rangle \triangleright_{(n_1-1)} Q$. Trivially, from (b), we know that (e) $n_2 - 1 \leq n_1 - 1$. From (d) and (e), by the induction hypothesis, we conclude

  – If both T and T' are 0-atomic, we need to show that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$ and $I \models \langle T', \bar{\sigma}' \rangle \triangleright_{(n_2-1)} Q$. From (a) and (c), by Def. 7.8 (item 4.b), we know already that

$\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$, and also that (d) $I \models \langle T', \bar{\sigma}' \rangle \rhd_{(n_1-1)} Q$. Trivially, from (b), we know that (e) $n_2 - 1 \leq n_1 - 1$. From (d) and (e), by the induction hypothesis, we conclude

  - If T is 0-atomic and T' is 1-atomic, we need to show that for all $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that (d) $\bar{\sigma}_2 = \bar{\sigma}_1 \uplus \bar{\sigma}'$ and (e) $\bar{\sigma}_1 \in I$, we have $I \models \langle T', \bar{\sigma}_2 \rangle \rhd_{(n_2-1)} Q$. From (a), (c), (d), and (e), by Def. 7.8 (item 4.c), we know that (f) $I \models \langle T', \bar{\sigma}_2 \rangle \rhd_{(n_1-1)} Q$. Trivially, from (b), we know that (g) $n_2 - 1 \leq n_1 - 1$. From (f) and (g), by the induction hypothesis, we conclude
  - If T is 1-atomic and T' is 0-atomic, we need to show that exists $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that $\bar{\sigma}' = \bar{\sigma}_1 \uplus \bar{\sigma}_2$, $\bar{\sigma}_1 \in I$, and $I \models \langle T', \bar{\sigma}_2 \rangle \rhd_{(n_2-1)} Q$. From (a) and (c), by Def. 7.8 (item 4.d), we know there exists $\bar{\sigma}'_1$ and $\bar{\sigma}'_2$, such that (d) $\bar{\sigma}' = \bar{\sigma}'_1 \uplus \bar{\sigma}'_2$, (e) $\bar{\sigma}'_1 \in I$, and (f) $I \models \langle T', \bar{\sigma}'_2 \rangle \rhd_{(n_1-1)} Q$. Trivially, from (b), we know that (g) $n_2 - 1 \leq n_1 - 1$. From (f) and (g), by the induction hypothesis, we have (h) $I \models \langle T', \bar{\sigma}'_2 \rangle \rhd_{(n_2-1)} Q$. Instantiating the goal with $\bar{\sigma}'_1$ and $\bar{\sigma}'_2$, from (d), (e), and (h), we conclude

- We assume (c) $T = \textbf{skip}$. From (a) and (c), by Def. 7.8 (item 5), we know that $\bar{\sigma} \in Q$ and conclude

- We assume (c) $T = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$. From (a) and (c), by Def. 7.8 (item 6), we know that $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$ and conclude □

The following lemma allows the weakening of Q in a CSL triple.

Lemma 7.10. If $I \models \langle T, \bar{\sigma} \rangle \rhd_n Q$, and $Q \subseteq Q'$, then $I \models \langle T, \bar{\sigma} \rangle \rhd_n Q'$

Proof. By induction over n. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, by Def. 7.8, assuming (a) $I \models \langle T, \bar{\sigma} \rangle \rhd_n Q$ and (b) $Q \subseteq Q'$, we have 6 cases:

- From (a), by Def. 7.8 (item 1), we know that T is 0- or 1-atomic

- From (a), by Def. 7.8 (item 2), we know that $\langle T, \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{abort}$ is false

- From (a), by Def. 7.8 (item 3), we know that $\langle T, \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{race}$ is false

- If (c) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle T', \bar{\sigma}'|_{\Pi}\rangle$, then we have 4 cases:
  - If both T and T' are 1-atomic, we need to show that $I \models \langle T', \bar{\sigma}'\rangle \rhd_{n-1} Q'$. From (a) and (c), by Def. 7.8 (item 4.a), we know that (d) $I \models \langle T', \bar{\sigma}'\rangle \rhd_{n-1} Q$. From (b) and (d), by the induction hypothesis, we conclude
  - If both T and T' are 0-atomic, we need to show that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$ and $I \models \langle T', \bar{\sigma}'\rangle \rhd_{n-1} Q'$. From (a) and (c), by Def. 7.8 (item 4.b), we know already that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$, and also that (d) $I \models \langle T', \bar{\sigma}'\rangle \rhd_{n-1} Q$. From (b) and (d), by the induction hypothesis, we conclude
  - If T is 0-atomic and T' is 1-atomic, we need to show that for all $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that (d) $\bar{\sigma}_2 = \bar{\sigma}_1 \uplus \bar{\sigma}'$ and (e) $\bar{\sigma}_1 \in I$, we have $I \models \langle T', \bar{\sigma}_2\rangle \rhd_{n-1} Q'$. From (a), (c), (d), and (e), by Def. 7.8 (item 4.c), we know that (f) $I \models \langle T', \bar{\sigma}_2\rangle \rhd_{n-1} Q$. From (b) and (f), by the induction hypothesis, we conclude
  - If T is 1-atomic and T' is 0-atomic, we need to show that exists $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that $\bar{\sigma}' = \bar{\sigma}_1 \uplus \bar{\sigma}_2$, $\bar{\sigma}_1 \in I$, and $I \models \langle T', \bar{\sigma}_2\rangle \rhd_{n-1} Q'$. From (a) and (c), by Def. 7.8 (item 4.d), we know there exists $\bar{\sigma}_1'$ and $\bar{\sigma}_2'$, such that (d) $\bar{\sigma}' = \bar{\sigma}_1' \uplus \bar{\sigma}_2'$, (e) $\bar{\sigma}_1' \in I$, and (f) $I \models \langle T', \bar{\sigma}_2'\rangle \rhd_{n-1} Q$. From (b) and (f), by the induction hypothesis, we have (g) $I \models \langle T', \bar{\sigma}_2'\rangle \rhd_{n-1} Q'$. Instantiating the goal with $\bar{\sigma}_1'$ and $\bar{\sigma}_2'$, from (d), (e), and (g), we conclude
- We assume (c) $T = \textbf{skip}$. From (a) and (c), by Def. 7.8 (item 5), we know that (d) $\bar{\sigma} \in Q$. From (b) and (d), we know that $\bar{\sigma} \in Q'$ and conclude
- We assume (c) $T = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$. From (a) and (c), by Def. 7.8 (item 6), we know that $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$ and conclude □

We can construct a CSL triple from **skip** using the following lemma.

Lemma 7.11. If $\bar{\sigma} \in Q$, then $I \models \langle \textbf{skip}, \bar{\sigma}\rangle \rhd_n Q$

Proof. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, by Def. 7.8, we have 6 cases:

- By Def. 3.1, we know that **skip** is 0-atomic

- From the semantics, we know that $\langle \mathbf{skip}, \bar{\sigma}|_{\Pi} \rangle \longmapsto \textsf{abort}$ is false
- From the semantics, we know that $\langle \mathbf{skip}, \bar{\sigma}|_{\Pi} \rangle \longmapsto \textsf{race}$ is false
- From the semantics, we know that $\langle \mathbf{skip}, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle T', \bar{\sigma}'|_{\Pi} \rangle$ is false
- Since $\mathbf{skip} = \mathbf{skip}$, and $\bar{\sigma} \in Q$, we conclude
- We know that $\mathbf{skip} \neq \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$ □

The following lemma is used for sequential composition of triples.

Lemma 7.12. If $I \models \langle T, \bar{\sigma} \rangle \rhd_n P$, and, for all $\bar{\sigma}' \in P$, we have $I \models \langle c', \bar{\sigma}' \rangle \rhd_n Q$, then

1. If $T = c$, then $I \models \langle c; c', \bar{\sigma} \rangle \rhd_n Q$
2. If $T = \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$, then $I \models \langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} (c; c'), \bar{\sigma} \rangle \rhd_n Q$
3. If $T = \langle\!\langle T' \rangle\!\rangle_{\mathbf{a}} c$, then $I \models \langle \langle\!\langle T' \rangle\!\rangle_{\mathbf{a}} (c; c'), \bar{\sigma} \rangle \rhd_n Q$

Proof. By induction over n. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, we assume (a) $I \models \langle T, \bar{\sigma} \rangle \rhd_n P$ and (b) for all $\bar{\sigma}' \in P$, we have $I \models \langle c', \bar{\sigma}' \rangle \rhd_n Q$. We establish (c) for all $\bar{\sigma}' \in P$, we have $I \models \langle c', \bar{\sigma}' \rangle \rhd_{n-1} Q$:

- We assume $\bar{\sigma}' \in P$, from (b), using Lemma 7.9, we conclude

Then we consider 3 cases:

- If $T = c$, then we need to show that $I \models \langle c; c', \bar{\sigma} \rangle \rhd_n Q$. By Def. 7.8, we have 6 cases:
  - By Def. 3.1, we know that $c; c'$ is 0-atomic
  - From (a), by Def. 7.8 (item 2), we know that (d) $\langle c, \bar{\sigma}|_{\Pi} \rangle \longmapsto \textsf{abort}$ is false. From (d), and the semantics, we know that $\langle c; c', \bar{\sigma}|_{\Pi} \rangle \longmapsto \textsf{abort}$ is also false
  - From the semantics, we know that $\langle c; c', \bar{\sigma}|_{\Pi} \rangle \longmapsto \textsf{race}$ is false
  - If (d) $\langle c; c', \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle T', \bar{\sigma}'|_{\Pi} \rangle$, from the semantics, we have 4 cases:

    * If $c = \mathbf{skip}$, $T' = c'$, and $\bar{\sigma}'|_{\Pi} = \bar{\sigma}|_{\Pi}$, since both $\mathbf{skip}; c'$ and c' are 0-atomic, we need to show that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$, which holds trivially, and $I \models \langle c', \bar{\sigma}' \rangle \rhd_{n-1} Q$. From (a), by Def. 7.8 (item 5), we know that (e) $\bar{\sigma} \in P$. From (e) and (c), we conclude
    * If $T' = c''; c'$, and (e) $\langle c, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle c'', \bar{\sigma}'|_{\Pi} \rangle$, given that both $c; c'$ and $c''; c'$ are 0-atomic, we need to show $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$ and $I \models \langle c''; c', \bar{\sigma}' \rangle \rhd_{n-1} Q$. From (a) and (e), by Def. 7.8 (item 4.b), we have (f) $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$ and (g) $I \models \langle c'', \bar{\sigma}' \rangle \rhd_{n-1} P$. From (g) and (c), using the induction hypothesis (item 1), we have (h) $I \models \langle c''; c', \bar{\sigma}' \rangle \rhd_{n-1} Q$. From (f) and (h), we conclude
    * If $c = \mathbf{S}[\, c_1 \parallel c_2 \,]$, $T' = \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\, \mathbf{skip} \,]; c')$, and $\bar{\sigma}'|_{\Pi} = \bar{\sigma}|_{\Pi}$, since both $c; c'$ and $\langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\, \mathbf{skip} \,]; c')$ are 0-atomic, we need to show that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$, which holds trivially, and $I \models \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\, \mathbf{skip} \,]; c'), \bar{\sigma} \rangle \rhd_{n-1} Q$. From the semantics, we know that (e) $\langle c, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\, \mathbf{skip} \,]), \bar{\sigma}|_{\Pi} \rangle$. From (a) and (e), by Def. 7.8 (item 4.b), we know that (f) $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma})$ and (g) $I \models \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\, \mathbf{skip} \,]), \bar{\sigma} \rangle \rhd_{n-1} P$. From (g) and (c), using the induction hypothesis (item 2), we conclude
    * If $c = \mathbf{S}[\, \mathbf{atomic}\ c'' \,]$, $T' = \langle\!\langle c'' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\, \mathbf{skip} \,]; c')$, and $\bar{\sigma}'|_{\Pi} = \bar{\sigma}|_{\Pi}$, since $c; c'$ is 0-atomic and $\langle\!\langle c'' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\, \mathbf{skip} \,]; c')$ is 1-atomic, we need to show that for all $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that (e) $\bar{\sigma}_2 = \bar{\sigma}_1 \uplus \bar{\sigma}$, and (f) $\bar{\sigma}_1 \in I$, we have $I \models \langle T', \bar{\sigma}_2 \rangle \rhd_{n-1} Q$. From the semantics, we know that (g) $\langle c, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle \langle\!\langle c'' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\, \mathbf{skip} \,]), \bar{\sigma}|_{\Pi} \rangle$. From (a), (g), (e), and (f), by Def. 7.8 (item 4.c), we have
      (h) $I \models \langle \langle\!\langle c'' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\, \mathbf{skip} \,]), \bar{\sigma}_2 \rangle \rhd_{n-1} P$. From (h) and (c), using the induction hypothesis (item 3), we conclude
  - We know that $c; c' \neq \mathbf{skip}$
  - If $c; c' = \mathbf{T}[\, \mathbf{S}[\, a \,] \,]$, then we know that (d) $\mathbf{T} = \bullet$ and exists $\mathbf{S}'$ such that (e) $\mathbf{S} = (\mathbf{S}'; c')$ and (f) $c = \mathbf{S}'[\, a \,]$. From (d), (f), and (a), by Def. 7.8 (item 6), we know that $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$ and conclude

- If $T = \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$, and (d) $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$ is 0- or 1-atomic, then we need to show that

$I \models \langle \llangle T_1, T_2 \rrangle_{\mathrm{p}}(c; c'), \bar{\sigma} \rangle \rhd_n Q$. By Def. 7.8, we have 6 cases:

- From (d), by Def. 3.1, we know that $\llangle T_1, T_2 \rrangle_{\mathrm{p}}(c; c')$ is 0- or 1-atomic
- From (a), by Def. 7.8 (item 2), we know that (e) $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}} c, \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{abort}$ is false. From (e), and the semantics, we know that $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}}(c; c'), \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{abort}$ is also false
- From (a), by Def. 7.8 (item 3), we know that (e) $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}} c, \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{race}$ is false. From (e), and the semantics, we know that $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}}(c; c'), \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{race}$ is also false
- If (e) $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}}(c; c'), \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle T', \bar{\sigma}'|_{\Pi} \rangle$, from the semantics, we have 3 cases:
  * If $T_1 = \mathbf{skip}$, $T_2 = \mathbf{skip}$, $T' = c; c'$, and $\bar{\sigma}'|_{\Pi} = \bar{\sigma}|_{\Pi}$, since both $\llangle \mathbf{skip}, \mathbf{skip} \rrangle_{\mathrm{p}}(c; c')$ and $c; c'$ are 0-atomic, we need to show that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$, which holds trivially, and $I \models \langle c; c', \bar{\sigma} \rangle \rhd_{n-1} Q$. From the semantics, we know that (f) $\langle \llangle \mathbf{skip}, \mathbf{skip} \rrangle_{\mathrm{p}} c, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle c, \bar{\sigma}|_{\Pi} \rangle$. From (a) and (f), by Def. 7.8 (item 4.b), we know that (g) $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma})$ and (h) $I \models \langle c, \bar{\sigma} \rangle \rhd_{n-1} P$. From (h), (c), by the induction hypothesis (item 1), we conclude
  * If $T' = \llangle T_1', T_2 \rrangle_{\mathrm{p}}(c; c')$, and (f) $\langle T_1, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle T_1', \bar{\sigma}'|_{\Pi} \rangle$, we have 4 cases:
    · If both T_1 and T_1' are 1-atomic, we have to show that $I \models \langle \llangle T_1', T_2 \rrangle_{\mathrm{p}}(c; c'), \bar{\sigma}' \rangle \rhd_{n-1} Q$. From (f), and the semantics, we know (g) $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}} c, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle \llangle T_1', T_2 \rrangle_{\mathrm{p}} c, \bar{\sigma}'|_{\Pi} \rangle$. From (a) and (g), by Def. 7.8 (item 4.a), we know (h) $I \models \langle \llangle T_1', T_2 \rrangle_{\mathrm{p}} c, \bar{\sigma}' \rangle \rhd_{n-1} P$. From (h) and (c), using the induction hypothesis (item 2), we conclude

    · If both T_1 and T_1' are 0-atomic, we have to show that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$ and $I \models \langle \llangle T_1', T_2 \rrangle_{\mathrm{p}}(c; c'), \bar{\sigma}' \rangle \rhd_{n-1} Q$. From (f), and the semantics, we know (g) $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}} c, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle \llangle T_1', T_2 \rrangle_{\mathrm{p}} c, \bar{\sigma}'|_{\Pi} \rangle$. From (a) and (g), by Def. 7.8 (item 4.b), we know (h) $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$ and (i) $I \models \langle \llangle T_1', T_2 \rrangle_{\mathrm{p}} c, \bar{\sigma}' \rangle \rhd_{n-1} P$. From (i) and (c), using the induction hypothesis (item 2), we know (j) $I \models \langle \llangle T_1', T_2 \rrangle_{\mathrm{p}}(c; c'), \bar{\sigma}' \rangle \rhd_{n-1} Q$. From (h) and (j), we conclude

· If T_1 is 0-atomic and T_1' is 1-atomic, we need to show that for all $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that (g) $\bar{\sigma}_2 = \bar{\sigma}_1 \uplus \bar{\sigma}'$ and (h) $\bar{\sigma}_1 \in I$, we have $I \models \langle\langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathbf{p}}(c; c'), \bar{\sigma}_2\rangle \rhd_{n-1} Q$. From (f), and the semantics, we know (i) $\langle\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle\langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathbf{p}} c, \bar{\sigma}'|_{\Pi}\rangle$. From (a), (i), (g), and (h), by Def. 7.8 (item 4.c), we know that (j) $I \models \langle\langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathbf{p}} c, \bar{\sigma}_2\rangle \rhd_{n-1} P$. From (j) and (c), using the induction hypothesis (item 2), we conclude

· If T_1 is 1-atomic and T_1' is 0-atomic, we need to show that exists $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that $\bar{\sigma}' = \bar{\sigma}_1 \uplus \bar{\sigma}_2$, $\bar{\sigma}_1 \in I$, and $I \models \langle\langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathbf{p}}(c; c'), \bar{\sigma}_2\rangle \rhd_{n-1} Q$. From (f), and the semantics, we know (g) $\langle\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle\langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathbf{p}} c, \bar{\sigma}'|_{\Pi}\rangle$. From (a) and (g), by Def. 7.8 (item 4.d), we know there exists $\bar{\sigma}_1'$ and $\bar{\sigma}_2'$ such that (i) $\bar{\sigma}' = \bar{\sigma}_1' \uplus \bar{\sigma}_2'$, (j) $\bar{\sigma}_1' \in I$, and (k) $I \models \langle\langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathbf{p}} c, \bar{\sigma}_2'\rangle \rhd_{n-1} P$. From (k) and (c), using the induction hypothesis (item 2), we know (l) $I \models \langle\langle\!\langle T_1', T_2 \rangle\!\rangle_{\mathbf{p}}(c; c'), \bar{\sigma}_2'\rangle \rhd_{n-1} Q$. Instantiating the goal with $\bar{\sigma}_1'$ and $\bar{\sigma}_2'$, from (i), (j), and (l), we conclude

  * If $T' = \langle\!\langle T_1, T_2' \rangle\!\rangle_{\mathbf{p}}(c; c')$, and (f) $\langle T_2, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle T_2', \bar{\sigma}'|_{\Pi}\rangle$, the proof is symmetric to the previous case

– We know that $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}(c; c') \neq \mathbf{skip}$

– If $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}(c; c') = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$, we have 2 cases:

  * we know that exists $\mathbf{T}_1$ such that (d) $\mathbf{T} = \langle\!\langle \mathbf{T}_1, T_2 \rangle\!\rangle_{\mathbf{p}}(c; c')$ and (e) $T_1 = \mathbf{T}_1[\,\mathbf{S}[\,a\,]\,]$. From (e) and (a), by Def. 7.8 (item 6), we know that $\Delta^a_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$ and conclude

  * we know that exists $\mathbf{T}_2$ such that (d) $\mathbf{T} = \langle\!\langle T_1, \mathbf{T}_2 \rangle\!\rangle_{\mathbf{p}}(c; c')$ and (e) $T_2 = \mathbf{T}_2[\,\mathbf{S}[\,a\,]\,]$. From (e) and (a), by Def. 7.8 (item 6), we know that $\Delta^a_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$ and conclude

• If $T = \langle\!\langle T' \rangle\!\rangle_{\mathbf{a}} c$, then we need to show that $I \models \langle\langle\!\langle T' \rangle\!\rangle_{\mathbf{a}}(c; c'), \bar{\sigma}\rangle \rhd_n Q$. By Def. 7.8, we have 6 cases:

– By Def. 3.1, we know that $\langle\!\langle T' \rangle\!\rangle_{\mathbf{a}}(c; c')$ is 1-atomic

– From (a), by Def. 7.8 (item 2), we know that (d) $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} c, \bar{\sigma}|_{\Pi}\rangle \longmapsto \mathsf{abort}$ is false. From (d), and the semantics, we know that $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} (c; c'), \bar{\sigma}|_{\Pi}\rangle \longmapsto \mathsf{abort}$ is also false

– From (a), by Def. 7.8 (item 3), we know that (d) $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} c, \bar{\sigma}|_{\Pi}\rangle \longmapsto \mathsf{race}$ is false. From (d), and the semantics, we know that $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} (c; c'), \bar{\sigma}|_{\Pi}\rangle \longmapsto \mathsf{race}$ is also false

– If (d) $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} (c; c'), \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle T'', \bar{\sigma}'|_{\Pi}\rangle$, from the semantics, we have 2 cases:

  * If $T' = \mathbf{skip}$, $T'' = c; c'$, and $\bar{\sigma}'|_{\Pi} = \bar{\sigma}|_{\Pi}$, since $\langle\!\langle \mathbf{skip}\rangle\!\rangle_{\mathsf{a}} (c; c')$ is 1-atomic and $c; c'$ is 0-atomic, we need to show that exists $\bar{\sigma}_1$ and $\bar{\sigma}_2$ such that $\bar{\sigma}' = \bar{\sigma}_1 \uplus \bar{\sigma}_2$, $\bar{\sigma}_1 \in I$, and $I \models \langle c; c', \bar{\sigma}_2\rangle \rhd_{n-1} Q$. From the semantics, we know that (e) $\langle\langle\!\langle \mathbf{skip}\rangle\!\rangle_{\mathsf{a}} c, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle c, \bar{\sigma}|_{\Pi}\rangle$. From (a) and (e), by Def. 7.8 (item 4.d), we know that exists $\bar{\sigma}_1'$ and $\bar{\sigma}_2'$ such that (f) $\bar{\sigma} = \bar{\sigma}_1' \uplus \bar{\sigma}_2'$, (g) $\bar{\sigma}_1' \in I$, and (h) $I \models \langle c, \bar{\sigma}_2'\rangle \rhd_{n-1} P$. From (h), (c), by the induction hypothesis (item 1), we have (i) $I \models \langle c; c', \bar{\sigma}_2'\rangle \rhd_{n-1} Q$. Instantiating the goal with $\bar{\sigma}_1'$ and $\bar{\sigma}_2'$, from (f), (g), and (i), we conclude
  * If $T'' = \langle\!\langle T'''\rangle\!\rangle_{\mathsf{a}} (c; c')$, and (e) $\langle T', \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle T''', \bar{\sigma}'|_{\Pi}\rangle$, given that both $\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} (c; c')$ and $\langle\!\langle T'''\rangle\!\rangle_{\mathsf{a}} (c; c')$ are 1-atomic, we need to show $I \models \langle\langle\!\langle T'''\rangle\!\rangle_{\mathsf{a}} (c; c'), \bar{\sigma}'\rangle \rhd_{n-1} Q$. From (e), and the semantics, we know (f) $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} c, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle\langle\!\langle T'''\rangle\!\rangle_{\mathsf{a}} c, \bar{\sigma}'|_{\Pi}\rangle$. From (a) and (f), by Def. 7.8 (item 4.a), then (g) $I \models \langle\langle\!\langle T'''\rangle\!\rangle_{\mathsf{a}} c, \bar{\sigma}'\rangle \rhd_{n-1} P$. From (g) and (c), using the induction hypothesis (item 3), we conclude

– We know that $\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} (c; c') \neq \mathbf{skip}$

– If $\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} (c; c') = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$, then we know that exists $\mathbf{T}'$ such that (d) $\mathbf{T} = \langle\!\langle \mathbf{T}'\rangle\!\rangle_{\mathsf{a}} (c; c')$ and (e) $T' = \mathbf{T}'[\,\mathbf{S}[\,a\,]\,]$. From (e) and (a), by Def. 7.8 (item 6), we know that $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$ and conclude □

Conditional commands can be introduced using the following lemma.

Lemma 7.13. If $I \models \langle c, \bar{\sigma}\rangle \rhd_n Q$, then

1. If $\bar{\sigma} \in \lfloor b \rfloor$, then $I \models \langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \bar{\sigma} \rangle \rhd_n Q$
2. If $\bar{\sigma} \in \lfloor \neg b \rfloor$, then $I \models \langle \textbf{if } b \textbf{ then } c' \textbf{ else } c, \bar{\sigma} \rangle \rhd_n Q$

Proof. From assumption (a) $I \models \langle c, \bar{\sigma} \rangle \rhd_n Q$ we have 2 cases:

- We assume (b) $\bar{\sigma} \in \lfloor b \rfloor$. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, by Def. 7.8, we have 6 cases:
  - By Def. 3.1, we know that (c) **if** b **then** c **else** c' is 0-atomic
  - From the semantics, we know that $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \bar{\sigma}|_\Pi \rangle \longmapsto \mathsf{abort}$ is false if $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \bar{\sigma}|_\Pi \rangle \longrightarrow \mathsf{abort}$ is false. From the sequential semantics, we know $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \bar{\sigma}|_\Pi \rangle \longrightarrow \mathsf{abort}$ is false if there exists z such that $[\![b]\!]_{(\bar{\sigma}|_\Pi)} = z$. From (b), we know $[\![b]\!]_{(\bar{\sigma}|_\Pi)} = \textit{true}$ and conclude
  - From the semantics, we know that $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \bar{\sigma}|_\Pi \rangle \longmapsto \mathsf{race}$ is false
  - From the semantics, given (b), we know that $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \bar{\sigma}|_\Pi \rangle \longmapsto \langle c, \bar{\sigma}|_\Pi \rangle$. From (c), and since c is 0-atomic, we need to show that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma})$ and $I \models \langle c, \bar{\sigma} \rangle \rhd_{n-1} Q$. From (a), using Lemma 7.9, we conclude
  - We know that **if** b **then** c **else** $c' \neq \textbf{skip}$
  - We know that **if** b **then** c **else** $c' \neq \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$
- We assume (b) $\bar{\sigma} \in \lfloor \neg b \rfloor$. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, by Def. 7.8, we have 6 cases:
  - By Def. 3.1, we know that (c) **if** b **then** c' **else** c is 0-atomic
  - From the semantics, we know that $\langle \textbf{if } b \textbf{ then } c' \textbf{ else } c, \bar{\sigma}|_\Pi \rangle \longmapsto \mathsf{abort}$ is false if $\langle \textbf{if } b \textbf{ then } c' \textbf{ else } c, \bar{\sigma}|_\Pi \rangle \longrightarrow \mathsf{abort}$ is false. From the sequential semantics, we know $\langle \textbf{if } b \textbf{ then } c' \textbf{ else } c, \bar{\sigma}|_\Pi \rangle \longrightarrow \mathsf{abort}$ is false if there exists z such that $[\![b]\!]_{(\bar{\sigma}|_\Pi)} = z$. From (b), we know $[\![b]\!]_{(\bar{\sigma}|_\Pi)} = \textit{false}$ and conclude
  - From the semantics, we know that $\langle \textbf{if } b \textbf{ then } c' \textbf{ else } c, \bar{\sigma}|_\Pi \rangle \longmapsto \mathsf{race}$ is false

– From the semantics, given (b), we know that $\langle \textbf{if}\ b\ \textbf{then}\ c'\ \textbf{else}\ c, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle c, \bar{\sigma}|_{\Pi}\rangle$. From (c), and since c is 0-atomic, we need to show that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma})$ and $I \models \langle c, \bar{\sigma}\rangle \rhd_{n-1} Q$. From (a), using Lemma 7.9, we conclude

– We know that $\textbf{if}\ b\ \textbf{then}\ c'\ \textbf{else}\ c \neq \textbf{skip}$

– We know that $\textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ c' \neq \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$ □

A loop command can be introduced using the following lemma.

Lemma 7.14. If $P \subseteq \lfloor b\rfloor \cup \lfloor \neg b\rfloor$, $\bar{\sigma} \in P$, and, for all $\bar{\sigma}' \in P \cap \lfloor b\rfloor$, we have $I \models \langle c, \bar{\sigma}'\rangle \rhd_n P$, then $I \models \langle \textbf{while}\ b\ \textbf{do}\ c, \bar{\sigma}\rangle \rhd_n (P \cap \lfloor \neg b\rfloor)$

Proof. By induction over n. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, by Def. 7.8, assuming (a) $P \subseteq \lfloor b\rfloor \cup \lfloor \neg b\rfloor$, (b) $\bar{\sigma} \in P$, and, (c) for all $\bar{\sigma}' \in P \cap \lfloor b\rfloor$, we have $I \models \langle c, \bar{\sigma}'\rangle \rhd_n P$, we have 6 cases:

- By Def. 3.1, we know that (d) $\textbf{while}\ b\ \textbf{do}\ c$ is 0-atomic

- From the semantics, we know that $\langle \textbf{while}\ b\ \textbf{do}\ c, \bar{\sigma}|_{\Pi}\rangle \longmapsto \mathsf{abort}$ is false

- From the semantics, we know that $\langle \textbf{while}\ b\ \textbf{do}\ c, \bar{\sigma}|_{\Pi}\rangle \longmapsto \mathsf{race}$ is false

- From the semantics, $\langle \textbf{while}\ b\ \textbf{do}\ c, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle \textbf{if}\ b\ \textbf{then}\ (c; \textbf{while}\ b\ \textbf{do}\ c)\ \textbf{else skip}, \bar{\sigma}|_{\Pi}\rangle$. From (d), and since $\textbf{if}\ b\ \textbf{then}\ (c; \textbf{while}\ b\ \textbf{do}\ c)\ \textbf{else skip}$ is 0-atomic, we need to show that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma})$ and $I \models \langle \textbf{if}\ b\ \textbf{then}\ (c; \textbf{while}\ b\ \textbf{do}\ c)\ \textbf{else skip}, \bar{\sigma}\rangle \rhd_{n-1} (P \cap \lfloor \neg b\rfloor)$. From (a) and (b), we know (e) $\bar{\sigma} \in \lfloor b\rfloor \cup \lfloor \neg b\rfloor$. From (e), we have two cases:

  – We assume (f) $\bar{\sigma} \in \lfloor b\rfloor$. From (b) and (f), we know (g) $\bar{\sigma} \in P \cap \lfloor b\rfloor$. We establish (h) for all $\bar{\sigma}' \in P \cap \lfloor b\rfloor$, we have $I \models \langle c, \bar{\sigma}'\rangle \rhd_{n-1} P$:

    * We assume $\bar{\sigma}' \in P \cap \lfloor b\rfloor$, from (c), using Lemma 7.9, we conclude

  From (g) and (h), we know (i) $I \models \langle c, \bar{\sigma}\rangle \rhd_{n-1} P$. We establish (j) for all $\bar{\sigma}' \in P$, we have $I \models \langle \textbf{while}\ b\ \textbf{do}\ c, \bar{\sigma}'\rangle \rhd_{n-1} (P \cap \lfloor \neg b\rfloor)$:

    * We assume $\bar{\sigma}' \in P$, with (a) and (h), using the induction hypothesis, we conclude

From (i) and (j), using Lemma 7.12 (item 1), we get
(k) $I \models \langle c; \textbf{while}\ b\ \textbf{do}\ c, \bar{\sigma}\rangle \rhd_{n-1} (P \cap \lfloor\neg b\rfloor)$. From (f) and (k), using Lemma 7.13 (item 1), we conclude

  - We assume (f) $\bar{\sigma} \in \lfloor\neg b\rfloor$. From (b) and (f), we know (g) $\bar{\sigma} \in P \cap \lfloor\neg b\rfloor$. From (g), using Lemma 7.11, we know (h) $I \models \langle \textbf{skip}, \bar{\sigma}\rangle \rhd_{n-1} (P \cap \lfloor\neg b\rfloor)$. From (f) and (h), using Lemma 7.13 (item 2), we conclude

- We know that $\textbf{while}\ b\ \textbf{do}\ c \neq \textbf{skip}$
- We know that $\textbf{while}\ b\ \textbf{do}\ c \neq \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$ ☐

The following lemma is used for parallel composition of triples.

Lemma 7.15. If $I \models \langle T_1, \bar{\sigma}_1\rangle \rhd_n Q_1$, $I \models \langle T_2, \bar{\sigma}_2\rangle \rhd_n Q_2$, and $\langle\!\langle T_1, T_2\rangle\!\rangle_{\mathrm{p}}\textbf{skip}$ is 0- or 1-atomic, then $I \models \langle \langle\!\langle T_1, T_2\rangle\!\rangle_{\mathrm{p}}\textbf{skip}, \bar{\sigma}_1 \uplus \bar{\sigma}_2\rangle \rhd_n (Q_1 {*} Q_2)$

Proof. By induction over n. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, by Def. 7.8, assuming (a) $I \models \langle T_1, \bar{\sigma}_1\rangle \rhd_n Q_1$, (b) $I \models \langle T_2, \bar{\sigma}_2\rangle \rhd_n Q_2$, and (c) $\langle\!\langle T_1, T_2\rangle\!\rangle_{\mathrm{p}}\textbf{skip}$ is 0- or 1-atomic, we have 6 cases:

- From (c), we conclude
- From the semantics, we know that $\langle \langle\!\langle T_1, T_2\rangle\!\rangle_{\mathrm{p}}\textbf{skip}, \bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1 \uplus \Pi_2}\rangle \longmapsto \mathsf{abort}$ if either:
  - $\langle T_1, \bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1 \uplus \Pi_2}\rangle \longmapsto \mathsf{abort}$. From (a), by Def. 7.8 (item 2), we know (d) $\langle T_1, \bar{\sigma}_1|_{\Pi_1}\rangle \longmapsto \mathsf{abort}$ is false. From (d), using Lemma 3.22 (item 1), we conclude
  - or, $\langle T_2, \bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1 \uplus \Pi_2}\rangle \longmapsto \mathsf{abort}$. From (b), by Def. 7.8 (item 2), we know (e) $\langle T_2, \bar{\sigma}_2|_{\Pi_2}\rangle \longmapsto \mathsf{abort}$ is false. From (e), using Lemma 3.22 (item 1), we conclude
- From the semantics, we know that $\langle \langle\!\langle T_1, T_2\rangle\!\rangle_{\mathrm{p}}\textbf{skip}, \bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1 \uplus \Pi_2}\rangle \longmapsto \mathsf{race}$ if either:
  - $\langle T_1, \bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1 \uplus \Pi_2}\rangle \longmapsto \mathsf{race}$. From (a), by Def. 7.8 (item 3), we know
(f) $\langle T_1, \bar{\sigma}_1|_{\Pi_1}\rangle \longmapsto \mathsf{race}$ is false. From (d) and (f), using Lemma 3.22 (item 2), we conclude

– or, $\langle T_2, \bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1 \uplus \Pi_2}\rangle \longmapsto \textsf{race}$. From (b), by Def. 7.8 (item 3), we know (g) $\langle T_2, \bar{\sigma}_2|_{\Pi_2}\rangle \longmapsto \textsf{race}$ is false. From (e) and (g), using Lemma 3.22 (item 2), we conclude

– or, $T_1 = \mathbf{T}_1[\,c_1\,]$, $T_2 = \mathbf{T}_2[\,c_2\,]$, $\langle c_1, \bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1 \uplus \Pi_2}\rangle \xrightarrow[\delta_1]{} \langle c_1', \sigma'\rangle$, $\langle c_2, \sigma'\rangle \xrightarrow[\delta_2]{} \kappa$ and $\delta_1 \not\smile \delta_2$. By contradiction, we will assume (f) $\langle c_1, \bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1 \uplus \Pi_2}\rangle \xrightarrow[\delta_1]{} \langle c_1', \sigma'\rangle$, and (g) $\langle c_2, \sigma'\rangle \xrightarrow[\delta_2]{} \kappa$ in order to obtain $\delta_1 \smile \delta_2$. From the semantics, and (d), we know (h) $\langle c1, \bar{\sigma}_1|_{\Pi_1}\rangle \longrightarrow \textsf{abort}$ is false. From (h) and (f), using Lemma 3.16 (item 2), we know there exists $\bar{\sigma}_1'$ such that (i) $\sigma' = \bar{\sigma}_1' \uplus \bar{\sigma}_2|_{\Pi_1 \uplus \Pi_2}$ and (j) $\langle c1, \bar{\sigma}_1|_{\Pi_1}\rangle \xrightarrow[\delta_1]{} \langle c_1', \bar{\sigma}_1'|_{\Pi_1}\rangle$. From the semantics, and (e), we know (k) $\langle c2, \bar{\sigma}_2|_{\Pi_2}\rangle \longrightarrow \textsf{abort}$ is false. From (k), using Lemma 3.16 (item 1), we know (l) $\langle c2, \bar{\sigma}_1' \uplus \bar{\sigma}_2|_{\Pi_1 \uplus \Pi_2}\rangle \longrightarrow \textsf{abort}$ is false. From (g), (i), and (l), we know there exists c_2' and σ'' such that (m) $\kappa = \langle c_2', \sigma''\rangle$. From (k), (g) and (m), using Lemma 3.16 (item 2), we know there exists $\bar{\sigma}_2'$ such that (n) $\sigma'' = \bar{\sigma}_1' \uplus \bar{\sigma}_2'|_{\Pi_1 \uplus \Pi_2}$ and (o) $\langle c_2, \bar{\sigma}_2|_{\Pi_2}\rangle \xrightarrow[\delta_2]{} \langle c_2', \bar{\sigma}_2'|_{\Pi_2}\rangle$. From (j), using Remark 3.10 (item 2), by Def. 7.3, we know that
(p) $\delta_1 \subseteq (\textsf{dom}(\Pi_1), \textsf{dom}(\sigma_1|_{\Pi_1})\backslash\textsf{dom}(\Pi_1))$. From (o), using Remark 3.10 (item 2), by Def. 7.3, we know that (q) $\delta_2 \subseteq (\textsf{dom}(\Pi_2), \textsf{dom}(\sigma_2|_{\Pi_2})\backslash\textsf{dom}(\Pi_2))$. Then, we have 2 cases:

  ∗ If $\mathbf{T}_1[\,c_1\,]$ is 0-atomic, then from (a) and (j), by Def. 6.3 (item 4.b), we know that (r) $\textsf{dom}(\sigma_1) = \textsf{dom}(\sigma_1')$. From (p) and (r), we know that (s) $\delta_1 \subseteq (\emptyset, \textsf{dom}(\sigma_1'))$. From (i), (n), we know that (t) $\textsf{dom}(\sigma_1') \cap (\textsf{dom}(\sigma_2) \cup \textsf{dom}(\sigma_2')) = \emptyset$. From (t), (s), and (q), we know that $\delta_1.ws \cap (\delta_2.rs \cup \delta_2.ws) = \emptyset$ and conclude

  ∗ If $\mathbf{T}_1[\,c_1\,]$ is 1-atomic, from (c) we know that (r) $\mathbf{T}_2[\,c_2\,]$ is 0-atomic. From (b), (o), and (r), by Def. 6.3 (item 4.b), we know that (s) $\textsf{dom}(\sigma_2) = \textsf{dom}(\sigma_2')$. From (q) and (s), we know that (t) $\delta_2 \subseteq (\emptyset, \textsf{dom}(\sigma_2))$. From (i), we know that (u) $(\textsf{dom}(\sigma_1) \cup \textsf{dom}(\sigma_1')) \cap \textsf{dom}(\sigma_2) = \emptyset$. From (u), (p), and (t), we know that $\delta_1.ws \cap (\delta_2.rs \cup \delta_2.ws) = \emptyset$ and conclude

– or, $T_1 = \mathbf{T}_1[\,c_1\,]$, $T_2 = \mathbf{T}_2[\,c_2\,]$, $\langle c_2, \bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1 \uplus \Pi_2}\rangle \xrightarrow[\delta_2]{} \langle c_2', \sigma'\rangle$, $\langle c_1, \sigma'\rangle \xrightarrow[\delta_1]{} \kappa$ and $\delta_2 \not\smile \delta_1$. The proof is symmetric to the previous case

- From the semantics, if (f) $\langle\langle\!\langle T_1, T_2\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}, \bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1\uplus\Pi_2}\rangle \longmapsto \langle T', \bar{\sigma}'|_{\Pi_1\uplus\Pi_2}\rangle$, then either:
  - $T_1 = \mathbf{skip}$, $T_2 = \mathbf{skip}$, $T' = \mathbf{skip}$, and $\bar{\sigma}'|_{\Pi_1\uplus\Pi_2} = \bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1\uplus\Pi_2}$. Since both $\langle\!\langle\mathbf{skip}, \mathbf{skip}\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}$ and $\mathbf{skip}$ are 0-atomic, we need to show that $\mathsf{dom}(\bar{\sigma}') = \mathsf{dom}(\bar{\sigma}_1 \uplus \bar{\sigma}_2)$, which holds trivially, and that $I \models \langle\mathbf{skip}, \bar{\sigma}'\rangle \rhd_{n-1} (Q_1 * Q_2)$. From (a), by Def. 7.8 (item 5), we know (g) $\bar{\sigma}_1 \in Q_1$. From (b), by Def. 7.8 (item 5), we know (h) $\bar{\sigma}_2 \in Q_2$. From (g) and (h), we know (i) $\bar{\sigma}' \in Q_1 * Q_2$. From (i), using Lemma 7.11, we conclude
  - or, $T' = \langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}$ and (g) $\langle T_1, \bar{\sigma}_1 \uplus \bar{\sigma}_2|_{\Pi_1\uplus\Pi_2}\rangle \longmapsto \langle T_1', \bar{\sigma}'|_{\Pi_1\uplus\Pi_2}\rangle$. From (d) and (g), using Lemma 3.22 (item 3), we know that exists $\bar{\sigma}_1'$ such that $\bar{\sigma}' = \bar{\sigma}_1' \uplus \bar{\sigma}_2$ and (h) $\langle T_1, \bar{\sigma}_1|_{\Pi_1}\rangle \longmapsto \langle T_1', \bar{\sigma}_1'|_{\Pi_1}\rangle$. From (c), and (f), using Remark 3.17, we have 5 cases:
    * If both T_1 and T_1' are 1-atomic, and T_2 is 0-atomic, we have to show that $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}, \bar{\sigma}_1' \uplus \bar{\sigma}_2\rangle \rhd_{n-1} (Q_1 * Q_2)$. From (a) and (h), by Def. 7.8 (item 4.a), we know that (i) $I \models \langle T_1', \bar{\sigma}_1'\rangle \rhd_{n-1} Q_1$. From (b), using Lemma 7.9, we know that (j) $I \models \langle T_2, \bar{\sigma}_2\rangle \rhd_{n-1} Q_2$. From (i) and (j), using the induction hypothesis, we conclude
    * If both T_1 and T_1' are 0-atomic, and T_2 is 1-atomic, we have to show that $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}, \bar{\sigma}_1' \uplus \bar{\sigma}_2\rangle \rhd_{n-1} (Q_1 * Q_2)$. From (a) and (h), by Def. 7.8 (item 4.b), we know that (i) $\mathsf{dom}(\bar{\sigma}_1) = \mathsf{dom}(\bar{\sigma}_1')$ and (j) $I \models \langle T_1', \bar{\sigma}_1'\rangle \rhd_{n-1} Q_1$. From (b), using Lemma 7.9, we know that (k) $I \models \langle T_2, \bar{\sigma}_2\rangle \rhd_{n-1} Q_2$. From (j) and (k), using the induction hypothesis, we conclude
    * If all T_1, T_1', and T_2 are 0-atomic, we have to show that $\mathsf{dom}(\bar{\sigma}_1 \uplus \bar{\sigma}_2) = \mathsf{dom}(\bar{\sigma}_1' \uplus \bar{\sigma}_2)$ and $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}, \bar{\sigma}_1' \uplus \bar{\sigma}_2\rangle \rhd_{n-1} (Q_1 * Q_2)$. From (a) and (h), by Def. 7.8 (item 4.b), we know that (i) $\mathsf{dom}(\bar{\sigma}_1) = \mathsf{dom}(\bar{\sigma}_1')$ and (j) $I \models \langle T_1', \bar{\sigma}_1'\rangle \rhd_{n-1} Q_1$. From (b), using Lemma 7.9, we know that (k) $I \models \langle T_2, \bar{\sigma}_2\rangle \rhd_{n-1} Q_2$. From (j) and (k), using the induction hypothesis, we know (l) $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}, \bar{\sigma}_1' \uplus \bar{\sigma}_2\rangle \rhd_{n-1} (Q_1 * Q_2)$. From (i), we know that

(m) $\mathsf{dom}(\bar\sigma_1 \uplus \bar\sigma_2) = \mathsf{dom}(\bar\sigma_1' \uplus \bar\sigma_2)$. From (l) and (m), we conclude

  * If both T_1 and T_2 are 0-atomic, and T_1' is 1-atomic, we have to show that for all $\bar\sigma_1''$ and $\bar\sigma_2''$, such that (i) $\bar\sigma_2'' = \bar\sigma_1'' \uplus \bar\sigma'$ and (j) $\bar\sigma_1'' \in I$, we have $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathbf{p}}\mathbf{skip}, \bar\sigma_2''\rangle \rhd_{n-1} (Q_1 * Q_2)$. From (a), (j) and (h), by Def. 7.8 (item 4.c), we know that (k) $I \models \langle T_1', \bar\sigma_1'' \uplus \bar\sigma_1'\rangle \rhd_{n-1} Q_1$. From (b), using Lemma 7.9, we know that (l) $I \models \langle T_2, \bar\sigma_2\rangle \rhd_{n-1} Q_2$. From (k) and (l), by the induction hypothesis, we conclude
  * If both T_1' and T_2 are 0-atomic, and T_1 is 1-atomic, we have to show that exists $\bar\sigma_1''$ and $\bar\sigma_2''$, such that $\bar\sigma' = \bar\sigma_1'' \uplus \bar\sigma_2''$, $\bar\sigma_1'' \in I$, and $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathbf{p}}\mathbf{skip}, \bar\sigma_2''\rangle \rhd_{n-1} (Q_1 * Q_2)$. From (a) and (h), by Def. 7.8 (item 4.d), we know there exists $\bar\sigma_1'''$ and $\bar\sigma_2'''$ such that (i) $\bar\sigma_1' = \bar\sigma_1''' \uplus \bar\sigma_2'''$, (j) $\bar\sigma_1''' \in I$, and (k) $I \models \langle T_1', \bar\sigma_2'''\rangle \rhd_{n-1} Q_1$. From (b), using Lemma 7.9, we know that (l) $I \models \langle T_2, \bar\sigma_2\rangle \rhd_{n-1} Q_2$. From (k), and (l), by the induction hypothesis, we have (m) $I \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathbf{p}}\mathbf{skip}, \bar\sigma_2''' \uplus \bar\sigma_2\rangle \rhd_{n-1} (Q_1 * Q_2)$. Instantiating the goal with $\bar\sigma_1'''$ and $\bar\sigma_2''' \uplus \bar\sigma_2$, from (j), and (m), we conclude

  – or, $T' = \langle\!\langle T_1, T_2'\rangle\!\rangle_{\mathbf{p}}\mathbf{skip}$ and $\langle T_2, \bar\sigma_1 \uplus \bar\sigma_2|_{\Pi_1 \uplus \Pi_2}\rangle \longmapsto \langle T_2', \bar\sigma'|_{\Pi_1 \uplus \Pi_2}\rangle$. The proof is symmetric to the previous case

- $\langle\!\langle T_1, T_2\rangle\!\rangle_{\mathbf{p}}\mathbf{skip} \neq \mathbf{skip}$
- If $\langle\!\langle T_1, T_2\rangle\!\rangle_{\mathbf{p}}\mathbf{skip} = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$, we have 2 cases:
  – there exists $\mathbf{T}_1$ such that (d) $T_1 = \mathbf{T}_1[\,\mathbf{S}[\,a\,]\,]$ and (e) $\mathbf{T} = \langle\!\langle \mathbf{T}_1, T_2\rangle\!\rangle_{\mathbf{p}}\mathbf{skip}$. From (d) and (a), by Def. 7.8 (item 6), we conclude
  – there exists $\mathbf{T}_2$ such that (d) $T_2 = \mathbf{T}_2[\,\mathbf{S}[\,a\,]\,]$ and (e) $\mathbf{T} = \langle\!\langle T_1, \mathbf{T}_2\rangle\!\rangle_{\mathbf{p}}\mathbf{skip}$. From (d) and (b), by Def. 7.8 (item 6), we conclude □

An atomic block can be introduced using the following lemma.

Lemma 7.16. If $\mathsf{Emp} \models \langle T, \bar\sigma\rangle \rhd_n (I * Q)$, then $I \models \langle\langle\!\langle T\rangle\!\rangle_{\mathbf{a}}\mathbf{skip}, \bar\sigma\rangle \rhd_n Q$

Proof. By induction over n. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, by Def. 7.8, assuming (a) $\mathsf{Emp} \models \langle T, \bar\sigma\rangle \rhd_n (I * Q)$, we have 6 cases:

- By Def. 3.1, we know that (b) $\langle\!\langle T \rangle\!\rangle_{\mathsf{a}}\mathbf{skip}$ is 1-atomic
- From the semantics, if we assume $\langle \langle\!\langle T \rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{abort}$, then we know that (c) $\langle T, \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{abort}$. From (a), by Def. 7.8 (item 2), we know that (c) is false
- From the semantics, if we assume $\langle \langle\!\langle T \rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{race}$, then we know that (c) $\langle T, \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{race}$. From (a), by Def. 7.8 (item 3), we know that (c) is false
- If (c) $\langle \langle\!\langle T \rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle T', \bar{\sigma}'|_{\Pi} \rangle$, given (b) and from the semantics, we have 2 cases:
  - We have $T' = \langle\!\langle T'' \rangle\!\rangle_{\mathsf{a}}\mathbf{skip}$, which is 1-atomic, (d) $\langle T, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle T'', \bar{\sigma}'|_{\Pi} \rangle$, and we need to show that $I \models \langle \langle\!\langle T'' \rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \bar{\sigma}' \rangle \rhd_{n-1} Q$. From (a) and (d), by Def. 7.8 (items 4.a through 4.b), we know that (e) $\mathsf{Emp} \models \langle T'', \bar{\sigma}' \rangle \rhd_{n-1} (I {*} Q)$. From (e), by the induction hypothesis, we conclude
  - We have (d) $T = \mathbf{skip}$, $T' = \mathbf{skip}$ which is 0-atomic, $\bar{\sigma}|_{\Pi} = \bar{\sigma}'|_{\Pi}$, and we need to show that exists $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that $\bar{\sigma} = \bar{\sigma}_1 \uplus \bar{\sigma}_2$, $\bar{\sigma}_1 \in I$, and $I \models \langle \mathbf{skip}, \bar{\sigma}_2 \rangle \rhd_{n-1} Q$. From (a) and (d), by Def. 7.8 (item 5), we know that (e) $\bar{\sigma} \in I {*} Q$. From (e), we know that there exists $\bar{\sigma}_1'$ and $\bar{\sigma}_2'$ such that (f) $\bar{\sigma} = \bar{\sigma}_1' \uplus \bar{\sigma}_2'$, (g) $\bar{\sigma}_1' \in I$, and (h) $\bar{\sigma}_2' \in Q$. From (h), using Lemma 7.11, we know (i) $I \models \langle \mathbf{skip}, \bar{\sigma}_2' \rangle \rhd_{n-1} Q$. Instantiating the goal with $\bar{\sigma}_1'$ and $\bar{\sigma}_2'$, from (f), (g), and (h), we conclude
- We know that $\langle\!\langle T \rangle\!\rangle_{\mathsf{a}}\mathbf{skip} \neq \mathbf{skip}$
- If $\langle\!\langle T \rangle\!\rangle_{\mathsf{a}}\mathbf{skip} = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$, then we know that exists $\mathbf{T}'$ such that (c) $\mathbf{T} = \langle\!\langle \mathbf{T}' \rangle\!\rangle_{\mathsf{a}}\mathbf{skip}$ and (d) $T = \mathbf{T}'[\,\mathbf{S}[\,a\,]\,]$. From (d) and (a), by Def. 7.8 (item 6), we know that $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$ and conclude □

The following lemma is used for framing a triple into a larger shared memory.

Lemma 7.17. If $I \models \langle T, \bar{\sigma} \rangle \rhd_n Q$, then

1. If T is 0-atomic, then $I' {*} I \models \langle T, \bar{\sigma} \rangle \rhd_n Q$
2. If T is 1-atomic, and $\bar{\sigma}' \in I'$, then $I' {*} I \models \langle T, \bar{\sigma}' \uplus \bar{\sigma} \rangle \rhd_n Q$

Proof. By induction over n. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, assuming (a) $I \models \langle T, \bar{\sigma} \rangle \rhd_n Q$, we have 2 cases:

- If (b) T is 0-atomic, we need to show $I' {*} I \models \langle T, \bar{\sigma} \rangle \rhd_n Q$. By Def. 7.8, we have 6 cases:
  - From (b), we know that T is 0-atomic
  - From (a), by Def. 7.8 (item 2), we know that $\langle T, \bar{\sigma}|_{\Pi} \rangle \longmapsto \textsf{abort}$ is false
  - From (a), by Def. 7.8 (item 3), we know that $\langle T, \bar{\sigma}|_{\Pi} \rangle \longmapsto \textsf{race}$ is false
  - If (c) $\langle T, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle T', \bar{\sigma}'|_{\Pi} \rangle$, given (b) the we have 2 cases:
    * If T' is 0-atomic, we need to show that $\textsf{dom}(\bar{\sigma}) = \textsf{dom}(\bar{\sigma}')$ and $I' * I \models \langle T', \bar{\sigma}' \rangle \rhd_{n-1} Q$. From (a) and (c), by Def. 7.8 (item 4.b), we know that (d) $\textsf{dom}(\bar{\sigma}) = \textsf{dom}(\bar{\sigma}')$ and (e) $I \models \langle T', \bar{\sigma}' \rangle \rhd_{n-1} Q$. From (e), by the induction hypothesis (item 1), we know that (f) $I' {*} I \models \langle T', \bar{\sigma}' \rangle \rhd_{n-1} Q$. From (d) and (f) we conclude
    * If T' is 1-atomic, we need to show that for all $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that (d) $\bar{\sigma}_2 = \bar{\sigma}_1 \uplus \bar{\sigma}'$ and (e) $\bar{\sigma}_1 \in I' * I$, we have $I' * I \models \langle T', \bar{\sigma}_2 \rangle \rhd_{n-1} Q$. From (e), we know exists $\bar{\sigma}_1'$ and $\bar{\sigma}_1''$, such that $\bar{\sigma}_1 = \bar{\sigma}_1' \uplus \bar{\sigma}_1''$, (f) $\bar{\sigma}_1' \in I'$, and (g) $\bar{\sigma}_1'' \in I$. From (a), (c), and (g), by Def. 7.8 (item 4.c), we know that (h) $I \models \langle T', \bar{\sigma}_1'' \uplus \bar{\sigma}' \rangle \rhd_{n-1} Q$. From (h) and (f), by the induction hypothesis (item 2), we conclude
  - We assume (c) $T = \textbf{skip}$. From (a) and (c), by Def. 7.8 (item 5), we know $\bar{\sigma} \in Q$ and conclude
  - We assume (c) $T = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$. From (a) and (c), by Def. 7.8 (item 6), we know $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$ and conclude
- If (b) T is 1-atomic, and (c) $\bar{\sigma}' \in I'$, we need to show $I' * I \models \langle T, \bar{\sigma}' \uplus \bar{\sigma} \rangle \rhd_n Q$. By Def. 7.8, we have 6 cases:
  - From (b), we know that T is 1-atomic
  - From (a), by Def. 7.8 (item 2), we know that (e) $\langle T, \bar{\sigma}|_{\Pi} \rangle \longmapsto \textsf{abort}$ is false. From (e), using Lemma 3.22 (item 1), we conclude

– From (a), by Def. 7.8 (item 3), we know that (f) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \mathsf{race}$ is false. From (e) and (f), using Lemma 3.22 (item 2), we conclude

– We know (g) $\langle T, \bar{\sigma}' \uplus \bar{\sigma}|_{\Pi'\uplus\Pi}\rangle \longmapsto \langle T', \bar{\sigma}''|_{\Pi'\uplus\Pi}\rangle$. From (e) and (g), using Lemma 3.22 (item 3), we know exists $\bar{\sigma}'''$ such that (h) $\bar{\sigma}''|_{\Pi'\uplus\Pi} = \bar{\sigma}' \uplus \bar{\sigma}'''|_{\Pi'\uplus\Pi}$ and (i) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle T', \bar{\sigma}'''|_{\Pi}\rangle$. Then we have 2 cases:

  ∗ If T' is 1-atomic, we need to show that $I'{*}I \models \langle T', \bar{\sigma}' \uplus \bar{\sigma}'''\rangle \rhd_{n-1} Q$. From (a) and (i), by Def. 7.8 (item 4.a), we know that (j) $I \models \langle T', \bar{\sigma}'''\rangle \rhd_{n-1} Q$. From (j) and (c), by the induction hypothesis (item 2), we conclude

  ∗ If T' is 0-atomic, we need to show that exists $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that $\bar{\sigma}'' = \bar{\sigma}_1 \uplus \bar{\sigma}_2$, $\bar{\sigma}_1 \in I'{*}I$, and $I'{*}I \models \langle T', \bar{\sigma}_2\rangle \rhd_{n-1} Q$. From (a) and (i), by Def. 7.8 (item 4.d), we know there exists $\bar{\sigma}_1'$ and $\bar{\sigma}_2'$, such that (j) $\bar{\sigma}''' = \bar{\sigma}_1' \uplus \bar{\sigma}_2'$, (k) $\bar{\sigma}_1' \in I$, and (l) $I \models \langle T', \bar{\sigma}_2'\rangle \rhd_{n-1} Q$. From (l), by the induction hypothesis (item 1), we have (m) $I'{*}I \models \langle T', \bar{\sigma}_2'\rangle \rhd_{n-1} Q$. From (c), (h), (j), and (k), we have (n) $\bar{\sigma}' \uplus \bar{\sigma}_1' \in I'{*}I$. Instantiating the goal with $\bar{\sigma}' \uplus \bar{\sigma}_1'$ and $\bar{\sigma}_2'$, from (h), (j), (n), and (m), we conclude

– From (b), we know that $T \neq \mathbf{skip}$

– We assume (e) $T = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$. From (a) and (e), by Def. 7.8 (item 6), we know (f) $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$. From (f), using Remark 7.7, we conclude □

The following lemma is used to transfer a resource from shared to private in a triple.

Lemma 7.18. If $I'{*}I \models \langle T, \bar{\sigma}\rangle \rhd_n Q$, then

1. If T is 0-atomic, and $\bar{\sigma}' \in I'$, then $I \models \langle T, \bar{\sigma}' \uplus \bar{\sigma}\rangle \rhd_n (I'{*}Q)$
2. If T is 1-atomic, then $I \models \langle T, \bar{\sigma}\rangle \rhd_n (I'{*}Q)$

Proof. By induction over n. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, assuming (a) $I'{*}I \models \langle T, \bar{\sigma}\rangle \rhd_n Q$, we have 2 cases:

- If (b) T is 0-atomic, and (c) $\bar{\sigma}' \in I'$, we need to show $I \models \langle T, \bar{\sigma}' \uplus \bar{\sigma}\rangle \rhd_n (I'{*}Q)$. By Def. 6.3, we have 6 cases:

- From (b), we know that T is 0-atomic
- From (a), by Def. 7.8 (item 2), we know that (e) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \textsf{abort}$ is false. From (e), using Lemma 3.22 (item 1), we conclude
- From (a), by Def. 7.8 (item 3), we know that (f) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \textsf{race}$ is false. From (e) and (f), using Lemma 3.22 (item 2), we conclude
- We know (g) $\langle T, \bar{\sigma}' \uplus \bar{\sigma}|_{\Pi'\uplus\Pi}\rangle \longmapsto \langle T', \bar{\sigma}''|_{\Pi'\uplus\Pi}\rangle$. From (e) and (g), using Lemma 3.22 (item 3), we know exists $\bar{\sigma}'''$ such that (h) $\bar{\sigma}''|_{\Pi'\uplus\Pi} = \bar{\sigma}' \uplus \bar{\sigma}'''|_{\Pi'\uplus\Pi}$ and (i) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle T', \bar{\sigma}'''|_{\Pi}\rangle$. Then we have 2 cases:
  * If T' is 0-atomic, we need to show that $\textsf{dom}(\bar{\sigma}' \uplus \bar{\sigma}) = \textsf{dom}(\bar{\sigma}' \uplus \bar{\sigma}''')$ and $I \models \langle T', \bar{\sigma}' \uplus \bar{\sigma}'''\rangle \rhd_{n-1} (I' {*} Q)$. From (a) and (g), by Def. 7.8 (item 4.b), we know that (j) $\textsf{dom}(\bar{\sigma}) = \textsf{dom}(\bar{\sigma}''')$ and (k) $I' {*} I \models \langle T', \bar{\sigma}'''\rangle \rhd_{n-1} Q$. From (k) and (c), by the induction hypothesis (item 1), we know (l) $I \models \langle T', \bar{\sigma}' \uplus \bar{\sigma}'''\rangle \rhd_{n-1} (I' {*} Q)$. From (j) and (l), we conclude
  * If T' is 1-atomic, we need to show that for all $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that (j) $\bar{\sigma}_2 = \bar{\sigma}_1 \uplus \bar{\sigma}' \uplus \bar{\sigma}'''$ and (k) $\bar{\sigma}_1 \in I$, we have $I \models \langle T', \bar{\sigma}_2\rangle \rhd_{n-1} (I' {*} Q)$. From (c) and (k), we know (l) $\bar{\sigma}' \uplus \bar{\sigma}_1 \in I' {*} I$. From (a), (i), (j), and (l), by Def. 7.8 (item 4.c), we know that (m) $I' {*} I \models \langle T', \bar{\sigma}_2\rangle \rhd_{n-1} Q$. From (m), by the induction hypothesis (item 2), we conclude
- We assume (e) $T = \textbf{skip}$. From (a) and (e), by Def. 7.8 (item 5), we know that (f) $\bar{\sigma} \in Q$. From (c) and (f) we know that $\bar{\sigma}' \uplus \bar{\sigma} \in I' {*} Q$ and conclude
- We assume (e) $T = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$. From (a) and (e), by Def. 7.8 (item 6), we know (f) $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$. From (f), using Remark 7.7, we conclude

• If (b) T is 1-atomic, we need to show $I \models \langle T, \bar{\sigma}\rangle \rhd_n (I' {*} Q)$. By Def. 7.8, we have 6 cases:

- From (b), we know that T is 1-atomic
- From (a), by Def. 7.8 (item 2), we know that $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \textsf{abort}$ is false
- From (a), by Def. 7.8 (item 3), we know that $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \textsf{race}$ is false

- If (c) $\langle T, \bar{\sigma}|_\Pi \rangle \longmapsto \langle T', \bar{\sigma}'|_\Pi \rangle$, given (b) the we have 2 cases:
    - If T' is 1-atomic, we need to show that $I \models \langle T', \bar{\sigma}' \rangle \rhd_{n-1} (I' * Q)$. From (a) and (c), by Def. 7.8 (item 4.a), we know that (d) $I' * I \models \langle T', \bar{\sigma}' \rangle \rhd_{n-1} Q$. From (d), by the induction hypothesis (item 2), we conclude
    - If T' is 0-atomic, we need to show that exists $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that $\bar{\sigma}' = \bar{\sigma}_1 \uplus \bar{\sigma}_2$, $\bar{\sigma}_1 \in I$, and $I \models \langle T', \bar{\sigma}_2 \rangle \rhd_{n-1} (I' * Q)$. From (a) and (c), by Def. 7.8 (item 4.d), we know there exists $\bar{\sigma}_1'$ and $\bar{\sigma}_2'$, such that (d) $\bar{\sigma}' = \bar{\sigma}_1' \uplus \bar{\sigma}_2'$, (e) $\bar{\sigma}_1' \in I' * I$, and (f) $I' * I \models \langle T', \bar{\sigma}_2' \rangle \rhd_{n-1} Q$. From (e) we know exists $\bar{\sigma}_1''$ and $\bar{\sigma}_1'''$ such that (g) $\bar{\sigma}_1' = \bar{\sigma}_1'' \uplus \bar{\sigma}_1'''$, (h) $\bar{\sigma}_1'' \in I'$, and (i) $\bar{\sigma}_1''' \in I$. From (f) and (h), by the induction hypothesis (item 1), we have (j) $I \models \langle T', \bar{\sigma}_1'' \uplus \bar{\sigma}_2' \rangle \rhd_{n-1} (I' * Q)$. Instantiating the goal with $\bar{\sigma}_1'''$ and $\bar{\sigma}_1'' \uplus \bar{\sigma}_2'$, from (d), (g), (i), and (j), we conclude
- From (b), we know that $T \neq \textbf{skip}$
- We assume (c) $T = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$. From (a) and (c), by Def. 7.8 (item 6), we know $\Delta^a_{(\bar{\sigma}|_\Pi)} \subseteq \nabla(\bar{\sigma})$ and conclude □

The following lemma is used for the conjunction of triples.

Lemma 7.19. If $I \models \langle T, \bar{\sigma} \rangle \rhd_n Q_1$, $I \models \langle T, \bar{\sigma} \rangle \rhd_n Q_2$, and I is precise, then $I \models \langle T, \bar{\sigma} \rangle \rhd_n (Q_1 \cap Q_2)$

Proof. By induction over n. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, by Def. 7.8, assuming (a) $I \models \langle T, \bar{\sigma} \rangle \rhd_n Q_1$ and (b) $I \models \langle T, \bar{\sigma} \rangle \rhd_n Q_2$, we have 6 cases:

- From (a), by Def. 7.8 (item 1), we know that T is 0- or 1-atomic
- From (a), by Def. 7.8 (item 2), we know that $\langle T, \bar{\sigma}|_\Pi \rangle \longmapsto \mathsf{abort}$ is false
- From (a), by Def. 7.8 (item 3), we know that $\langle T, \bar{\sigma}|_\Pi \rangle \longmapsto \mathsf{race}$ is false
- If (c) $\langle T, \bar{\sigma}|_\Pi \rangle \longmapsto \langle T', \bar{\sigma}'|_\Pi \rangle$, then we have 4 cases:
    - If both T and T' are 1-atomic, we need to show that $I \models \langle T', \bar{\sigma}' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (c), by Def. 7.8 (item 4.a), we know that (d) $I \models \langle T', \bar{\sigma}' \rangle \rhd_{n-1} Q_1$.

From (b) and (c), by Def. 7.8 (item 4.a), we know that (e) $I \models \langle T', \bar{\sigma}' \rangle \rhd_{n-1} Q_2$. From (d) and (e), by the induction hypothesis, we conclude

  – If both T and T' are 0-atomic, we need to show that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$ and $I \models \langle T', \bar{\sigma}' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (c), by Def. 7.8 (item 4.b), we know already that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$, and also that (d) $I \models \langle T', \bar{\sigma}' \rangle \rhd_{n-1} Q_1$. From (b) and (c), by Def. 7.8 (item 4.b), we also know that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$ and (e) $I \models \langle T', \bar{\sigma}' \rangle \rhd_{n-1} Q_2$. From (d) and (e), by the induction hypothesis, we conclude

  – If T is 0-atomic and T' is 1-atomic, we need to show that for all $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that (d) $\bar{\sigma}_2 = \bar{\sigma}_1 \uplus \bar{\sigma}'$ and (e) $\bar{\sigma}_1 \in I$, we have $I \models \langle T', \bar{\sigma}_2 \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a), (c), (d), and (e), by Def. 7.8 (item 4.c), we know that (f) $I \models \langle T', \bar{\sigma}_2 \rangle \rhd_{n-1} Q_1$. From (b), (c), (d), and (e), by Def. 7.8 (item 4.c), we know that (g) $I \models \langle T', \bar{\sigma}_2 \rangle \rhd_{n-1} Q_2$. From (f) and (g), by the induction hypothesis, we conclude

  – If T is 1-atomic and T' is 0-atomic, we need to show that exists $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that $\bar{\sigma}' = \bar{\sigma}_1 \uplus \bar{\sigma}_2$, $\bar{\sigma}_1 \in I$, and $I \models \langle T', \bar{\sigma}_2 \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (c), by Def. 7.8 (item 4.d), we know there exists $\bar{\sigma}_1'$ and $\bar{\sigma}_2'$, such that (d) $\bar{\sigma}' = \bar{\sigma}_1' \uplus \bar{\sigma}_2'$, (e) $\bar{\sigma}_1' \in I$, and (f) $I \models \langle T', \bar{\sigma}_2' \rangle \rhd_{n-1} Q_1$. From (b) and (c), by Def. 7.8 (item 4.d), we know there exists $\bar{\sigma}_1''$ and $\bar{\sigma}_2''$, such that (g) $\bar{\sigma}' = \bar{\sigma}_1'' \uplus \bar{\sigma}_2''$, (h) $\bar{\sigma}_1'' \in I$, and (i) $I \models \langle T', \bar{\sigma}_2'' \rangle \rhd_{n-1} Q_2$. From (d), (e), (g), and (h), given that I is precise, we know that $\bar{\sigma}_1' = \bar{\sigma}_1''$ and $\bar{\sigma}_2' = \bar{\sigma}_2''$. From (f) and (i), by the induction hypothesis, we have (j) $I \models \langle T', \bar{\sigma}_2' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. Instantiating the goal with $\bar{\sigma}_1'$ and $\bar{\sigma}_2'$, from (d), (e), and (j), we conclude

- We assume (c) $T = \mathbf{skip}$. From (a) and (c), by Def. 7.8 (item 5), we know that (d) $\bar{\sigma} \in Q_1$. From (b) and (c), by Def. 7.8 (item 5), we know that (e) $\bar{\sigma} \in Q_2$. From (d) and (e), we know that $\bar{\sigma} \in Q_1 \cap Q_2$ and conclude

- We assume (c) $T = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$. From (a) and (c), by Def. 7.8 (item 6), we know $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$ and conclude □

Semantics rules. The quadruple $I \models \{P\}\, c\, \{Q\}$ is the semantic correspondent to $I \vdash \{P\}\, c\, \{Q\}$. It is defined in terms of $I \models \langle T, \bar{\sigma}\rangle \triangleright Q$ as show below:

Definition 7.20. $I \models \{P\}\, c\, \{Q\}$, if and only if, for all $\bar{\sigma}$, such that $\bar{\sigma} \in P$, we have $I \models \langle c, \bar{\sigma}\rangle \triangleright Q$

From Def. 7.20, we can prove Lemma 7.21 which states more explicitly the properties guaranteed by the semantic quadruple: safety, race-freedom, and partial correctness (items 1, 2, and 3, respectively).

Lemma 7.21. If $I \models \{P\}\, c\, \{Q\}$, then for all $\bar{\sigma}$, such that $\bar{\sigma} \in I * P$, we have:

1. $\neg\langle c, \bar{\sigma}|_{\Pi}\rangle \longmapsto^{*} \textsf{abort}$

2. $\neg\langle c, \bar{\sigma}|_{\Pi}\rangle \longmapsto^{*} \textsf{race}$

3. If $\langle c, \bar{\sigma}|_{\Pi}\rangle \longmapsto^{*} \langle \textbf{skip}, \bar{\sigma}'|_{\Pi}\rangle$, then $\bar{\sigma}' \in I * Q$

Proof. From $I \models \{P\}\, c\, \{Q\}$, using Lemma 7.33[1], we obtain (a) $\textsf{Emp} \models \{I{*}P\}\, c\, \{I{*}Q\}$. Given (a), and $\bar{\sigma} \in I * P$, from Def. 7.20, we obtain (b) $\textsf{Emp} \models \langle c, \bar{\sigma}\rangle \triangleright (I * Q)$. We then generalize the proof from command c to any 0- or 1-atomic thread tree T. Now we can consider each one of the goals:

- For goal 1, we need to show that for all n, (c) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto^{n} \textsf{abort}$ is false. From (b), we obtain (d) $\textsf{Emp} \models \langle T, \bar{\sigma}\rangle \triangleright_{n} (I * Q)$, which is our sole assumption. By induction over n, we have two cases. The base case, where $n = 0$, is trivial as $\langle T, \bar{\sigma}|_{\pi}\rangle \neq \textsf{abort}$. In the inductive case, where $n > 0$, we know there exists a configuration κ, such that (e) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \kappa$ and (f) $\kappa \longmapsto^{n-1} \textsf{abort}$. From (d), given that $n > 0$, we know, from items 2, 3, and 6, of Def. 7.8, that by exclusion, and using Remark 7.6, there must exists T' and $\bar{\sigma}'$, such that $\kappa = \langle T', \bar{\sigma}'|_{\Pi}\rangle$; therefore we obtain (g) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle T', \bar{\sigma}'|_{\Pi}\rangle$. From (g), using Remark 3.17, we know that T' is either 0- or 1-atomic. Given (g), and (d), from items 4.a through 4.d of Def. 7.8, we know that (h) $\textsf{Emp} \models \langle T', \bar{\sigma}'\rangle \triangleright_{n-1} (I{*}Q)$. From (h), and (f), using the induction hypothesis, we know that

[1]Although Lemma 7.33 is defined later on the text, there is no circularity.

(i) $\langle T', \bar{\sigma}'|_{\Pi}\rangle \longmapsto^{n-1}$ abort is false. From (g), and (i), we know $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto^{n-1}$ abort is false, which was our goal.

- For goal 2, we need to show that for all n, (c) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto^{n}$ race is false. From (b), we obtain (d) $\mathsf{Emp} \models \langle T, \bar{\sigma}\rangle \rhd_n (I{*}Q)$, which is our sole assumption. By induction over n, we have two cases. The base case, where $n = 0$, is trivial as $\langle T, \bar{\sigma}|_{\Pi}\rangle \neq$ race. In the inductive case, where $n > 0$, we know there exists a configuration κ, such that (e) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \kappa$ and (f) $\kappa \longmapsto^{n-1}$ race. From (d), given that $n > 0$, we know, from items 2, 3, and 6, of Def. 7.8, that by exclusion, and using Remark 7.6, there must exists T' and $\bar{\sigma}'$, such that $\kappa = \langle T', \bar{\sigma}'|_{\Pi}\rangle$; therefore we obtain (g) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle T', \bar{\sigma}'|_{\Pi}\rangle$. From (g), using Remark 3.17, we know that T' is either 0- or 1-atomic. Given (g), and (d), from items 4.a through 4.d of Def. 7.8, we know that (h) $\mathsf{Emp} \models \langle T', \bar{\sigma}'\rangle \rhd_{n-1} (I{*}Q)$. From (h), and (f), using the induction hypothesis, we know that (i) $\langle T', \bar{\sigma}'|_{\Pi}\rangle \longmapsto^{n-1}$ race is false. From (g), and (i), we know $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto^{n-1}$ race is false, which was our goal.

- For goal 3, we need to show that for all n, if (c) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto^{n} \langle \textbf{skip}, \bar{\sigma}'|_{\Pi}\rangle$, then $\bar{\sigma}' \in I{*}Q$. From (b), we obtain (d) $\mathsf{Emp} \models \langle T, \bar{\sigma}\rangle \rhd_n (I{*}Q)$, which is our sole assumption. By induction over n, we have two cases. In base case, where $n = 0$, we know that $T = \textbf{skip}$ and $\bar{\sigma}'|_{\Pi} = \bar{\sigma}|_{\Pi}$; given (d), from item 5 of Def. 7.8, we obtain the goal $\bar{\sigma}' \in I{*}Q$. In the inductive case, where $n > 0$, we know there exists a configuration κ, such that (e) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \kappa$ and (f) $\kappa \longmapsto^{n-1} \langle \textbf{skip}, \bar{\sigma}'|_{\Pi}\rangle$. From (d), given that $n > 0$, we know, from items 2, 3, and 6, of Def. 6.3, that by exclusion, and using Remark 7.6, there must exists T' and $\bar{\sigma}''$, such that $\kappa = \langle T', \bar{\sigma}''|_{\Pi}\rangle$; therefore we obtain (g) $\langle T, \bar{\sigma}|_{\Pi}\rangle \longmapsto \langle T', \bar{\sigma}''|_{\Pi}\rangle$. From (g), using Remark 3.17, we know that T' is either 0- or 1-atomic. Given (g), and (d), from items 4.a through 4.d of Def. 7.8, we know that (h) $\mathsf{Emp} \models \langle T', \bar{\sigma}''\rangle \rhd_{n-1} (I{*}Q)$. From (h), and (f), using the induction hypothesis, we obtain the goal $\bar{\sigma}' \in I{*}Q$. $\square$

In the following sequence of lemmas, Lemma 7.22 through Lemma 7.35, we show the correspondence between the CSL rules from Fig. 7.4, and their semantic equivalent using definition Def. 7.20.

Lemma 7.22.

$$\overline{\mathsf{Emp} \models \{Q \circ [\![\nu := e]\!]\}\, \nu := e\, \{Q\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in Q \circ [\![\nu := e]\!]$, and we need to show that $\mathsf{Emp} \models \langle \nu := e, \bar{\sigma}\rangle \rhd_n Q$. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, by Def. 7.8, we have 6 cases:

- By Def. 3.1, we know that (b) $\nu := e$ is 0-atomic
- From the semantics, we know that $\langle \nu := e, \bar{\sigma}|_\Pi\rangle \longmapsto \mathsf{abort}$ is false if $\langle \nu := e, \bar{\sigma}|_\Pi\rangle \longrightarrow \mathsf{abort}$ is false. From the sequential semantics, we know $\langle \nu := e, \bar{\sigma}|_\Pi\rangle \longrightarrow \mathsf{abort}$ is false if there exists σ' such that $(\bar{\sigma}|_\Pi, \sigma') \in [\![a]\!]$. From (a), by Def. 7.5 (item 1), we have (c) $\Delta^a_{(\bar{\sigma}|_\Pi)} \subseteq \nabla(\bar{\sigma})$. From (c), using Remark 7.6, we conclude
- From the semantics, we know that $\langle \nu := e, \bar{\sigma}|_\Pi\rangle \longmapsto \mathsf{race}$ is false
- From the semantics, assuming $\langle \nu := e, \bar{\sigma}|_\Pi\rangle \longmapsto \langle \mathbf{skip}, \bar{\sigma}'|_\Pi\rangle$, we have (c) $\langle \nu := e, \bar{\sigma}|_\Pi\rangle \longrightarrow \langle \mathbf{skip}, \bar{\sigma}'|_\Pi\rangle$. From (b), and since **skip** is 0-atomic, we need to show that $\mathsf{dom}(\bar{\sigma}) = \mathsf{dom}(\bar{\sigma}')$ and $\mathsf{Emp} \models \langle \mathbf{skip}, \bar{\sigma}'\rangle \rhd_{n-1} Q$. From the sequential semantics, and (c), we know (d) $(\bar{\sigma}|_\Pi, \bar{\sigma}'|_\Pi) \in [\![\nu := e]\!]$. From (d), using Remark 3.13, we know (e) $\mathsf{dom}(\bar{\sigma}|_\Pi) = \mathsf{dom}(\bar{\sigma}'|_\Pi)$. From (a) and (d), by Def. 7.5 (item 2), we know that (f) $\bar{\sigma}' \in Q$. From (f), and Lemma 7.11, we know (g) $\mathsf{Emp} \models \langle \mathbf{skip}, \bar{\sigma}'\rangle \rhd_{n-1} Q$. From (e) and (g) we conclude
- We know that $(\nu := e) \neq \mathbf{skip}$
- If (c) $(\nu := e) = \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$, we know that (d) $\mathbf{T} = \bullet$, (e) $\mathbf{S} = \bullet$, and (f) $a = (\nu := e)$. From (a), by Def. 7.5 (item 1), we conclude $\square$

Lemma 7.23.

$$\overline{\mathsf{Emp} \models \{Q \circ [\![a]\!]\}\, \langle a\rangle\, \{Q\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in Q \circ [\![a]\!]$, and we need to show that $\mathsf{Emp} \models \langle \mathbf{atomic}\ a, \bar{\sigma} \rangle \rhd_n Q$. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, by Def. 7.8, we have 6 cases:

- By Def. 3.1, we know that (b) $\mathbf{atomic}\ a$ is 0-atomic
- From the semantics, we know that $\langle \mathbf{atomic}\ a, \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{abort}$ is false
- From the semantics, we know that $\langle \mathbf{atomic}\ a, \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{race}$ is false
- From the semantics, we know that $\langle \mathbf{atomic}\ a, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}|_{\Pi} \rangle$, given that $\bullet$ is 0-atomic. From (b), and since $\langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}$ is 1-atomic, we need to show that for all $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that (c) $\bar{\sigma}_2 = \bar{\sigma}_1 \uplus \bar{\sigma}$ and (d) $\bar{\sigma}_1 \in \mathsf{Emp}$, $\mathsf{Emp} \models \langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}_2 \rangle \rhd_{n-1} Q$. From (d), we know $\bar{\sigma}_1 = \varnothing$, therefore, from (c), we know $\bar{\sigma}_2 = \bar{\sigma}$. If $n = 1$, by Def. 7.8, we conclude. If $n > 1$, by Def. 7.8, we have 6 cases:
  - By Def. 3.1, we know that (e) $\langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}$ is 1-atomic
  - From the semantics, we know that $\langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{abort}$ is false if $\langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}|_{\Pi} \rangle \longrightarrow \mathsf{abort}$ is false. From the sequential semantics, we know $\langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}|_{\Pi} \rangle \longrightarrow \mathsf{abort}$ is false if there exists σ' such that $(\bar{\sigma}|_{\Pi}, \sigma') \in [\![a]\!]$. From (a), by Def. 7.5 (item 1), we have (f) $\Delta^{a}_{(\bar{\sigma}|_{\Pi})} \subseteq \nabla(\bar{\sigma})$. From (f), using Remark 7.6, we conclude
  - From the semantics, we know that $\langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}|_{\Pi} \rangle \longmapsto \mathsf{race}$ is false
  - From the semantics, assuming $\langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}|_{\Pi} \rangle \longmapsto \langle \langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}'|_{\Pi} \rangle$, we have (f) $\langle a, \bar{\sigma}|_{\Pi} \rangle \longrightarrow \langle \mathbf{skip}, \bar{\sigma}'|_{\Pi} \rangle$. From (e), and since $\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}$ is 1-atomic, we need to show that $\mathsf{Emp} \models \langle \langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}' \rangle \rhd_{(n-2)} Q$. If $n = 2$, by Def. 7.8, we conclude. If $n > 2$, by Def. 7.8, we have 6 cases:
    * By Def. 3.1, we know that (g) $\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}$ is 1-atomic
    * From the semantics, we know that $\langle \langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}'|_{\Pi} \rangle \longmapsto \mathsf{abort}$ is false
    * From the semantics, we know that $\langle \langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}'|_{\Pi} \rangle \longmapsto \mathsf{race}$ is false
    * From the semantics, we know that $\langle \langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}'|_{\Pi} \rangle \longmapsto \langle \mathbf{skip}, \bar{\sigma}'|_{\Pi} \rangle$. From (g), and since $\mathbf{skip}$ is 0-atomic, we need to show there exists $\bar{\sigma}'_1$ and

$\bar{\sigma}_2'$ such that $\bar{\sigma}' = \bar{\sigma}_1' \uplus \bar{\sigma}_2'$, $\bar{\sigma}_1' \in \mathsf{Emp}$, and $\mathsf{Emp} \models \langle \mathbf{skip}, \bar{\sigma}_2' \rangle \rhd_{(n-3)} Q$. We instantiate $\bar{\sigma}_1'$ as $\varnothing$ and $\bar{\sigma}_2'$ as $\bar{\sigma}'$, as we know that $\bar{\sigma}' = \varnothing \uplus \bar{\sigma}'$ and $\varnothing \in \mathsf{Emp}$; it remains to show that $\mathsf{Emp} \models \langle \mathbf{skip}, \bar{\sigma}' \rangle \rhd_{(n-3)} Q$. From the sequential semantics, and (f), we know (h) $(\bar{\sigma}|_\Pi, \bar{\sigma}'|_\Pi) \in [\![a]\!]$. From (a) and (h), by Def. 7.5 (item 2), we know that (i) $\bar{\sigma}' \in Q$. From (i), and Lemma 7.11, we conclude

    * We know that $\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} \mathbf{skip} \neq \mathbf{skip}$
    * We know that $\langle\!\langle \mathbf{skip} \rangle\!\rangle_{\mathsf{a}} \mathbf{skip} \neq \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$

  – We know that $\langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \mathbf{skip} \neq \mathbf{skip}$
  – If (f) $\langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \mathbf{skip} = \mathbf{T}[\,\mathbf{S}[\,a'\,]\,]$, we know that (g) $\mathbf{T} = \langle\!\langle \bullet \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}$, (h) $\mathbf{S} = \bullet$, and (i) $a = a'$. From (a), by Def. 7.5 (item 1), we conclude

- We know that **atomic** $a \neq \mathbf{skip}$
- We know that **atomic** $a \neq \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$ □

Lemma 7.24.

$$\frac{I \models \{P\}\, c_1\, \{P'\} \quad I \models \{P'\}\, c_2\, \{Q\}}{I \models \{P\}\, c_1; c_2\, \{Q\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in P$, and we need to show that $I \models \langle c_1; c_2, \bar{\sigma} \rangle \rhd_n Q$. From (a), and $I \models \{P\}\, c_1\, \{P'\}$, by Def. 7.20, we know that (b) $I \models \langle c_1, \bar{\sigma} \rangle \rhd_n P'$. From $I \models \{P'\}\, c_2\, \{Q\}$, by Def. 7.20, we know that (c) for all $\bar{\sigma}' \in P'$, we have $I \models \langle c_2, \bar{\sigma}' \rangle \rhd_n Q$. From (b) and (c), using Lemma 7.12 (item 1), we conclude □

Lemma 7.25.

$$\frac{}{\mathsf{Emp} \models \{P\}\, \mathbf{skip}\, \{P\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in P$, and we need to show that $\mathsf{Emp} \models \langle \mathbf{skip}, \bar{\sigma} \rangle \rhd_n P$. From (a), using Lemma 7.11, we conclude □

Lemma 7.26.

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad I \models \{P \cap \lfloor b \rfloor\}\, c_1\, \{Q\} \quad I \models \{P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q\}}{I \models \{P\}\, \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\, \{Q\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in P$, and we need to show that $I \models \langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \bar{\sigma}\rangle \rhd_n Q$. From (a), and $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, we know (b) $\bar{\sigma} \in \lfloor b \rfloor \cup \lfloor \neg b \rfloor$. From (b) we can consider two cases:

- If (c) $\bar{\sigma} \in \lfloor b \rfloor$, with (a), we know (d) $\bar{\sigma} \in P \cap \lfloor b \rfloor$. From (d), and $I \models \{P \cap \lfloor b \rfloor\}\, c_1\, \{Q\}$, by Def. 7.20, we know that (e) $I \models \langle c_1, \bar{\sigma}\rangle \rhd_n Q$. From (e) and (c), using Lemma 7.13 (item 1), we conclude

- If (c) $\bar{\sigma} \in \lfloor \neg b \rfloor$, with (a), we know (d) $\bar{\sigma} \in P \cap \lfloor \neg b \rfloor$. From (d), and $I \models \{P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q\}$, by Def. 7.20, we know that (e) $I \models \langle c_2, \bar{\sigma}\rangle \rhd_n Q$. From (e) and (c), using Lemma 7.13 (item 2), we conclude □

Lemma 7.27.

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad I \models \{P \cap \lfloor b \rfloor\}\, c\, \{P\}}{I \models \{P\}\, \textbf{while } b \textbf{ do } c\, \{P \cap \lfloor \neg b \rfloor\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in P$, and we need to show that $I \models \langle \textbf{while } b \textbf{ do } c, \bar{\sigma}\rangle \rhd_n (P \cap \lfloor \neg b \rfloor)$. From $I \models \{P \cap \lfloor b \rfloor\}\, c\, \{P\}$, by Def. 7.20, we know that (b) for all $\bar{\sigma}' \in P \cap \lfloor b \rfloor$, we have $I \models \langle c, \bar{\sigma}'\rangle \rhd_n P$. From (a), (b), and $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, using Lemma 7.14, we conclude □

Lemma 7.28.

$$\frac{I \models \{P_1\}\, c_1\, \{Q_1\} \quad I \models \{P_2\}\, c_2\, \{Q_2\}}{I \models \{P_1 * P_2\}\, c_1 \parallel c_2\, \{Q_1 * Q_2\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in P_1 * P_2$, and we need to show that $I \models \langle c_1 \parallel c_2, \bar{\sigma}\rangle \rhd_n (Q_1 * Q_2)$. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, by Def. 7.8, we have 6 cases:

- By Def. 3.1, we know that (b) $c_1 \parallel c_2$ is 0-atomic

- From the semantics, we know that $\langle c_1 \parallel c_2, \bar{\sigma}|_\Pi\rangle \longmapsto \textsf{abort}$ is false

- From the semantics, we know that $\langle c_1 \parallel c_2, \bar{\sigma}|_\Pi\rangle \longmapsto \textsf{race}$ is false

- From the semantics, we know that $\langle c_1 \,\|\, c_2, \bar{\sigma}|_\Pi \rangle \longmapsto \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathsf{p}} \mathbf{skip}, \bar{\sigma}|_\Pi \rangle$. By Def. 3.1, we know (c) $\langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathsf{p}} \mathbf{skip}$ is 0-atomic. From (b) and (c), we need to show that $\bar{\sigma}|_\Pi = \bar{\sigma}|_\Pi$, which is trivial, and that $I \models \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathsf{p}} \mathbf{skip}, \bar{\sigma} \rangle \rhd_{n-1} (Q_1 * Q_2)$. From (a), we know there exists $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that $\bar{\sigma} = \bar{\sigma}_1 \uplus \bar{\sigma}_2$, (d) $\bar{\sigma}_1 \in P_1$, and (e) $\bar{\sigma}_2 \in P_2$. From (d), and $I \models \{P_1\}\, c_1 \,\{Q_1\}$, by Def. 7.20, we know that (f) $I \models \langle c_1, \bar{\sigma}_1 \rangle \rhd_{n-1} Q_1$. From (e), and $I \models \{P_2\}\, c_2 \,\{Q_2\}$, by Def. 7.20, we know that (g) $I \models \langle c_2, \bar{\sigma}_2 \rangle \rhd_{n-1} Q_2$. From (f), (g), and (c), using Lemma 7.15, we conclude
- We know that $c_1 \,\|\, c_2 \neq \mathbf{skip}$
- We know that $c_1 \,\|\, c_2 \neq \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$ □

Lemma 7.29.

$$\frac{\mathsf{Emp} \models \{I * P\}\, c \,\{I * Q\}}{I \models \{P\}\, \textbf{atomic}\, c \,\{Q\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in P$, and we need to show that $I \models \langle \textbf{atomic}\, c, \bar{\sigma} \rangle \rhd_n Q$. If $n = 0$, by Def. 7.8, we conclude. If $n > 0$, by Def. 7.8, we have 6 cases:

- By Def. 3.1, we know that (b) **atomic** c is 0-atomic
- From the semantics, we know that $\langle \textbf{atomic}\, c, \bar{\sigma}|_\Pi \rangle \longmapsto \mathsf{abort}$ is false
- From the semantics, we know that $\langle \textbf{atomic}\, c, \bar{\sigma}|_\Pi \rangle \longmapsto \mathsf{race}$ is false
- From the semantics, we know that $\langle \textbf{atomic}\, c, \bar{\sigma}|_\Pi \rangle \longmapsto \langle \langle\!\langle c \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}|_\Pi \rangle$, given that $\bullet$ is 0-atomic. From (b), and since $\langle\!\langle c \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}$ is 1-atomic, we need to show that for all $\bar{\sigma}_1$ and $\bar{\sigma}_2$, such that (c) $\bar{\sigma}_2 = \bar{\sigma}_1 \uplus \bar{\sigma}$ and (d) $\bar{\sigma}_1 \in I$, $I \models \langle \langle\!\langle c \rangle\!\rangle_{\mathsf{a}} \mathbf{skip}, \bar{\sigma}_2 \rangle \rhd_{n-1} Q$. From (a), (c), and (d), we know (e) $\bar{\sigma}_2 \in I * P$. From (e), and $\mathsf{Emp} \models \{I * P\}\, c \,\{I * Q\}$, by Def. 7.20, we know that (f) $\mathsf{Emp} \models \langle c, \bar{\sigma}_2 \rangle \rhd_{n-1} (I * Q)$. From (f), using Lemma 7.16, we conclude
- We know that **atomic** $c \neq \mathbf{skip}$
- We know that **atomic** $c \neq \mathbf{T}[\,\mathbf{S}[\,a\,]\,]$ □

Lemma 7.30.

$$\frac{P \subseteq P' \quad I \models \{P'\}\, c\, \{Q'\} \quad Q' \subseteq Q}{I \models \{P\}\, c\, \{Q\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in P$, and we need to show that $I \models \langle c, \bar{\sigma}\rangle \rhd_n Q$. From (a), and $P \subseteq P'$, we get (b) $\bar{\sigma} \in P'$. From (b), and $I \models \{P'\}\, c\, \{Q'\}$, by Def. 7.20, we know that (c) $I \models \langle c, \bar{\sigma}\rangle \rhd_n Q'$. From (c), and $Q' \subseteq Q$, using Lemma 7.10, we conclude □

Lemma 7.31.

$$\frac{\forall x.\ I \models \{\mathcal{P}(x)\}\, c\, \{\mathcal{Q}(x)\}}{I \models \{\exists x.\ \mathcal{P}(x)\}\, c\, \{\exists x.\ \mathcal{Q}(x)\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$, n, and x, such that (a) $\bar{\sigma} \in \mathcal{P}(x)$, and we need to show that $I \models \langle c, \bar{\sigma}\rangle \rhd_n (\exists x.\ \mathcal{Q}(x))$. From (a), and $\forall x.\ I \models \{\mathcal{P}(x)\}\, c\, \{\mathcal{Q}(x)\}$, by Def. 7.20, we know that (b) $I \models \langle c, \bar{\sigma}\rangle \rhd_n (\mathcal{Q}(x))$. We also know that (c) $(\mathcal{Q}(x)) \subseteq (\exists x.\ \mathcal{Q}(x))$. From (b) and (c), using Lemma 7.10, we conclude □

Lemma 7.32.

$$\frac{I \models \{P\}\, c\, \{Q\}}{I' {*} I \models \{P\}\, c\, \{Q\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in P$, and we need to show that $I' {*} I \models \langle c, \bar{\sigma}\rangle \rhd_n Q$. From (a), and $I \models \{P\}\, c\, \{Q\}$, by Def. 7.20, we know that (b) $I \models \langle c, \bar{\sigma}\rangle \rhd_n Q$. We also know that (c) c is 0-atomic. From (b), and (c), using Lemma 7.17 (item 1), we conclude □

Lemma 7.33.

$$\frac{I' {*} I \models \{P\}\, c\, \{Q\}}{I \models \{I' {*} P\}\, c\, \{I' {*} Q\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in I' {*} P$, and we need to show that $I \models \langle c, \bar{\sigma}\rangle \rhd_n (I' {*} Q)$. From (a) we know that there exists $\bar{\sigma}_1$ and $\bar{\sigma}_2$ such that $\bar{\sigma} = \bar{\sigma}_1 \uplus \bar{\sigma}_2$, (b) $\bar{\sigma}_1 \in I'$, and (c) $\bar{\sigma}_2 \in P$. From (c), and $I' {*} I \models \{P\}\, c\, \{Q\}$, by Def. 7.20, we know that (d) $I' {*} I \models \langle c, \bar{\sigma}_2\rangle \rhd_n Q$. We also know that (e) c is 0-atomic. From (d), (e), and (b), using Lemma 7.18 (item 1), we conclude □

Lemma 7.34.

$$\frac{I \models \{P_1\}\, c\, \{Q_1\} \quad I \models \{P_2\}\, c\, \{Q_2\} \quad I \text{ is precise}}{I \models \{P_1 \cap P_2\}\, c\, \{Q_1 \cap Q_2\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in P_1 \cap P_2$, and we need to show that $I \models \langle c, \bar{\sigma}\rangle \rhd_n (Q_1 \cap Q_2)$. From (a) we know that (b) $\bar{\sigma} \in P_1$ and (c) $\bar{\sigma} \in P_2$. From (b), and $I \models \{P_1\}\, c\, \{Q_1\}$, by Def. 7.20, we know that (d) $I \models \langle c, \bar{\sigma}\rangle \rhd_n Q_1$. From (c), and $I \models \{P_2\}\, c\, \{Q_2\}$, by Def. 7.20, we know that (e) $I \models \langle c, \bar{\sigma}\rangle \rhd_n Q_2$. From (d), (e), and knowing that I is precise, using Lemma 7.19, we conclude □

Lemma 7.35.

$$\frac{I \models \{P_1\}\, c\, \{Q_1\} \quad I \models \{P_2\}\, c\, \{Q_2\}}{I \models \{P_1 \cup P_2\}\, c\, \{Q_1 \cup Q_2\}}$$

Proof. From Def. 7.20, we assume there exist $\bar{\sigma}$ and n, such that (a) $\bar{\sigma} \in P_1 \cup P_2$, and we need to show that $I \models \langle c, \bar{\sigma}\rangle \rhd_n (Q_1 \cup Q_2)$. From (a) we know that either (b) $\bar{\sigma} \in P_1$ or (c) $\bar{\sigma} \in P_2$. If we assume (b), then from $I \models \{P_1\}\, c\, \{Q_1\}$, by Def. 7.20, we know that (d) $I \models \langle c, \bar{\sigma}\rangle \rhd_n Q_1$. We also know that (e) $Q_1 \subseteq Q_1 \cup Q_2$. From (d) and (e), using Lemma 7.10, we conclude. Similarly, if we assume (c), then from $I \models \{P_2\}\, c\, \{Q_2\}$, by Def. 7.20, we know that (f) $I \models \langle c, \bar{\sigma}\rangle \rhd_n Q_2$. We also know that (g) $Q_2 \subseteq Q_1 \cup Q_2$. From (f) and (g), using Lemma 7.10, we conclude □

Soundness theorem. The proof structure of Theorem 7.36 is similar for all CSL rules. It uses all lemmas from Lemma 7.22 to Lemma 7.35, one for each corresponding CSL rule. The proof structure is modular, if an extra rule is added to Fig. 7.4, we just need to prove an extra lemma for its correspondent semantic rule.

Theorem 7.36. If $I \vdash \{P\}\, c\, \{Q\}$, then $I \models \{P\}\, c\, \{Q\}$

Proof. By strong induction over the derivation tree depth of (a) $I \vdash \{P\}\, c\, \{Q\}$. After inversion of (a), we have one case for each rule:

- ASSIGNMENT: we know $I = \mathsf{Emp}$, $P = Q \circ [\![\nu := e]\!]$, and $c = (\nu := e)$, using Lemma 7.22, we conclude

- ACTION: we know $I = \mathsf{Emp}$, $P = Q \circ [\![a]\!]$, and $c = \langle a \rangle$, using Lemma 7.23, we conclude.

- SEQUENTIAL: we know $c = (c_1; c_2)$ and that there exists P' such that (a) $I \vdash \{P\}\, c_1\, \{P'\}$ and (b) $I \vdash \{P'\}\, c_2\, \{Q\}$. From (a), using the induction hypothesis, we obtain (c) $I \models \{P\}\, c_1\, \{P'\}$. From (b), using the induction hypothesis, we obtain (d) $I \models \{P'\}\, c_2\, \{Q\}$. From (c), and (d), using Lemma 7.24, we conclude

- SKIP: we know $I = \mathsf{Emp}$, $P = Q$, and $c = \textbf{skip}$, using Lemma 7.25, we conclude.

- CONDITIONAL: we know $c = (\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2)$ and that (a) $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, (b) $I \vdash \{P \cap \lfloor b \rfloor\}\, c_1\, \{Q\}$, and (c) $I \vdash \{P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q\}$. From (b), using the induction hypothesis, we obtain (d) $I \models \{P \cap \lfloor b \rfloor\}\, c_1\, \{Q\}$. From (c), using the induction hypothesis, we obtain (e) $I \models \{P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q\}$. From (a), (d), and (e), using Lemma 7.26, we conclude

- LOOP: we know $c = (\textbf{while } b \textbf{ do } c)$, $Q = (P \cap \lfloor \neg b \rfloor)$, (a) $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, and (b) $I \vdash \{P \cap \lfloor b \rfloor\}\, c\, \{P\}$. From (b), using the induction hypothesis, we obtain (c) $I \models \{P \cap \lfloor b \rfloor\}\, c\, \{P\}$. From (a), and (c), using Lemma 7.27, we conclude

- PARALLEL: we know $P = (P_1 * P_2)$, $c = (c_1 \parallel c_2)$, $Q = (Q_1 * Q_2)$, and that (a) $I \vdash \{P_1\}\, c_1\, \{Q_1\}$ and (b) $I \vdash \{P_2\}\, c_2\, \{Q_2\}$. From (a), using the induction hypothesis, we obtain (c) $I \models \{P_1\}\, c_1\, \{Q_1\}$. From (b), using the induction hypothesis, we obtain (d) $I \models \{P_2\}\, c_2\, \{Q_2\}$. From (c), and (d), using Lemma 7.28, we conclude

- ATOMIC: we know $c = (\textbf{atomic } c)$, and (a) $\mathsf{Emp} \vdash \{I * P\}\, c\, \{I * Q\}$. From (a), using the induction hypothesis, we obtain (b) $\mathsf{Emp} \models \{I * P\}\, c\, \{I * Q\}$. From (b), using Lemma 7.29, we conclude

- CONSEQUENCE: we know that there exists P' and Q', such that (a) $P \subseteq P'$, (b) $Q' \subseteq Q$, and (c) $I \vdash \{P'\}\, c\, \{Q'\}$. From (c), using the induction hypothesis, we obtain (d) $I \models \{P'\}\, c\, \{Q'\}$. From (a), (b), and (d), using Lemma 7.30, we conclude

- EXISTENTIAL: we know that $P = (\exists x.\ \mathcal{P}(x))$, $Q = (\exists x.\ \mathcal{Q}(x))$, and (a) $\forall x.\ I \vdash \{\mathcal{P}(x)\}\, c\, \{\mathcal{Q}(x)\}$. From (a), using the induction hypothesis, we obtain (b) $\forall x.\ I \models \{\mathcal{P}(x)\}\, c\, \{\mathcal{Q}(x)\}$. From (b), using Lemma 7.31, we conclude

- FRAME: we know $I = (I'' * I')$, and (a) $I' \vdash \{P\}\, c\, \{Q\}$. From (a), using the induction hypothesis, we obtain (b) $I' \models \{P\}\, c\, \{Q\}$. From (b), using Lemma 7.32, we conclude

- RESOURCE: we know $P = (I' * P')$, $Q = (I' * Q')$, and (a) $I' * I \vdash \{P'\}\, c\, \{Q'\}$. From (a), using the induction hypothesis, we obtain (b) $I' * I \models \{P'\}\, c\, \{Q'\}$. From (b), using Lemma 7.33, we conclude

- CONJUNCTION: we know $P = (P_1 \cap P_2)$, $Q = (Q_1 \cap Q_2)$, and that (a) $I \vdash \{P_1\}\, c\, \{Q_1\}$, (b) $I \vdash \{P_2\}\, c\, \{Q_2\}$, and (c) I is precise. From (a), using the induction hypothesis, we obtain (d) $I \models \{P_1\}\, c\, \{Q_1\}$. From (b), using the induction hypothesis, we obtain (e) $I \models \{P_2\}\, c\, \{Q_2\}$. From (d), (e), and (c), using Lemma 7.34, we conclude

- DISJUNCTION: we know $P = (P_1 \cup P_2)$, $Q = (Q_1 \cup Q_2)$, and that (a) $I \vdash \{P_1\}\, c\, \{Q_1\}$ and (b) $I \vdash \{P_2\}\, c\, \{Q_2\}$. From (a), using the induction hypothesis, we obtain (c) $I \models \{P_1\}\, c\, \{Q_1\}$. From (b), using the induction hypothesis, we obtain (d) $I \models \{P_2\}\, c\, \{Q_2\}$. From (c), and (d), using Lemma 7.35, we conclude $\square$

7.3.2 With Regard to the Parameterized Semantics

In this section, we proof the soundness of CSL with partial permissions with regard to the parameterized semantics of Sec. 3.8. First, we need to define the semantic meaning of a Λ-parameterized CSL quadruple.

Definition 7.37. $I \models_{[\Lambda]} \{P\}\, c\, \{Q\}$, if and only if, for all $\bar{\sigma}$, such that $\bar{\sigma} \in I * P$, we have:

1. $\neg[\Lambda]\ \langle c, \bar{\sigma}|_{\Pi}\rangle \longmapsto^* \mathsf{abort}$
2. If $[\Lambda]\ \langle c, \bar{\sigma}|_{\Pi}\rangle \longmapsto^* \langle \mathbf{skip}, \bar{\sigma}'|_{\Pi}\rangle$, then $\bar{\sigma}' \in I * Q$

This definition is straightforward. The $I \models_{[\Lambda]} \{P\}\, c\, \{Q\}$ quadruple ensures that for any state satisfying the pre-condition $I * P$ will not abort, and, if it the execution completes, the final state will satisfy $I * Q$. Given this definition, we can phrase and prove the soundness theorem below:

Theorem 7.38. If $I \vdash \{P\}\, c\, \{Q\}$, and Λ provides the DRF-guarantee, then $I \models_{[\Lambda]} \{P\}\, c\, \{Q\}$

Proof. From $I \vdash \{P\}\, c\, \{Q\}$, using Theorem 7.36, we obtain (a) $I \models \{P\}\, c\, \{Q\}$. From Def. 7.37, we can prove the goal if, by assuming there is a state $\bar{\sigma}$, such that (b) $\bar{\sigma} \in I * P$, we can establish the following two conditions:

(c) $\neg[\Lambda]\ \langle c, \bar{\sigma}|_{\Pi}\rangle \longmapsto^* \textsf{abort}$

(d) If $[\Lambda]\ \langle c, \bar{\sigma}|_{\Pi}\rangle \longmapsto^* \langle \mathbf{skip}, \bar{\sigma}'|_{\Pi}\rangle$, then $\bar{\sigma}' \in I * Q$

From (a), and (b), using Lemma 7.21, we know that:

(e) $\neg\langle c, \bar{\sigma}|_{\Pi}\rangle \longmapsto^* \textsf{abort}$

(f) $\neg\langle c, \bar{\sigma}|_{\Pi}\rangle \longmapsto^* \textsf{race}$

(g) If $\langle c, \bar{\sigma}|_{\Pi}\rangle \longmapsto^* \langle \mathbf{skip}, \bar{\sigma}'|_{\Pi}\rangle$, then $\bar{\sigma}' \in I * Q$

Since Λ provides the DRF-guarantee, from (e), and (f), based on Def. 5.2, we establish (c) and we know

(h) If $[\Lambda]\ \langle c, \bar{\sigma}|_{\Pi}\rangle \longmapsto^* \langle \mathbf{skip}, \bar{\sigma}'|_{\Pi}\rangle$, then $\langle c, \bar{\sigma}|_{\Pi}\rangle \longmapsto^* \langle \mathbf{skip}, \bar{\sigma}'|_{\Pi}\rangle$

From (h), and (g), we can establish (d) $\square$

7.3.3 With Regard to the Relaxed Semantics

The proof of soundness regarding the $\rightsquigarrow$-parameterized relaxed semantics of Sec. 3.10 is straightforward.

Theorem 7.39. If $I \vdash \{P\}\, c\, \{Q\}$, then $I \models_{[\rightsquigarrow]} \{P\}\, c\, \{Q\}$

Proof. From Theorem 5.12, we know that (a) $\rightsquigarrow$ provides the DRF-guarantee. From $I \vdash \{P\}\, c\, \{Q\}$, and (a), using Theorem 7.38, we prove our goal $I \models_{[\rightsquigarrow]} \{P\}\, c\, \{Q\}$ $\square$

7.4 Verification Examples

In this section, we present some examples of code verified using CSL with permissions.

7.4.1 Dijkstra Semaphore

The first example is the well know Dijkstra semaphore, presented in Fig. 7.5. The semaphore is made of a single memory cell that has two operations P and V. It allows at most n number of threads to execute the code block between a pair of P and V, where n is the initial value of the semaphore.

```
P(sem)                          V(sem)
  atomic                          atomic [sem] := [sem] + 1
    wait [sem] > 0;
    [sem] := [sem] − 1
```

Figure 7.5: Dijkstra semaphore

Here we will verify the semaphore implementation as working as a way to access a shared read-only cell of memory. In the specification the semaphore is at location p, and the data it protects is in location d. The initial value for the semaphore is assumed to be n.

```
∀p, n, d. P(sem)
(∃v. (0 ≤ v ≤ n) ∩ (p ↦ v ∗ d ↦^{v/n} _)) ⊢
  {sem ↦ p}
  atomic
    {(∃v. (0 ≤ v ≤ n) ∩ (p ↦ v ∗ d ↦^{v/n} _)) ∗ sem ↦ p}
    wait [sem] > 0;
    {(∃v. (0 < v ≤ n) ∩ (p ↦ v ∗ d ↦^{v/n} _)) ∗ sem ↦ p}
    [sem] := [sem] − 1
    {(∃v. (0 < v ≤ n) ∩ (p ↦ (v−1) ∗ d ↦^{v/n} _)) ∗ sem ↦ p}
    {(∃v. (0 ≤ v ≤ n) ∩ (p ↦ v ∗ d ↦^{v/n} _)) ∗ d ↦^{1/n} _ ∗ sem ↦ p}
  {d ↦^{1/n} _ ∗ sem ↦ p}
```

Just like the mutex unlock operations (from Sec. 6.5.2), the semaphore V operation does not make any check. From the precondition, we know that the value v of the semaphore is less than n (we already have $1/n$ part of the data), because otherwise there is no initial state that satisfies the separating conjunction of the invariant and the precondition. That would mean, in practice, that the precondition is false and we would never be able to use V.

```
∀p, n, d. V(sem)
(∃v. (0≤v≤n)∩(p↦v∗d ↦^{v/n} _)) ⊢
  {d ↦^{1/n} _∗sem↦p}
  atomic
    {(∃v. (0≤v≤n)∩(p↦v∗d ↦^{v/n} _))∗d ↦^{1/n} _∗sem↦p}
    {(∃v. (0≤v<n)∩(p↦v∗d ↦^{(v+1)/n} _))∗sem↦p}
    [sem]:=[sem]+1
    {(∃v. (0≤v<n)∩(p↦(v+1)∗d ↦^{(v+1)/n} _))∗sem↦p}
    {(∃v. (0≤v≤n)∩(p↦v∗d ↦^{v/n} _))∗sem↦p}
  {sem↦p}
```

7.4.2 Dekker's Algorithm

Dekker's algorithm is a primitive synchronization algorithm that implements mutual exclusion between two threads. The code for Dekker's algorithm is shown in Fig. 7.6.

```
atomic [a1]:=1;               atomic [a2]:=1;
atomic v1:=[a2];              atomic v2:=[a1];
if v1=0 then             ||   if v2=0 then
  /* critical section */;       /* critical section */;
atomic [a1]:=0                atomic [a2]:=0
```

Figure 7.6: Dekker's algorithm

In order to verify Dekker's algorithm, we use two auxiliary variables, cs1 and cs2, which are used to mark the boundaries of the critical section. cs1 is 1 when the left hand side thread is inside the critical section, and 0 otherwise; cs2 works analogously.

The invariant for Dekker's algorithm is defined below:

$$\begin{aligned}
\mathsf{dekkers}(I) \stackrel{\text{def}}{=}\ & \exists v_1, v_2, c_1, c_2.\ \texttt{cs1} \overset{.5}{\mapsto} c_1 * \texttt{cs2} \overset{.5}{\mapsto} c_2 \\
& * ((c_1 \neq 0 \cup c_2 \neq 0) \cap \mathsf{Emp} \cup (c_1 = 0 \cap c_2 = 0) \cap I) \\
& * (\exists p.\ \texttt{a1} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} v_1) * (\exists p.\ \texttt{a2} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} v_2) \\
& \cap (v_1 = 0 \cap c_1 = 0 \cup v_1 = 1) \cap (v_2 = 0 \cap c_2 = 0 \cup v_2 = 1)
\end{aligned}$$

The invariant ensures that the memory locations pointed by `a1` and `a2` are either 0 or 1. Also, it ensures that `cs1` is always 0 when the memory location pointed by `a1` is 0; similarly, it ensures that `cs2` is always 0 when the memory location pointed by `a2` is 0. The ownership of shared memory described by I depends on the values `cs1` and `cs2`. If they are both 0, it means that none of them is in the critical section, therefore I is part of the invariant. If any of them is 1, it means that one of them is in the critical section and own the shared memory protected by I. We use partial permissions to protect the variables `cs1`, `a1` and the memory location pointed by `a1` from being written by the right hand side thread; and, conversely, to protect the variables `cs2`, `a2` and the memory location pointed by `a2` from being written by the left hand side thread.

Now we are ready to lay down the proof for Dekker's algorithm, which is shown below

$$
\begin{array}{l}
\forall I.\ \mathsf{dekkers}(I) \vdash \\
\quad \left\{(\exists p.\ \mathtt{a1} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 0) * \mathtt{v1} \mapsto \_ * \mathtt{cs1} \overset{.5}{\mapsto} 0 * (\exists p.\ \mathtt{a2} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 0) * \mathtt{v2} \mapsto \_ * \mathtt{cs2} \overset{.5}{\mapsto} 0\right\} \\
\begin{array}{lcl}
\left\{\exists p.\ \mathtt{a1} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 0 * \mathtt{v1} \mapsto \_ * \mathtt{cs1} \overset{.5}{\mapsto} 0\right\} & & \left\{\exists p.\ \mathtt{a2} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 0 * \mathtt{v2} \mapsto \_ * \mathtt{cs2} \overset{.5}{\mapsto} 0\right\} \\
\textbf{atomic}\ [\mathtt{a1}] := 1; & & \textbf{atomic}\ [\mathtt{a2}] := 1; \\
\left\{\exists p.\ \mathtt{a1} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 1 * \mathtt{v1} \mapsto \_ * \mathtt{cs1} \overset{.5}{\mapsto} 0\right\} & & \left\{\exists p.\ \mathtt{a2} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 1 * \mathtt{v2} \mapsto \_ * \mathtt{cs2} \overset{.5}{\mapsto} 0\right\} \\
\textbf{atomic}\ (\mathtt{v1} := [\mathtt{a2}]; \mathtt{cs1} := -(\mathtt{v1} - 1)); & & \textbf{atomic}\ (\mathtt{v2} := [\mathtt{a1}]; \mathtt{cs2} := -(\mathtt{v2} - 1)); \\
\left\{\begin{array}{l} \exists p.\ \mathtt{a1} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 1 * (\mathtt{v1} \mapsto 1 * \mathtt{cs1} \overset{.5}{\mapsto} 0) \\ \quad \cup (\mathtt{v1} \mapsto 0 * \mathtt{cs1} \overset{.5}{\mapsto} 1 * I) \end{array}\right\} & & \left\{\begin{array}{l} \exists p.\ \mathtt{a2} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 1 * (\mathtt{v2} \mapsto 1 * \mathtt{cs2} \overset{.5}{\mapsto} 0) \\ \quad \cup (\mathtt{v2} \mapsto 0 * \mathtt{cs2} \overset{.5}{\mapsto} 1 * I) \end{array}\right\} \\
\textbf{if}\ \mathtt{v1} = 0\ \textbf{then} & & \textbf{if}\ \mathtt{v2} = 0\ \textbf{then} \\
\quad \left\{\exists p.\ \mathtt{a1} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 1 * \mathtt{v1} \mapsto \_ * \mathtt{cs1} \overset{.5}{\mapsto} 1 * I\right\} & \parallel & \quad \left\{\exists p.\ \mathtt{a2} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 1 * \mathtt{v2} \mapsto \_ * \mathtt{cs2} \overset{.5}{\mapsto} 1 * I\right\} \\
\quad \textit{/* critical section */}; & & \quad \textit{/* critical section */}; \\
\quad \left\{\exists p.\ \mathtt{a1} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 1 * \mathtt{v1} \mapsto \_ * \mathtt{cs1} \overset{.5}{\mapsto} 1 * I\right\} & & \quad \left\{\exists p.\ \mathtt{a2} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 1 * \mathtt{v2} \mapsto \_ * \mathtt{cs2} \overset{.5}{\mapsto} 1 * I\right\} \\
\left\{\begin{array}{l} \exists p.\ \mathtt{a1} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 1 * \mathtt{v1} \mapsto \_ \\ \quad * \mathtt{cs1} \overset{.5}{\mapsto} 0 \cup (\mathtt{cs1} \overset{.5}{\mapsto} 1 * I) \end{array}\right\} & & \left\{\begin{array}{l} \exists p.\ \mathtt{a2} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 1 * \mathtt{v2} \mapsto \_ \\ \quad * \mathtt{cs2} \overset{.5}{\mapsto} 0 \cup (\mathtt{cs2} \overset{.5}{\mapsto} 1 * I) \end{array}\right\} \\
\textbf{atomic}\ ([\mathtt{a1}] := 0; \mathtt{cs1} := 0) & & \textbf{atomic}\ ([\mathtt{a2}] := 0; \mathtt{cs2} := 0) \\
\left\{\exists p.\ \mathtt{a1} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 0 * \mathtt{v1} \mapsto \_ * \mathtt{cs1} \overset{.5}{\mapsto} 0\right\} & & \left\{\exists p.\ \mathtt{a2} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 0 * \mathtt{v2} \mapsto \_ * \mathtt{cs2} \overset{.5}{\mapsto} 0\right\}
\end{array} \\
\quad \left\{(\exists p.\ \mathtt{a1} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 0) * \mathtt{v1} \mapsto \_ * \mathtt{cs1} \overset{.5}{\mapsto} 0 * (\exists p.\ \mathtt{a2} \overset{.5}{\mapsto} p * p \overset{.5}{\mapsto} 0) * \mathtt{v2} \mapsto \_ * \mathtt{cs2} \overset{.5}{\mapsto} 0\right\}
\end{array}
$$

7.4.3 Two-Lock Queue

The code displayed in Fig. 7.7 implements the two-lock concurrent queue algorithm presented by Michael and Scott [55]. The data structures are simple. The queue is a quadruple composed, in displacement order, by a mutex protecting the head pointer, a mutex protecting the tail pointer, the head pointer and the tail pointer. A node is a pair composed by value and next pointer fields.

The algorithm keeps a queue with two pointers, one to the head and the other to the tail. To simplify operations, an extra node is always present in the queue. Whenever the queue is empty, the next field of that node is nil. Otherwise, if the queue is non-empty, it points to the node holding the first value, and the next field of the last node is nil.

Concurrent insertions synchronize through a mutex. In the same way, concurrent deletions synchronize through another mutex. A concurrent insertion can happen without synchronization. However, enqueue updates the next field of the last node, and dequeue checks the next field of the first node to tell whether the queue is empty of not. When the queue is empty, both ends point to a single node, and both operations will refer to the same next field. To avoid races, we need to wrap those two operations inside an atomic block, as can be seen in Fig. 7.7.

```
INIT(queue)
local node
  ⟨node := cons(0, 0)⟩;
  ⟨queue := cons(0, 0, node, node)⟩

ENQUEUE(queue, value)
local node, tail, mutex
  ⟨node := cons(value, 0)⟩;
  mutex := queue+1;
  LOCK(mutex);
  tail := [queue+3];
  atomic [tail+1] := node;
  [queue+3] := node;
  UNLOCK(mutex)
```

```
DEQUEUE(queue, value, result)
local node, head, mutex
  mutex := queue;
  LOCK(mutex);
  head := [queue+2];
  atomic node := [head+1];
  if node = 0 then
    UNLOCK(mutex);
    result := 0
  else
    value := [node];
    [queue+2] := node;
    UNLOCK(mutex);
    ⟨dispose(head, 2)⟩;
    result := 1
```

Figure 7.7: Two-lock concurrent queue

In order to verify the two-lock queue, we add two extra fields to the queue record, which are auxiliary for the proof and are not read by the code. Those fields are supposed to contain the most current head and and tail pointers even when the queue is being in the middle of an update and in an inconsistent state.

In order to describe a two-lock queue, we use the following definitions (we will reuse the mutex description and proofs from Sec. 6.5.2).

$$\begin{aligned}\mathsf{twolock}(q) \;&\stackrel{\text{def}}{=}\; \exists h, t, n.\ (q{+}4) \overset{.5}{\mapsto} (h, t) * \mathsf{list}_n(h{+}1, t{+}1) * (t{+}1) \mapsto 0\\ &\qquad * \mathsf{mux}(q, (q{+}2) \mapsto h * (q{+}4) \overset{.5}{\mapsto} h * h \mapsto \_)\\ &\qquad * \mathsf{mux}(q{+}1, (q{+}3) \mapsto t * (q{+}5) \overset{.5}{\mapsto} t)\end{aligned}$$

$$\mathsf{list}_n(h, t) \;\stackrel{\text{def}}{=}\; \begin{cases} h = t \cap \mathsf{Emp} & \text{when } n = 0\\ \exists p.\ p \neq 0 \cap h \mapsto p * p \mapsto \_ * \mathsf{list}_{n-1}(p{+}1, t) & \text{when } n > 0\end{cases}$$

The $\mathsf{twolock}(q)$ is defined in terms of two mutexes. When the mutexes are not owned, the queue has six fields: head mutex, tail mutex, head, tail, head mirror, and tail mirror. Both mutexes will have value 0, both the head and head mirror point to the first node, and both tail and tail mirror point to the last node. There are $n + 1$ nodes which we define as the head cell of the first node, a n-length ragged list of pointer and values, and finally the next pointer field of the last node.

When the tail pointer mutex is locked (by enqueue) the tail pointer becomes private memory. When the head pointer mutex is locked (by dequeue) both the head pointer and the value field of the cell it points to become private memory. In both cases, the tail/head pointer mirror also become available as shared read only; but that is used only by the proof to maintain coherence between the shared memory invariant and the private assertions. We use partial permissions to achieve that.

We are now ready to present the proof, starting with the init routine which creates a two lock queue in the private memory.

```
$\mathsf{INIT}(\texttt{queue})$
$\textbf{local}\ \texttt{node}$
$\mathsf{Emp} \vdash$
    $\{\texttt{queue} \mapsto \_ * \texttt{node} \mapsto \_\}$
    $\langle \texttt{node} := \textbf{cons}(0, 0) \rangle;$
    $\{\texttt{queue} \mapsto \_ * (\exists p.\ \texttt{node} \mapsto p * p \mapsto (0, 0))\}$
    $\texttt{queue} := \textbf{cons}(0, 0, \texttt{node}, \texttt{node}, \texttt{node}, \texttt{node})$
    $\{\exists q, p.\ \texttt{queue} \mapsto q * \texttt{node} \mapsto p * p \mapsto (0, 0) * q \mapsto (0, 0, p, p, p, p)\}$
    $\{\exists q.\ \texttt{queue} \mapsto q * \texttt{node} \mapsto \_ * \mathsf{twolock}(q)\}$
```

Next we present the verification of the enqueue operation. The interesting thing about the proof is that lock and unlock just get exclusive access to the list tail. It is actually the atomic write that performs the enqueueing of the new node. Just after that operation the tail pointer mirror points to the new node while tail pointer still points to the previous node, and is updated before the unlock. The question is what if, concurrently, there is a dequeue at that point? As we know there is always one node in the queue, and no one else is enqueueing (we have the lock) so the tail pointer mirror is always valid.

```
$\forall q, v.$ ENQUEUE(queue, val)
**local** node, tail, mux
twolock$(q) \vdash$
    $\{\texttt{queue} \mapsto q * \texttt{val} \mapsto v * \texttt{node} \mapsto \_ * \texttt{tail} \mapsto \_ * \texttt{mux} \mapsto \_\}$
    $\langle \texttt{node} := \mathbf{cons}(\texttt{val}, 0) \rangle;$
    $\{\exists p.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto v * \texttt{node} \mapsto p * \texttt{tail} \mapsto \_ * \texttt{mux} \mapsto \_ * p \mapsto (v, 0)\}$
    $\texttt{mux} := \texttt{queue} + 1;$
    $\{\exists p.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto v * \texttt{node} \mapsto p * \texttt{tail} \mapsto \_ * \texttt{mux} \mapsto (q+1) * p \mapsto (v, 0)\}$
    $[q+1, \exists t.\ (q+3) \mapsto t * (q+5) \overset{.5}{\mapsto} t]$ LOCK(mux);
    $\left\{\begin{array}{l} \exists p.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto v * \texttt{node} \mapsto p * \texttt{tail} \mapsto \_ * \texttt{mux} \mapsto (q+1) * p \mapsto (v, 0) \\ \quad * (\exists t.\ (q+3) \mapsto t * (q+5) \overset{.5}{\mapsto} t) \end{array}\right\}$
    $\texttt{tail} := [\texttt{queue} + 3];$
    $\left\{\begin{array}{l} \exists p, t.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto v * \texttt{node} \mapsto p * \texttt{tail} \mapsto t * \texttt{mux} \mapsto (q+1) * p \mapsto (v, 0) \\ \quad * (q+3) \mapsto t * (q+5) \overset{.5}{\mapsto} t \end{array}\right\}$
    **atomic** $([\texttt{tail} + 1] := \texttt{node}; [\texttt{queue} + 5] := \texttt{node});$
    $\left\{\begin{array}{l} \exists p, t.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto v * \texttt{node} \mapsto p * \texttt{tail} \mapsto t * \texttt{mux} \mapsto (q+1) \\ \quad * (q+3) \mapsto t * (q+5) \overset{.5}{\mapsto} p \end{array}\right\}$
    $[\texttt{queue} + 3] := \texttt{node};$
    $\left\{\begin{array}{l} \exists p.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto v * \texttt{node} \mapsto p * \texttt{tail} \mapsto \_ * \texttt{mux} \mapsto (q+1) \\ \quad * (q+3) \mapsto p * (q+5) \overset{.5}{\mapsto} p \end{array}\right\}$
    $[q+1, \exists t.\ (q+3) \mapsto t * (q+5) \overset{.5}{\mapsto} t]$ UNLOCK(mux)
    $\{\texttt{queue} \mapsto q * \texttt{val} \mapsto v * \texttt{node} \mapsto \_ * \texttt{tail} \mapsto \_ * \texttt{mux} \mapsto \_\}$
```

The dequeue operation follows the same ideas as enqueue. Similarly, the lock and unlock do not actually perform the dequeue operation. The node is acquired in the atomic block.

```
$\forall q.$ DEQUEUE(queue, val, res)
**local** node, head, mux
twolock$(q) \vdash$
  $\{\texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto \_ * \texttt{node} \mapsto \_ * \texttt{head} \mapsto \_ * \texttt{mux} \mapsto \_\}$
  mux := queue;
  $\{\texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto \_ * \texttt{node} \mapsto \_ * \texttt{head} \mapsto \_ * \texttt{mux} \mapsto q\}$
  $[q, \exists h.\ (q+2) \mapsto h * (q+4) \overset{.5}{\mapsto} h * h \mapsto \_]$ LOCK(mux);
  $\{\texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto \_ * \texttt{node} \mapsto \_ * \texttt{head} \mapsto \_ * \texttt{mux} \mapsto q$
      $* (\exists h.\ (q+2) \mapsto h * (q+4) \overset{.5}{\mapsto} h * h \mapsto \_)\}$
  head := [queue + 2];
  $\{\exists h.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto \_ * \texttt{node} \mapsto \_ * \texttt{head} \mapsto h * \texttt{mux} \mapsto q$
      $* (q+2) \mapsto h * (q+4) \overset{.5}{\mapsto} h * h \mapsto \_\}$
  **atomic** (node := [head + 1]; **if** node $\neq 0$ **then** [queue + 4] := node);
  $\{\exists h.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto \_ * \texttt{head} \mapsto h * \texttt{mux} \mapsto q * (q+2) \mapsto h * h \mapsto \_$
      $* ((\texttt{node} \mapsto 0 * (q+4) \overset{.5}{\mapsto} h) \cup (\exists n \neq 0.\ \texttt{node} \mapsto n * (q+4) \overset{.5}{\mapsto} n * (h+1) \mapsto n * n \mapsto \_))\}$
  **if** node = 0 **then**
    $\{\exists h.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto \_ * \texttt{head} \mapsto h * \texttt{mux} \mapsto q * (q+2) \mapsto h * h \mapsto \_$
        $* \texttt{node} \mapsto 0 * (q+4) \overset{.5}{\mapsto} h\}$
    $[q, \exists h.\ (q+2) \mapsto h * (q+4) \overset{.5}{\mapsto} h * h \mapsto \_]$ UNLOCK(mux);
    $\{\texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto \_ * \texttt{node} \mapsto \_ * \texttt{head} \mapsto \_ * \texttt{mux} \mapsto \_\}$
    res := 0
    $\{\texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto 0 * \texttt{node} \mapsto \_ * \texttt{head} \mapsto \_ * \texttt{mux} \mapsto \_\}$
  **else**
    $\{\exists h, n.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto \_ * \texttt{head} \mapsto h * \texttt{mux} \mapsto q * (q+2) \mapsto h * h \mapsto \_$
        $* \texttt{node} \mapsto n * (q+4) \overset{.5}{\mapsto} n * (h+1) \mapsto n * n \mapsto \_\}$
    val := [node];
    $\{\exists h, n, v.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto v * \texttt{res} \mapsto \_ * \texttt{head} \mapsto h * \texttt{mux} \mapsto q * (q+2) \mapsto h * h \mapsto \_$
        $* \texttt{node} \mapsto n * (q+4) \overset{.5}{\mapsto} n * (h+1) \mapsto n * n \mapsto v\}$
    [queue + 2] := node;
    $\{\exists h, n, v.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto v * \texttt{res} \mapsto \_ * \texttt{head} \mapsto h * \texttt{mux} \mapsto q * (q+2) \mapsto n * h \mapsto \_$
        $* \texttt{node} \mapsto n * (q+4) \overset{.5}{\mapsto} n * (h+1) \mapsto n * n \mapsto v\}$
    $[q, \exists h.\ (q+2) \mapsto h * (q+4) \overset{.5}{\mapsto} h * h \mapsto \_]$ UNLOCK(mux);
    $\{\exists h, n.\ \texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto \_ * \texttt{head} \mapsto h * \texttt{mux} \mapsto q * h \mapsto \_$
        $* \texttt{node} \mapsto n * (h+1) \mapsto n\}$
    $\langle$**dispose**(head, 2)$\rangle$;
    $\{\texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto \_ * \texttt{head} \mapsto \_ * \texttt{mux} \mapsto \_ * \texttt{node} \mapsto \_\}$
    res := 1
    $\{\texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto 1 * \texttt{head} \mapsto \_ * \texttt{mux} \mapsto \_ * \texttt{node} \mapsto \_\}$
  $\{\texttt{queue} \mapsto q * \texttt{val} \mapsto \_ * \texttt{res} \mapsto \_ * \texttt{head} \mapsto \_ * \texttt{mux} \mapsto \_ * \texttt{node} \mapsto \_\}$
```

Chapter 8

Separated Assume-Guarantee Logic

In this chapter, we present the Separated Assume-Guarantee Logic (SAGL) [30, 71], a logic for concurrent programming that combines the idea of memory separation from CSL with assume-guarantee reasoning [48].

8.1 Assertion Language

We present SAGL using the same assertion language from Sec. 6.1, but we define some extensions. As presented earlier (in Fig. 6.1), assertions are sets of states in the meta-logic. We extend the notion of assertion to a pair of states. The definition of a *binary assertion* is shown in Fig. 8.1. As for assertions, set operators also apply to binary assertions (e.g. $\cap$ for conjunction,$\cup$ for disjunction, $\subseteq$ for implication, etc). Since we use binary assertions as a way to represent state transitions, we also call them *actions*.

$$(\mathit{BinaryAssertion}) \quad A, G \subseteq \mathit{State} \times \mathit{State}$$

Figure 8.1: Binary assertions

In Fig. 8.2, we define common operations and constants related to binary assertions. To create a binary assertion from a pair of assertions we take the cartesian product $P \times Q$. We use the notation P^2, to describe the cartesian product of P and itself. We use the notation $P \circ Q$ to describe the set of states for which the image from G is in P. We also define a separating conjunction for binary assertions, and the identity binary assertion.

$$\begin{aligned}
P \times Q &\stackrel{\text{def}}{=} \{(\sigma, \sigma') \mid \sigma \in P \wedge \sigma' \in Q\} \\
P^2 &\stackrel{\text{def}}{=} P \times P \\
P \circ G &\stackrel{\text{def}}{=} \{\sigma \mid \forall \sigma'.\, (\sigma, \sigma') \in G \rightarrow \sigma' \in P\} \\
G_1 * G_2 &\stackrel{\text{def}}{=} \{(\sigma_1 \uplus \sigma_2, \sigma'_1 \uplus \sigma'_2) \mid (\sigma_1, \sigma'_1) \in G_1 \wedge (\sigma_2, \sigma'_2) \in G_2\} \\
\mathsf{Id} &\stackrel{\text{def}}{=} \{(\sigma, \sigma)\}
\end{aligned}$$

Figure 8.2: Auxiliary binary assertion definitions

8.2 Inference Rules

The inference rules for SAGL are shown in Figure 8.3. Informally, the judgment $A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\}$ says that the state can be split implicitly into a shared part and a private part. The private part can be accessed only by c. P and Q are pre- and post-conditions for the private state. The shared part can be accessed by both c, when within atomic blocks, and its environment. P_s and Q_s are pre- and post-conditions for the shared state. The binary assertions A and G are used to coordinate shared memory access using an *assume-guarantee* methodology.

Assume-Guarantee reasoning is a modular way of reasoning about a shared resource. In Assume-Guarantee, each participant has a requirement over the shared resource, which is not know globally. In order to maintain its requirement, the participant has to publish an assumption about the resource, for which the requirement is stable, i.e. does not change. On the other hand, a participant might want to modify the shared resource. It is possible to do that as long as the participant publishes a guarantee about the changes. Having all participants publishing their assumptions and guarantees, we can combine them as long as the guarantee of every participant is stronger than the assumptions of the others. A and G are c's assumption and guarantee, respectively.

The first six rules are sequential rules adorned with assumptions, guarantees, and assertions over the shared memory which do not play any important role in these rules. As for CSL, we have a special rule ACTION which must be used to verify allocation and

$$\frac{}{\mathsf{Emp}^2, \mathsf{Emp}^2 \vdash \{\mathsf{Emp}, Q \circ [\![\nu := e]\!]\}\, \nu := e\, \{\mathsf{Emp}, Q\}} \quad (\text{ASSIGNMENT})$$

$$\frac{}{\mathsf{Emp}^2, \mathsf{Emp}^2 \vdash \{\mathsf{Emp}, Q \circ [\![a]\!]\}\, \langle a \rangle\, \{\mathsf{Emp}, Q\}} \quad (\text{ACTION})$$

$$\frac{A, G \vdash \{P_s, P\}\, c_1\, \{P'_s, P'\} \quad A, G \vdash \{P'_s, P'\}\, c_2\, \{Q_s, Q\}}{A, G \vdash \{P_s, P\}\, c_1; c_2\, \{Q_s, Q\}} \quad (\text{SEQUENTIAL})$$

$$\frac{}{\mathsf{Emp}^2, \mathsf{Emp}^2 \vdash \{\mathsf{Emp}, P\}\, \textbf{skip}\, \{\mathsf{Emp}, P\}} \quad (\text{SKIP})$$

$$\frac{\begin{array}{c} A, G \vdash \{P_s, P \cap \lfloor b \rfloor\}\, c_1\, \{Q_s, Q\} \\ P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad A, G \vdash \{P_s, P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q_s, Q\} \end{array}}{A, G \vdash \{P_s, P\}\, \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\, \{Q_s, Q\}} \quad (\text{CONDITIONAL})$$

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad A, G \vdash \{P_s, P \cap \lfloor b \rfloor\}\, c\, \{P_s, P\}}{A, G \vdash \{P_s, P\}\, \textbf{while}\ b\ \textbf{do}\ c\, \{P_s, P \cap \lfloor \neg b \rfloor\}} \quad (\text{LOOP})$$

$$\frac{\begin{array}{c} A_1, G \cap A_2 \vdash \{P_{s1}, P_1\}\, c_1\, \{Q_{s1}, Q_1\} \quad P_{s1} \subseteq P_{s1} \circ A_1 \\ A_2, G \cap A_1 \vdash \{P_{s2}, P_2\}\, c_2\, \{Q_{s2}, Q_2\} \quad P_{s2} \subseteq P_{s2} \circ A_2 \end{array}}{A_1 \cap A_2, G \vdash \{P_{s1} \cap P_{s2}, P_1 * P_2\}\, c_1 \parallel c_2\, \{Q_{s1} \cap Q_{s2}, Q_1 * Q_2\}} \quad (\text{PARALLEL})$$

$$\frac{\mathsf{Emp}^2, \mathsf{Emp}^2 \vdash \{\mathsf{Emp}, P_s * P\}\, c\, \{\mathsf{Emp}, Q_s * Q\} \quad P_s \times Q_s \subseteq G \quad Q_s \subseteq Q_s \circ A}{A, G \vdash \{P_s, P\}\, \textbf{atomic}\ c\, \{Q_s, Q\}} \quad (\text{ATOMIC})$$

$$\frac{\begin{array}{c} A', G' \vdash \{P'_s, P'\}\, c\, \{Q'_s, Q'\} \\ A \subseteq A' \quad P_s \subseteq P'_s \quad P \subseteq P' \quad G' \subseteq G \quad Q'_s \subseteq Q_s \quad Q' \subseteq Q \end{array}}{A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\}} \quad (\text{CONSEQUENCE})$$

$$\frac{\forall x.\ A, G \vdash \{P_s, \mathcal{P}(x)\}\, c\, \{Q_s, \mathcal{Q}(x)\}}{A, G \vdash \{P_s, \exists x.\ \mathcal{P}(x)\}\, c\, \{Q_s, \exists x.\ \mathcal{Q}(x)\}} \quad (\text{EXISTENTIAL})$$

$$\frac{\begin{array}{c} A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\} \\ I \text{ is precise } \textbf{or } \mathsf{img}(G) \text{ and } \mathsf{dom}(A) \text{ coincide} \end{array}}{I^2 * A, I^2 * G \vdash \{I * P_s, P\}\, c\, \{I * Q_s, Q\}} \quad (\text{FRAME})$$

$$\frac{A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\}}{\mathsf{Emp}^2, \mathsf{Emp}^2 \vdash \{\mathsf{Emp}, P_s * P\}\, c\, \{\mathsf{Emp}, Q_s * Q\}} \quad (\text{RESOURCE})$$

$$\frac{\begin{array}{c} \mathsf{img}(G) \text{ is precise} \\ A, G \vdash \{P_s, P_1\}\, c\, \{Q_s, Q_1\} \quad A, G \vdash \{P_s, P_2\}\, c\, \{Q_s, Q_2\} \end{array}}{A, G \vdash \{P_s, P_1 \cap P_2\}\, c\, \{Q_s, Q_1 \cap Q_2\}} \quad (\text{CONJUNCTION})$$

$$\frac{A, G \vdash \{P_s, P_1\}\, c\, \{Q_s, Q_1\} \quad A, G \vdash \{P_s, P_2\}\, c\, \{Q_s, Q_2\}}{A, G \vdash \{P_s, P_1 \cup P_2\}\, c\, \{Q_s, Q_1 \cup Q_2\}} \quad (\text{DISJUNCTION})$$

Figure 8.3: SAGL inference rules

deallocation. Regular memory accesses can be verified using the ASSIGNMENT instead. They alter private memory solely. Sequential composition, conditional commands, and looping are just composition rules very similar to Hoare logic.

The PARALLEL rule set constraints over parallel composition. As we know, the state in SAGL is separated into shared and private parts. For parallel composition, we can split the private part in two, one for each thread. The shared state is of course shared, but in a coordinated way. The shared state must belong to a conjunction of the pre-conditions of the threads. Each precondition must be stable regarding the respective assumption ($P_{s1} \subseteq P_{s2} \circ A_1$ and $P_{s2} \subseteq P_{s2} \circ A_2$). Furthermore, each thread must be verified using a strengthened guarantee that is compatible with the other thread's assumption ($G \cap A_2$ for c_1 and $G \cap A_2$ for c_2).

In order to fully understand SAGL, we must also take a look ate the ATOMIC rule. As in CSL, we also require using an atomic block to modify shared memory. In the same fashion, during the access we get a hold on the shared memory portion (P_s) which must be reinstated afterwards (Q_s). During the transition not only the shared memory might change but also its description, from P_s to Q_s. But there are two constraints: first, after the shared memory access the changes must satisfy the thread's guarantee ($P_s \times Q_s \subseteq G$), and the new assertion over the shared memory must be stable regarding the thread's assumption ($Q_S \subseteq Q_s \circ A$). As in CSL, during shared memory access, ownership transfer may occur.

SAGL also provide FRAME and RESOURCE rules. The frame rule allows abstracting away part of the shared memory. The resource rule allows moving some memory from private to shared. There are some peculiarities about these two rules. The FRAME rule requires I to be precise, or, alternatively, $\mathsf{img}(G)$ and $\mathsf{dom}(A)$ to coincide. This required in order to establish the stability of $I * P_s$ with regard to $I^2 * A$ given the stability of P_s with regard to A (See Lemma 8.10). The RESOURCE rule for SAGL is limited. In order to promote memory from private to shared, you must have no shared memory. This means that you can only create shared memory from private memory once you abstract away the shared memory you had. A version of SAGL with a complete resource rule exists [29], but requires carrying an extra precise invariant to "mark" the boundaries of shared memory. It is interesting that at least we can support this FRAME rule without the need of a heavier framework. Because of this limitation of the RESOURCE rule we cannot derive a flexible frame rule for private memory, as we did for CSL. We can support such rule which is presented in Sec. 8.4.

8.3 Soundness

In this section, we present the soundness of SAGL with regard to the semantics of Chapter 3. It follows a similar structure of Sec. 6.3

8.3.1 With Regard to the Interleaved Semantics

In this section, we present the soundness proof with regard to the interleaved semantics from Sec. 3.7. The proof is structured around the following definition:

Definition 8.1. $A, G, P_s, Q_s \models \langle T, \sigma \rangle \rhd_0 Q$ always holds; $A, G, P_s, Q_s \models \langle T, \sigma \rangle \rhd_{n+1} Q$ holds if, and only if, the following are true:

1. T is either 0- or 1-atomic
2. $\neg \langle T, \sigma \rangle \longmapsto \mathsf{abort}$
3. $\neg \langle T, \sigma \rangle \longmapsto \mathsf{race}$
4. If $\langle T, \sigma \rangle \longmapsto \langle T', \sigma' \rangle$, then
   (a) If both T and T' are 1-atomic, then $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_n Q$
   (b) If both T and T' are 0-atomic, then $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_n Q$
   (c) If T is 0-atomic and T' is 1-atomic, then for all σ_1 and σ_2, such that $\sigma_2 = \sigma_1 \uplus \sigma'$ and $\sigma_1 \in P_s$, we have $A, G, P_s, Q_s \models \langle T', \sigma_2 \rangle \rhd_n Q$

(d) If T is 1-atomic and T' is 0-atomic, then for all σ'', such that $\sigma'' \in P_s$, there exists P_s', where $P_s \times P_s' \subseteq G$ and $P_s' \subseteq P_s' \circ A$, and a pair of states, σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in P_s'$, and $A, G, P_s', Q_s \models \langle T', \sigma_2 \rangle \triangleright_n Q$

5. If $T = \textbf{skip}$, then $P_s \subseteq Q_s$ and $\sigma \in Q$

We define $A, G, P_s, Q_s \models \langle T, \sigma \rangle \triangleright Q$ as $\forall n.\ A, G, P_s, Q_s \models \langle T, \sigma \rangle \triangleright_n Q$.

The sixtuple $A, G, P_s, Q_s \models \langle T, \sigma \rangle \triangleright Q$ ensures that each step performed by a program configuration has at most one ongoing atomic block execution (item 1), does not abort (item 2), and is not at a race condition (item 3). Furthermore, if it reaches a final configuration $\langle \textbf{skip}, \sigma' \rangle$, then σ' must satisfy post-condition Q, and shared pre-condition P_s must imply shared post-condition Q_s (item 5). This definition also manages proper access to private memory (item 4). If the current configuration has an ongoing atomic block execution (item (a)), then it already has a hold of the shared memory, and it can perform the step with out constraints. If the current configuration does not have an ongoing atomic block execution (item (b)), then no memory allocation or deallocation must happen to avoid a race-condition with the environment, which may have an ongoing atomic block execution performing memory allocation or deallocation. This constraint is enforced by the condition $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$. If the current program configuration is starting to execute a top-level atomic block (item (c)), then it must get a hold on the shared memory, assuming it satisfies P_s. If the current program configuration is completing the execution of a top-level atomic block (item (d)), then it must return the shared memory ensuring that it satisfies some P_s' that together with P_s imply G and is stable with regard to A (same constraints of the ATOMIC rule).

We present the soundness proof in three sections, following the order:

1. Auxiliary lemmas for SAGL sixtuples, as in Def. 8.1

2. Semantic rules, each corresponding to a syntactic rule from Fig. 8.3

3. Top level soundness theorem

Auxiliary lemmas. Given that programs may diverge, and since $A, G, P_s, Q_s \models \langle T, \sigma \rangle \rhd Q$ is defined in terms of itself, we used indexing to ensure this definition is well-founded. Lemma 8.2 allows greater flexibility when dealing with indexing.

Lemma 8.2. If $A, G, P_s, Q_s \models \langle T, \sigma \rangle \rhd_{n_1} Q$, and $n_2 \leq n_1$, then $A, G, P_s, Q_s \models \langle T, \sigma \rangle \rhd_{n_2} Q$

Proof. By induction over n_1. If $n_1 = 0$, then $n_2 = 0$ as well, by Def. 8.1, we conclude. If $n_1 > 0$, by Def. 8.1, assuming (a) $A, G, P_s, Q_s \models \langle T, \sigma \rangle \rhd_{n_1} Q$ and (b) $n_2 \leq n_1$, we have 5 cases:

- From (a), by Def. 8.1 (item 1), we know that T is 0- or 1-atomic
- From (a), by Def. 8.1 (item 2), we know that $\langle T, \sigma \rangle \longmapsto \mathsf{abort}$ is false
- From (a), by Def. 8.1 (item 3), we know that $\langle T, \sigma \rangle \longmapsto \mathsf{race}$ is false
- If (c) $\langle T, \sigma \rangle \longmapsto \langle T', \sigma' \rangle$, then we have 4 cases:
  - If both T and T' are 1-atomic, we need to show that $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_{(n_2-1)} Q$. From (a) and (c), by Def. 8.1 (item 4.a), we know that (d) $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_{(n_1-1)} Q$. Trivially, from (b), we know that (e) $n_2 - 1 \leq n_1 - 1$. From (d) and (e), by the induction hypothesis, we conclude
  - If both T and T' are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_{(n_2-1)} Q$. From (a) and (c), by Def. 8.1 (item 4.b), we know already that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, and also that (d) $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_{(n_1-1)} Q$. Trivially, from (b), we know that (e) $n_2 - 1 \leq n_1 - 1$. From (d) and (e), by the induction hypothesis, we conclude
  - If T is 0-atomic and T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (d) $\sigma_2 = \sigma_1 \uplus \sigma'$ and (e) $\sigma_1 \in P_s$, we have $A, G, P_s, Q_s \models \langle T', \sigma_2 \rangle \rhd_{(n_2-1)} Q$. From (a), (c), (d), and (e), by Def. 8.1 (item 4.c), we know that (f) $A, G, P_s, Q_s \models \langle T', \sigma_2 \rangle \rhd_{(n_1-1)} Q$. Trivially, from (b), we know that (g) $n_2 - 1 \leq n_1 - 1$. From (f) and (g), by the induction hypothesis, we conclude

  - If T is 1-atomic and T' is 0-atomic, we need to show that for all σ'', such that (d) $\sigma'' \in P_s$, there exists P_s', where $P_s \times P_s' \subseteq G$ and $P_s' \subseteq P_s' \circ A$, and a pair of states, σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in P_s'$, and $A, G, P_s', Q_s \models \langle T', \sigma_2 \rangle \rhd_{(n_2-1)} Q$. From (a), (c), and (d), by Def. 8.1 (item 4.d), we know there exists P_s'', where (e) $P_s \times P_s'' \subseteq G$ and (f) $P_s'' \subseteq P_s'' \circ A$, and σ_1' and σ_2', such that (g) $\sigma' = \sigma_1' \uplus \sigma_2'$, (h) $\sigma_1' \in P_s''$, and (i) $A, G, P_s'', Q_s \models \langle T', \sigma_2' \rangle \rhd_{(n_1-1)} Q$. Trivially, from (b), we know that (j) $n_2 - 1 \leq n_1 - 1$. From (i) and (j), by the induction hypothesis, we have (h) $A, G, P_s'', Q_s \models \langle T', \sigma_2' \rangle \rhd_{(n_2-1)} Q$. Instantiating the goal with P_s'', σ_1', and σ_2', from (e), (f), (g), (h), and (i), we conclude

- We assume (c) $T = \textbf{skip}$. From (a) and (c), by Def. 8.1 (item 5), we know that $P_s \subseteq Q_s$ and $\sigma \in Q$, and we conclude □

The following lemma allows the strengthening of A and P_s, as well as the weakening of G, Q_s, and Q in a SAGL sixtuple.

Lemma 8.3. If $A, G, P_s, Q_s \models \langle T, \sigma \rangle \rhd_n Q$, $A' \subseteq A$, $G \subseteq G'$, $P_s' \subseteq P_s$, $Q_s \subseteq Q_s'$, and $Q \subseteq Q'$, then $A', G', P_s', Q_s' \models \langle T, \sigma \rangle \rhd_n Q'$

Proof. By induction over n. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, assuming (a) $A, G, P_s, Q_s \models \langle T, \sigma \rangle \rhd_n Q$, (b) $A' \subseteq A$, (c) $G \subseteq G'$, (d) $P_s' \subseteq P_s$, (e) $Q_s \subseteq Q_s'$, and (f) $Q \subseteq Q'$, we have 5 cases:

- From (a), by Def. 8.1 (item 1), we know that T is 0- or 1-atomic
- From (a), by Def. 8.1 (item 2), we know that $\langle T, \sigma \rangle \longmapsto \text{abort}$ is false
- From (a), by Def. 8.1 (item 3), we know that $\langle T, \sigma \rangle \longmapsto \text{race}$ is false
- If (g) $\langle T, \sigma \rangle \longmapsto \langle T', \sigma' \rangle$, then we have 4 cases:
  - If both T and T' are 1-atomic, we need to show that $A', G', P_s', Q_s' \models \langle T', \sigma' \rangle \rhd_{n-1} Q'$. From (a) and (g), by Def. 8.1 (item 4.a), we know that (h) $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_{n-1} Q$. From (b), (c), (d), (e), (f), and (h), by the induction hypothesis, we conclude

  - If both T and T' are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $A', G', P_s', Q_s' \models \langle T', \sigma' \rangle \rhd_{n-1} Q'$. From (a) and (g), by Def. 8.1 (item 4.b), we know already that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, and also that
    (h) $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_{n-1} Q$. From (b), (c), (d), (e), (f), and (h), by the induction hypothesis, we conclude
  - If T is 0-atomic and T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (h) $\sigma_2 = \sigma_1 \uplus \sigma'$ and (i) $\sigma_1 \in P_s'$, we have $A', G', P_s', Q_s' \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q'$. From (i), and (d), we have (j) $\sigma_1 \in P_s$. From (a), (g), (h), and (j), by Def. 8.1 (item 4.c), we know that (h) $A, G, P_s, Q_s \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q$. From (b), (c), (d), (e), (f), and (h), by the induction hypothesis, we conclude
  - If T is 1-atomic and T' is 0-atomic, we need to show that for all σ'', such that (h) $\sigma'' \in P_s'$, there exists P_s'', where $P_s' \times P_s'' \subseteq G'$ and $P_s'' \subseteq P_s'' \circ A'$, and a pair of states, σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in P_s''$, and $A', G', P_s'', Q_s' \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q'$. From (h), and (d), we have (i) $\sigma'' \in P_s$. From (a), (g), and (i), by Def. 8.1 (item 4.d), we know there exists P_s''', where (j) $P_s \times P_s''' \subseteq G$ and (k) $P_s''' \subseteq P_s''' \circ A$, and σ_1' and σ_2', such that (l) $\sigma' = \sigma_1' \uplus \sigma_2'$, (m) $\sigma_1' \in P_s'''$, and (n) $A, G, P_s''', Q_s \models \langle T', \sigma_2' \rangle \rhd_{n-1} Q$. From (j), (d), and (c), we have (o) $P_s' \times P_s''' \subseteq G'$. From (k), and (b), we have (p) $P_s''' \subseteq P_s''' \circ A'$. From (b), (c), (e), (f), and (n), by the induction hypothesis, we have (q) $A', G', P_s''', Q_s' \models \langle T', \sigma_2' \rangle \rhd_{n-1} Q'$. Instantiating the goal with P_s''', σ_1', and σ_2', from (o), (p), (l), (m), and (q), we conclude

- We assume (g) $T = \textbf{skip}$. From (a) and (g), by Def. 8.1 (item 5), we know that (h) $P_s \subseteq Q_s$ and (i) $\sigma \in Q$. From (d), (e), and (h), we know that (j) $P_s' \subseteq Q_s'$. From (f) and (i), we know that (k) $\sigma \in Q'$. From (j) and (k), we conclude □

We can construct a SAGL sixtuple from **skip** using the following lemma.

Lemma 8.4. If $\sigma \in Q$, then $A, G, Q_s, Q_s \models \langle \textbf{skip}, \sigma \rangle \rhd_n Q$

Proof. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, we have 5 cases:

- By Def. 3.1, we know that **skip** is 0-atomic

- From the semantics, we know that $\langle \mathbf{skip}, \sigma \rangle \longmapsto \textsf{abort}$ is false
- From the semantics, we know that $\langle \mathbf{skip}, \sigma \rangle \longmapsto \textsf{race}$ is false
- From the semantics, we know that $\langle \mathbf{skip}, \sigma \rangle \longmapsto \langle T', \sigma' \rangle$ is false
- Since $\mathbf{skip} = \mathbf{skip}$, $Q_s \subseteq Q_s$, and $\sigma \in Q$, we conclude $\square$

The following lemma is used for sequential composition of sixtuples.

Lemma 8.5. If $A, G, P_s, P'_s \models \langle T, \sigma \rangle \rhd_n P'$, and, for all $\sigma' \in P'$, we have $A, G, P'_s, Q_s \models \langle c', \sigma' \rangle \rhd_n Q$, then

1. If $T = c$, then $A, G, P_s, Q_s \models \langle c; c', \sigma \rangle \rhd_n Q$
2. If $T = \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$, and $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$ is 0- or 1-atomic, then $A, G, P_s, Q_s \models \langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} (c; c'), \sigma \rangle \rhd_n Q$
3. If $T = \langle\!\langle T' \rangle\!\rangle_{\mathbf{a}} c$, then $A, G, P_s, Q_s \models \langle \langle\!\langle T' \rangle\!\rangle_{\mathbf{a}} (c; c'), \sigma \rangle \rhd_n Q$

Proof. By induction over n. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, we assume (a) $A, G, P_s, P'_s \models \langle T, \sigma \rangle \rhd_n P'$ and (b) for all $\sigma' \in P'$, we have $A, G, P'_s, Q_s \models \langle c', \sigma' \rangle \rhd_n Q$. We establish (c) for all $\sigma' \in P'$, we have $A, G, P'_s, Q_s \models \langle c', \sigma' \rangle \rhd_{n-1} Q$:

- We assume $\sigma' \in P'$, from (b), using Lemma 8.2, we conclude

Then we consider 3 cases:

- If $T = c$, then we need to show that $A, G, P_s, Q_s \models \langle c; c', \sigma \rangle \rhd_n Q$. By Def. 8.1, we have 5 cases:
  - By Def. 3.1, we know that $c; c'$ is 0-atomic
  - From (a), by Def. 8.1 (item 2), we know that (d) $\langle c, \sigma \rangle \longmapsto \textsf{abort}$ is false. From (d), and the semantics, we know that $\langle c; c', \sigma \rangle \longmapsto \textsf{abort}$ is also false
  - From the semantics, we know that $\langle c; c', \sigma \rangle \longmapsto \textsf{race}$ is false
  - If (d) $\langle c; c', \sigma \rangle \longmapsto \langle T', \sigma' \rangle$, from the semantics, we have 4 cases:

    * If $c = \mathbf{skip}$, $T' = c'$, and $\sigma' = \sigma$, since both $\mathbf{skip}; c'$ and c' are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, which holds trivially, and $A, G, P_s, Q_s \models \langle c', \sigma' \rangle \rhd_{n-1} Q$. From (a), by Def. 8.1 (item 5), we know that (e) $P_s \subseteq P_s'$ and (f) $\sigma' \in P'$. From (f) and (c), we have (g) $A, G, P_s', Q_s \models \langle c', \sigma' \rangle \rhd_{n-1} Q$. From (e), and (g), using Lemma 8.3, we conclude
    * If $T' = c''; c'$, and (e) $\langle c, \sigma \rangle \longmapsto \langle c'', \sigma' \rangle$, given that both $c; c'$ and $c''; c'$ are 0-atomic, we need to show $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $A, G, P_s, Q_s \models \langle c''; c', \sigma' \rangle \rhd_{n-1} Q$. From (a) and (e), by Def. 8.1 (item 4.b), we have (f) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and (g) $A, G, P_s, P_s' \models \langle c'', \sigma' \rangle \rhd_{n-1} P'$. From (g) and (c), using the induction hypothesis (item 1), we have (h) $A, G, P_s, Q_s \models \langle c''; c', \sigma' \rangle \rhd_{n-1} Q$. From (f) and (h), we conclude
    * If $c = \mathbf{S}[\, c_1 \,\|\, c_2 \,]$, $T' = \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,]; c')$, and $\sigma' = \sigma$, since both $c; c'$ and $\langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,]; c')$ are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, which holds trivially, and $A, G, P_s, Q_s \models \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,]; c'), \sigma \rangle \rhd_{n-1} Q$. From the semantics, we know that (e) $\langle c, \sigma \rangle \longmapsto \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,]), \sigma \rangle$. From (a) and (e), by Def. 8.1 (item 4.b), we know that (f) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma)$ and (g) $A, G, P_s, P_s' \models \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}}(\mathbf{S}[\,\mathbf{skip}\,]), \sigma \rangle \rhd_{n-1} P'$. From (g) and (c), using the induction hypothesis (item 2), we conclude
    * If $c = \mathbf{S}[\,\mathbf{atomic}\ c''\,]$, $T' = \langle\!\langle c'' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\mathbf{skip}\,]; c')$, and $\sigma' = \sigma$, since $c; c'$ is 0-atomic and $\langle\!\langle c'' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\mathbf{skip}\,]; c')$ is 1-atomic, we need to show that for all σ_1 and σ_2, such that (e) $\sigma_2 = \sigma_1 \uplus \sigma$, and (f) $\sigma_1 \in P_s$, we have $A, G, P_s, Q_s \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q$. From the semantics, we know that (g) $\langle c, \sigma \rangle \longmapsto \langle \langle\!\langle c'' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\mathbf{skip}\,]), \sigma \rangle$. From (a), (g), (e), and (f), by Def. 8.1 (item 4.c), we have (h) $A, G, P_s, P_s' \models \langle \langle\!\langle c'' \rangle\!\rangle_{\mathbf{a}}(\mathbf{S}[\,\mathbf{skip}\,]), \sigma_2 \rangle \rhd_{n-1} P'$. From (h) and (c), using the induction hypothesis (item 3), we conclude
  - We know that $c; c' \neq \mathbf{skip}$

- If $T = \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$, and (d) $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}} c$ is 0- or 1-atomic, then we need to show that $A, G, P_s, Q_s \models \langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}(c; c'), \sigma \rangle \rhd_n Q$. By Def. 8.1, we have 5 cases:

– From (d), by Def. 3.1, we know that $\llangle T_1, T_2 \rrangle_{\mathrm{p}}(c; c')$ is 0- or 1-atomic

– From (a), by Def. 6.3 (item 2), we know that (e) $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}} c, \sigma \rangle \longmapsto \mathsf{abort}$ is false. From (e), and the semantics, we know that $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}}(c; c'), \sigma \rangle \longmapsto \mathsf{abort}$ is also false

– From (a), by Def. 6.3 (item 3), we know that (e) $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}} c, \sigma \rangle \longmapsto \mathsf{race}$ is false. From (e), and the semantics, we know that $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}}(c; c'), \sigma \rangle \longmapsto \mathsf{race}$ is also false

– If (e) $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}}(c; c'), \sigma \rangle \longmapsto \langle T', \sigma' \rangle$, from the semantics, we have 3 cases:

  ∗ If $T_1 = \mathbf{skip}$, $T_2 = \mathbf{skip}$, $T' = c; c'$, and $\sigma' = \sigma$, since both $\llangle \mathbf{skip}, \mathbf{skip} \rrangle_{\mathrm{p}}(c; c')$ and $c; c'$ are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, which holds trivially, and $A, G, P_s, Q_s \models \langle c; c', \sigma \rangle \rhd_{n-1} Q$. From the semantics, we know that (f) $\langle \llangle \mathbf{skip}, \mathbf{skip} \rrangle_{\mathrm{p}} c, \sigma \rangle \longmapsto \langle c, \sigma \rangle$. From (a) and (f), by Def. 8.1 (item 4.b), we know that (g) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma)$ and (h) $A, G, P_s, P'_s \models \langle c, \sigma \rangle \rhd_{n-1} P'$. From (h), (c), by the induction hypothesis (item 1), we conclude

  ∗ If $T' = \llangle T'_1, T_2 \rrangle_{\mathrm{p}}(c; c')$, and (f) $\langle T_1, \sigma \rangle \longmapsto \langle T'_1, \sigma' \rangle$, we have 4 cases:

    · If both T_1 and T'_1 are 1-atomic, we have to show that $A, G, P_s, Q_s \models \langle \llangle T'_1, T_2 \rrangle_{\mathrm{p}}(c; c'), \sigma' \rangle \rhd_{n-1} Q$. From (f), and the semantics, we know (g) $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}} c, \sigma \rangle \longmapsto \langle \llangle T'_1, T_2 \rrangle_{\mathrm{p}} c, \sigma' \rangle$. From (a) and (g), by Def. 8.1 (item 4.a), we know (h) $A, G, P_s, P'_s \models \langle \llangle T'_1, T_2 \rrangle_{\mathrm{p}} c, \sigma' \rangle \rhd_{n-1} P'$. From (h) and (c), using the induction hypothesis (item 2), we conclude

    · If both T_1 and T'_1 are 0-atomic, we have to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $A, G, P_s, Q_s \models \langle \llangle T'_1, T_2 \rrangle_{\mathrm{p}}(c; c'), \sigma' \rangle \rhd_{n-1} Q$. From (f), and the semantics, we know (g) $\langle \llangle T_1, T_2 \rrangle_{\mathrm{p}} c, \sigma \rangle \longmapsto \langle \llangle T'_1, T_2 \rrangle_{\mathrm{p}} c, \sigma' \rangle$. From (a) and (g), by Def. 8.1 (item 4.b), we know (h) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and (i) $A, G, P_s, P'_s \models \langle \llangle T'_1, T_2 \rrangle_{\mathrm{p}} c, \sigma' \rangle \rhd_{n-1} P'$. From (i) and (c), using the induction hypothesis (item 2), we know
    (j) $A, G, P_s, Q_s \models \langle \llangle T'_1, T_2 \rrangle_{\mathrm{p}}(c; c'), \sigma' \rangle \rhd_{n-1} Q$. From (h) and (j), we conclude

· If T_1 is 0-atomic and T_1' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (g) $\sigma_2 = \sigma_1 \uplus \sigma'$ and (h) $\sigma_1 \in P_s$, we have $A, G, P_s, Q_s \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathbf{p}}(c; c'), \sigma_2\rangle \rhd_{n-1} Q$. From (f), and the semantics, we know (i) $\langle\langle\!\langle T_1, T_2\rangle\!\rangle_{\mathbf{p}}c, \sigma\rangle \longmapsto \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathbf{p}}c, \sigma'\rangle$. From (a), (i), (g), and (h), by Def. 8.1 (item 4.c), we know that (j) $A, G, P_s, P_s' \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathbf{p}}c, \sigma_2\rangle \rhd_{n-1} P'$. From (j) and (c), using the induction hypothesis (item 2), we conclude

· If T_1 is 1-atomic and T_1' is 0-atomic, we need to show that for all σ'', such that (g) $\sigma'' \in P_s$, there exists P_s'', where $P_s \times P_s'' \subseteq G$ and $P_s'' \subseteq P_s'' \circ A$, and a pair of states, σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in P_s''$, and $A, G, P_s'', Q_s \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathbf{p}}(c; c'), \sigma_2\rangle \rhd_{n-1} Q$. From (f), and the semantics, we know (h) $\langle\langle\!\langle T_1, T_2\rangle\!\rangle_{\mathbf{p}}c, \sigma\rangle \longmapsto \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathbf{p}}c, \sigma'\rangle$. From (a), (h), and (g), by Def. 8.1 (item 4.d), we know there exists P_s''', where (i) $P_s \times P_s''' \subseteq G$ and (j) $P_s''' \subseteq P_s''' \circ A$, and a pair of states, σ_1' and σ_2', such that (k) $\sigma' = \sigma_1' \uplus \sigma_2'$, (l) $\sigma_1' \in P_s'''$, and (m) $A, G, P_s''', P_s' \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathbf{p}}c, \sigma_2'\rangle \rhd_{n-1} P'$. From (m) and (c), using the induction hypothesis (item 2), we know (n) $A, G, P_s''', Q_s \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathbf{p}}(c; c'), \sigma_2'\rangle \rhd_{n-1} Q$. Instantiating the goal with P_s''', σ_1', and σ_2', from (i), (j), (k), (l), and (n), we conclude

∗ If $T' = \langle\!\langle T_1, T_2'\rangle\!\rangle_{\mathbf{p}}(c; c')$, and (f) $\langle T_2, \sigma\rangle \longmapsto \langle T_2', \sigma'\rangle$, the proof is symmetric to the previous case

– We know that $\langle\!\langle T_1, T_2\rangle\!\rangle_{\mathbf{p}}(c; c') \neq \mathbf{skip}$

- If $T = \langle\!\langle T'\rangle\!\rangle_{\mathbf{a}}c$, then we need to show that $A, G, P_s, Q_s \models \langle\langle\!\langle T'\rangle\!\rangle_{\mathbf{a}}(c; c'), \sigma\rangle \rhd_n Q$. By Def. 6.3, we have 5 cases:

  – By Def. 3.1, we know that $\langle\!\langle T'\rangle\!\rangle_{\mathbf{a}}(c; c')$ is 1-atomic

  – From (a), by Def. 6.3 (item 2), we know that (d) $\langle\langle\!\langle T'\rangle\!\rangle_{\mathbf{a}}c, \sigma\rangle \longmapsto \mathsf{abort}$ is false. From (d), and the semantics, we know that $\langle\langle\!\langle T'\rangle\!\rangle_{\mathbf{a}}(c; c'), \sigma\rangle \longmapsto \mathsf{abort}$ is also false

  – From (a), by Def. 6.3 (item 3), we know that (d) $\langle\langle\!\langle T'\rangle\!\rangle_{\mathbf{a}}c, \sigma\rangle \longmapsto \mathsf{race}$ is false. From (d), and the semantics, we know that $\langle\langle\!\langle T'\rangle\!\rangle_{\mathbf{a}}(c; c'), \sigma\rangle \longmapsto \mathsf{race}$ is also false

- If (d) $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}}(c;c'),\sigma\rangle \longmapsto \langle T'',\sigma'\rangle$, from the semantics, we have 2 cases:
  - $*$ If $T' = \mathbf{skip}$, $T'' = c;c'$, and $\sigma' = \sigma$, since $\langle\!\langle \mathbf{skip}\rangle\!\rangle_{\mathsf{a}}(c;c')$ is 1-atomic and $c;c'$ is 0-atomic, we need to show that for all σ'', such that (e) $\sigma'' \in P_s$, there exists P_s'', where $P_s \times P_s'' \subseteq G$ and $P_s'' \subseteq P_s''\circ A$, and a pair of states, σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in P_s''$, and $A, G, P_s'', Q_s \models \langle c;c',\sigma_2\rangle \rhd_{n-1} Q$. From the semantics, we know that (f) $\langle\langle\!\langle \mathbf{skip}\rangle\!\rangle_{\mathsf{a}} c,\sigma\rangle \longmapsto \langle c,\sigma\rangle$. From (a), (f), and (e), by Def. 8.1 (item 4.d), we know that exists P_s''', where (g) $P_s \times P_s''' \subseteq G$ and (h) $P_s''' \subseteq P_s'''\circ A$, and a pair of states, σ_1' and σ_2', such that (i) $\sigma = \sigma_1' \uplus \sigma_2'$, (j) $\sigma_1' \in P_s'''$, and (k) $A, G, P_s''', P_s' \models \langle c,\sigma_2'\rangle \rhd_{n-1} P'$. From (k), (c), by the induction hypothesis (item 1), we have (l) $A, G, P_s''', Q_s \models \langle c;c',\sigma_2'\rangle \rhd_{n-1} Q$. Instantiating the goal with P_s''', σ_1', and σ_2', from (g), (h), (i), (j), and (l), we conclude
  - $*$ If $T'' = \langle\!\langle T'''\rangle\!\rangle_{\mathsf{a}}(c;c')$, and (e) $\langle T',\sigma\rangle \longmapsto \langle T''',\sigma'\rangle$, given that both $\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}}(c;c')$ and $\langle\!\langle T'''\rangle\!\rangle_{\mathsf{a}}(c;c')$ are 1-atomic, we need to show $A, G, P_s, Q_s \models \langle\langle\!\langle T'''\rangle\!\rangle_{\mathsf{a}}(c;c'),\sigma'\rangle \rhd_{n-1} Q$. From (e), and the semantics, we know (f) $\langle\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}} c,\sigma\rangle \longmapsto \langle\langle\!\langle T'''\rangle\!\rangle_{\mathsf{a}} c,\sigma'\rangle$. From (a) and (f), by Def. 8.1 (item 4.a), then (g) $A, G, P_s, P_s' \models \langle\langle\!\langle T'''\rangle\!\rangle_{\mathsf{a}} c,\sigma'\rangle \rhd_{n-1} P'$. From (g) and (c), using the induction hypothesis (item 3), we conclude
- We know that $\langle\!\langle T'\rangle\!\rangle_{\mathsf{a}}(c;c') \neq \mathbf{skip}$ □

Conditional commands can be introduced using the following lemma.

Lemma 8.6. If $A, G, P_s, Q_s \models \langle c,\sigma\rangle \rhd_n Q$, then

1. If $\sigma \in \lfloor b\rfloor$, then $A, G, P_s, Q_s \models \langle \mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{else}\ c',\sigma\rangle \rhd_n Q$
2. If $\sigma \in \lfloor \neg b\rfloor$, then $A, G, P_s, Q_s \models \langle \mathbf{if}\ b\ \mathbf{then}\ c'\ \mathbf{else}\ c,\sigma\rangle \rhd_n Q$

Proof. From assumption (a) $A, G, P_s, Q_s \models \langle c,\sigma\rangle \rhd_n Q$ we have 2 cases:

- We assume (b) $\sigma \in \lfloor b\rfloor$. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, we have 5 cases:

- By Def. 3.1, we know that (c) **if** b **then** c **else** c' is 0-atomic
- From the semantics, we know that $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \sigma \rangle \longmapsto \textsf{abort}$ is false if $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \sigma \rangle \longrightarrow \textsf{abort}$ is false. From the sequential semantics, we know $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \sigma \rangle \longrightarrow \textsf{abort}$ is false if there exists z such that $[\![b]\!]_\sigma = z$. From (b), we know $[\![b]\!]_\sigma = \textit{true}$ and conclude
- From the semantics, we know that $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \sigma \rangle \longmapsto \textsf{race}$ is false
- From the semantics, given (b), we know that $\langle \textbf{if } b \textbf{ then } c \textbf{ else } c', \sigma \rangle \longmapsto \langle c, \sigma \rangle$. From (c), and since c is 0-atomic, we need to show that $\textsf{dom}(\sigma) = \textsf{dom}(\sigma)$ and $A, G, P_s, Q_s \models \langle c, \sigma \rangle \rhd_{n-1} Q$. From (a), using Lemma 8.2, we conclude
- We know that **if** b **then** c **else** $c' \neq$ **skip**

• We assume (b) $\sigma \in \lfloor \neg b \rfloor$. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, we have 5 cases:

- By Def. 3.1, we know that (c) **if** b **then** c' **else** c is 0-atomic
- From the semantics, we know that $\langle \textbf{if } b \textbf{ then } c' \textbf{ else } c, \sigma \rangle \longmapsto \textsf{abort}$ is false if $\langle \textbf{if } b \textbf{ then } c' \textbf{ else } c, \sigma \rangle \longrightarrow \textsf{abort}$ is false. From the sequential semantics, we know $\langle \textbf{if } b \textbf{ then } c' \textbf{ else } c, \sigma \rangle \longrightarrow \textsf{abort}$ is false if there exists z such that $[\![b]\!]_\sigma = z$. From (b), we know $[\![b]\!]_\sigma = \textit{false}$ and conclude
- From the semantics, we know that $\langle \textbf{if } b \textbf{ then } c' \textbf{ else } c, \sigma \rangle \longmapsto \textsf{race}$ is false
- From the semantics, given (b), we know that $\langle \textbf{if } b \textbf{ then } c' \textbf{ else } c, \sigma \rangle \longmapsto \langle c, \sigma \rangle$. From (c), and since c is 0-atomic, we need to show that $\textsf{dom}(\sigma) = \textsf{dom}(\sigma)$ and $A, G, P_s, Q_s \models \langle c, \sigma \rangle \rhd_{n-1} Q$. From (a), using Lemma 8.2, we conclude
- We know that **if** b **then** c' **else** $c \neq$ **skip** $\square$

A loop command can be introduced using the following lemma.

Lemma 8.7. If $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, $\sigma \in P$, and, for all $\sigma' \in P \cap \lfloor b \rfloor$, we have $A, G, P_s, P_s \models \langle c, \sigma' \rangle \rhd_n P$, then $A, G, P_s, P_s \models \langle \textbf{while } b \textbf{ do } c, \sigma \rangle \rhd_n (P \cap \lfloor \neg b \rfloor)$

Proof. By induction over n. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, assuming (a) $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, (b) $\sigma \in P$, and, (c) for all $\sigma' \in P \cap \lfloor b \rfloor$, we have $A, G, P_s, P_s \models \langle c, \sigma' \rangle \rhd_n P$, we have 5 cases:

- By Def. 3.1, we know that (d) **while** b **do** c is 0-atomic
- From the semantics, we know that $\langle$**while** b **do** $c, \sigma\rangle \longmapsto \mathsf{abort}$ is false
- From the semantics, we know that $\langle$**while** b **do** $c, \sigma\rangle \longmapsto \mathsf{race}$ is false
- From the semantics, $\langle$**while** b **do** $c, \sigma\rangle \longmapsto \langle$**if** b **then** $(c;$ **while** b **do** $c)$ **else skip**$, \sigma\rangle$. From (d), and since **if** b **then** $(c;$ **while** b **do** $c)$ **else skip** is 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma)$ and $A, G, P_s, P_s \models \langle$**if** b **then** $(c;$ **while** b **do** $c)$ **else skip**$, \sigma\rangle \rhd_{n-1} (P \cap \lfloor \neg b \rfloor)$. From (a) and (b), we know (e) $\sigma \in \lfloor b \rfloor \cup \lfloor \neg b \rfloor$. From (e), we have two cases:
  - We assume (f) $\sigma \in \lfloor b \rfloor$. From (b) and (f), we know (g) $\sigma \in P \cap \lfloor b \rfloor$. We establish (h) for all $\sigma' \in P \cap \lfloor b \rfloor$, we have $A, G, P_s, P_s \models \langle c, \sigma' \rangle \rhd_{n-1} P$:
    * We assume $\sigma' \in P \cap \lfloor b \rfloor$, from (c), using Lemma 8.2, we conclude

    From (g) and (h), we know (i) $A, G, P_s, P_s \models \langle c, \sigma \rangle \rhd_{n-1} P$. We establish (j) for all $\sigma' \in P$, we have $A, G, P_s, P_s \models \langle$**while** b **do** $c, \sigma'\rangle \rhd_{n-1} (P \cap \lfloor \neg b \rfloor)$:
    * We assume $\sigma' \in P$, with (a) and (h), using the induction hypothesis, we conclude

    From (i) and (j), using Lemma 8.5 (item 1), we get
    (k) $A, G, P_s, P_s \models \langle c;$ **while** b **do** $c, \sigma\rangle \rhd_{n-1} (P \cap \lfloor \neg b \rfloor)$. From (f) and (k), using Lemma 8.6 (item 1), we conclude
  - We assume (f) $\sigma \in \lfloor \neg b \rfloor$. From (b) and (f), we know (g) $\sigma \in P \cap \lfloor \neg b \rfloor$. From (g), using Lemma 8.4, we know (h) $A, G, P_s, P_s \models \langle$**skip**$, \sigma\rangle \rhd_{n-1} (P \cap \lfloor \neg b \rfloor)$. From (f) and (h), using Lemma 8.6 (item 2), we conclude
- We know that **while** b **do** $c \neq$ **skip** □

The following lemma is used for parallel composition of sixtuples.

Lemma 8.8. If $P_{s1} \subseteq P_{s1} \circ A_1$, $P_{s2} \subseteq P_{s2} \circ A_2$, $A_1, G \cap A_2, P_{s1}, P_{s2} \models \langle T_1, \sigma_1 \rangle \rhd_n Q_1$, $A_2, G \cap A_1, P_{s2}, Q_{s2} \models \langle T_2, \sigma_2 \rangle \rhd_n Q_2$, and $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}\mathbf{skip}$ is 0- or 1-atomic, then $A_1 \cap A_2, G, P_{s1} \cap P_{s2}, Q_{s1} \cap Q_{s2} \models \langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}\mathbf{skip}, \sigma_1 \uplus \sigma_2 \rangle \rhd_n (Q_1 * Q_2)$

Proof. By induction over n. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, assuming (a) $P_{s1} \subseteq P_{s1} \circ A_1$, (b) $P_{s2} \subseteq P_{s2} \circ A_2$, (c) $A_1, G \cap A_2, P_{s1}, Q_{s1} \models \langle T_1, \sigma_1 \rangle \rhd_n Q_1$, (d) $A_2, G \cap A_1, P_{s2}, Q_{s2} \models \langle T_2, \sigma_2 \rangle \rhd_n Q_2$, and (e) $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}\mathbf{skip}$ is 0- or 1-atomic, we have 5 cases:

- From (e), we conclude
- From the semantics, we know that $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}\mathbf{skip}, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \mathsf{abort}$ if either:
  - $\langle T_1, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \mathsf{abort}$. From (c), by Def. 8.1 (item 2), we know (f) $\langle T_1, \sigma_1 \rangle \longmapsto \mathsf{abort}$ is false. From (f), using Lemma 3.22 (item 1), we conclude
  - or, $\langle T_2, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \mathsf{abort}$. From (d), by Def. 8.1 (item 2), we know (g) $\langle T_2, \sigma_2 \rangle \longmapsto \mathsf{abort}$ is false. From (g), using Lemma 3.22 (item 1), we conclude
- From the semantics, we know that $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}\mathbf{skip}, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \mathsf{race}$ if either:
  - $\langle T_1, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \mathsf{race}$. From (c), by Def. 8.1 (item 3), we know (h) $\langle T_1, \sigma_1 \rangle \longmapsto \mathsf{race}$ is false. From (f) and (h), using Lemma 3.22 (item 2), we conclude
  - or, $\langle T_2, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \mathsf{race}$. From (d), by Def. 8.1 (item 3), we know (i) $\langle T_2, \sigma_2 \rangle \longmapsto \mathsf{race}$ is false. From (g) and (i), using Lemma 3.22 (item 2), we conclude
  - or, $T_1 = \mathbf{T}_1[\,c_1\,]$, $T_2 = \mathbf{T}_2[\,c_2\,]$, $\langle c_1, \sigma_1 \uplus \sigma_2 \rangle \underset{\delta_1}{\longrightarrow} \langle c_1', \sigma' \rangle$, $\langle c_2, \sigma' \rangle \underset{\delta_2}{\longrightarrow} \kappa$ and $\delta_1 \not\smile \delta_2$. By contradiction, we will assume (h) $\langle c_1, \sigma_1 \uplus \sigma_2 \rangle \underset{\delta_1}{\longrightarrow} \langle c_1', \sigma' \rangle$, and (i) $\langle c_2, \sigma' \rangle \underset{\delta_2}{\longrightarrow} \kappa$ in order to obtain $\delta_1 \smile \delta_2$. From the semantics, and (f), we know (j) $\langle c1, \sigma_1 \rangle \longrightarrow \mathsf{abort}$ is false. From (j) and (h), using Lemma 3.16 (item 2), we know there exists σ_1' such that (k) $\sigma' = \sigma_1' \uplus \sigma_2$ and (l) $\langle c1, \sigma_1 \rangle \underset{\delta_1}{\longrightarrow} \langle c_1', \sigma_1' \rangle$. From the semantics, and (g), we know (m) $\langle c2, \sigma_2 \rangle \longrightarrow \mathsf{abort}$ is false. From (m), using Lemma 3.16 (item 1), we know (n) $\langle c2, \sigma_1' \uplus \sigma_2 \rangle \longrightarrow \mathsf{abort}$ is false. From (i), (k), and (n), we know there exists c_2' and σ'' such that (o) $\kappa = \langle c_2', \sigma'' \rangle$. From (m),

(i) and (o), using Lemma 3.16 (item 2), we know there exists σ'_2 such that (p) $\sigma'' = \sigma'_1 \uplus \sigma'_2$ and (q) $\langle c_2, \sigma_2 \rangle \xrightarrow[\delta_2]{} \langle c'_2, \sigma'_2 \rangle$. From (l), using Remark 3.10 (item 2), by Def. 3.4, we know that (r) $\delta_1 \subseteq (\varnothing, \mathsf{dom}(\sigma_1) \cup \mathsf{dom}(\sigma'_1))$. From (q), using Remark 3.10 (item 2), by Def. 3.4, we know that (s) $\delta_2 \subseteq (\varnothing, \mathsf{dom}(\sigma_2) \cup \mathsf{dom}(\sigma'_2))$. Then, we have 2 cases:

  * If $\mathbf{T}_1[\,c_1\,]$ is 0-atomic, then from (c) and (l), by Def. 8.1 (item 4.b), we know that (t) $\mathsf{dom}(\sigma_1) = \mathsf{dom}(\sigma'_1)$. From (r) and (t), we know that (u) $\delta_1 \subseteq (\varnothing, \mathsf{dom}(\sigma'_1))$. From (k), (p), we know that (v) $\mathsf{dom}(\sigma'_1) \cap (\mathsf{dom}(\sigma_2) \cup \mathsf{dom}(\sigma'_2)) = \varnothing$. From (v), (u), and (s), we know that $\delta_1.ws \cap (\delta_2.rs \cup \delta_2.ws) = \varnothing$ and conclude
  * If $\mathbf{T}_1[\,c_1\,]$ is 1-atomic, from (e) we know that (t) $\mathbf{T}_2[\,c_2\,]$ is 0-atomic. From (d), (q), and (t), by Def. 8.1 (item 4.b), we know that (u) $\mathsf{dom}(\sigma_2) = \mathsf{dom}(\sigma'_2)$. From (s) and (u), we know that (v) $\delta_2 \subseteq (\varnothing, \mathsf{dom}(\sigma_2))$. From (k), we know that (x) $(\mathsf{dom}(\sigma_1) \cup \mathsf{dom}(\sigma'_1)) \cap \mathsf{dom}(\sigma_2) = \varnothing$. From (x), (r), and (v), we know that $\delta_1.ws \cap (\delta_2.rs \cup \delta_2.ws) = \varnothing$ and conclude

– or, $T_1 = \mathbf{T}_1[\,c_1\,]$, $T_2 = \mathbf{T}_2[\,c_2\,]$, $\langle c_2, \sigma_1 \uplus \sigma_2 \rangle \xrightarrow[\delta_2]{} \langle c'_2, \sigma' \rangle$, $\langle c_1, \sigma' \rangle \xrightarrow[\delta_1]{} \kappa$ and $\delta_2 \not\smile \delta_1$. The proof is symmetric to the previous case

- From the semantics, if (h) $\langle \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \langle T', \sigma' \rangle$, then either:

  – $T_1 = \mathbf{skip}$, $T_2 = \mathbf{skip}$, $T' = \mathbf{skip}$, and $\sigma' = \sigma_1 \uplus \sigma_2$. Since both $\langle\!\langle \mathbf{skip}, \mathbf{skip} \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}$ and $\mathbf{skip}$ are 0-atomic, we need to show that $\mathsf{dom}(\sigma') = \mathsf{dom}(\sigma_1 \uplus \sigma_2)$, which holds trivially, and that $A_1 \cap A_2, G, P_{s1} \cap P_{s2}, Q_{s1} \cap Q_{s2} \models \langle \mathbf{skip}, \sigma' \rangle \rhd_{n-1} (Q_1 * Q_2)$. From (c), by Def. 8.1 (item 5), we know (i) $P_{s1} \subseteq Q_{s1}$ and (j) $\sigma_1 \in Q_1$. From (d), by Def. 8.1 (item 5), we know (k) $P_{s2} \subseteq Q_{s2}$ and (l) $\sigma_2 \in Q_2$. From (i) and (k), we know (m) $P_{s1} \cap P_{s2} \subseteq Q_{s1} \cap Q_{s2}$ From (j) and (l), we know (n) $\sigma' \in Q_1 * Q_2$. From (b), using Lemma 8.4, we have (o) $A_1 \cap A_2, G, Q_{s1} \cap Q_{s2}, Q_{s1} \cap Q_{s2} \models \langle \mathbf{skip}, \sigma' \rangle \rhd_{n-1} (Q_1 * Q_2)$. From (o), and (m), using Lemma 8.3, we conclude.

  – or, $T' = \langle\!\langle T'_1, T_2 \rangle\!\rangle_{\mathrm{p}} \mathbf{skip}$ and (i) $\langle T_1, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \langle T'_1, \sigma' \rangle$. From (f) and (i), using Lemma 3.22 (item 3), we know that exists σ'_1 such that $\sigma' = \sigma'_1 \uplus \sigma_2$ and (j)

$\langle T_1, \sigma_1\rangle \longmapsto \langle T_1', \sigma_1'\rangle$. From (e), and (i), using Remark 3.17, we have 5 cases:

* If both T_1 and T_1' are 1-atomic, and T_2 is 0-atomic, we have to show that $A_1{\cap}A_2, G, P_{s1}{\cap}P_{s2}, Q_{s1}{\cap}Q_{s2} \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}, \sigma_1' \uplus \sigma_2\rangle \rhd_{n-1} (Q_1{*}Q_2)$. From (c) and (j), by Def. 8.1 (item 4.a), we know that (k) $A_1, G{\cap}A_2, P_{s1}, Q_{s1} \models \langle T_1', \sigma_1'\rangle \rhd_{n-1} Q_1$. From (d), using Lemma 8.2, we know that (l) $A_2, G{\cap}A_1, P_{s2}, Q_{s2} \models \langle T_2, \sigma_2\rangle \rhd_{n-1} Q_2$. From (a), (b), (k), and (l), using the induction hypothesis, we conclude
* If both T_1 and T_1' are 0-atomic, and T_2 is 1-atomic, we have to show that $A_1{\cap}A_2, G, P_{s1}{\cap}P_{s2}, Q_{s1}{\cap}Q_{s2} \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}, \sigma_1' \uplus \sigma_2\rangle \rhd_{n-1} (Q_1{*}Q_2)$. From (c) and (j), by Def. 8.1 (item 4.b), we know that (k) $\mathsf{dom}(\sigma_1) = \mathsf{dom}(\sigma_1')$ and (l) $A_1, G{\cap}A_2, P_{s1}, Q_{s1} \models \langle T_1', \sigma_1'\rangle \rhd_{n-1} Q_1$. From (d), using Lemma 8.2, we know that (m) $A_2, G{\cap}A_1, P_{s2}, Q_{s2} \models \langle T_2, \sigma_2\rangle \rhd_{n-1} Q_2$. From (a), (b), (l), and (m), using the induction hypothesis, we conclude
* If all T_1, T_1', and T_2 are 0-atomic, we have to show that $\mathsf{dom}(\sigma_1 \uplus \sigma_2) = \mathsf{dom}(\sigma_1' \uplus \sigma_2)$ and $A_1 \cap A_2, G, P_{s1} \cap P_{s2}, Q_{s1} \cap Q_{s2} \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}, \sigma_1' \uplus \sigma_2\rangle \rhd_{n-1} (Q_1{*}Q_2)$. From (c) and (j), by Def. 8.1 (item 4.b), we know that (k) $\mathsf{dom}(\sigma_1) = \mathsf{dom}(\sigma_1')$ and (l) $A_1, G{\cap}A_2, P_{s1}, Q_{s1} \models \langle T_1', \sigma_1'\rangle \rhd_{n-1} Q_1$. From (d), using Lemma 8.2, we know that (m) $A_2, G{\cap}A_1, P_{s2}, Q_{s2} \models \langle T_2, \sigma_2\rangle \rhd_{n-1} Q_2$. From (a), (b), (l), and (m), using the induction hypothesis, we know (n) $A_1 \cap A_2, G, P_{s1} \cap P_{s2}, Q_{s1} \cap Q_{s2} \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}, \sigma_1' \uplus \sigma_2\rangle \rhd_{n-1} (Q_1 * Q_2)$. From (k), we know that (o) $\mathsf{dom}(\sigma_1 \uplus \sigma_2) = \mathsf{dom}(\sigma_1' \uplus \sigma_2)$. From (n) and (o), we conclude
* If both T_1 and T_2 are 0-atomic, and T_1' is 1-atomic, we have to show that for all σ_1'' and σ_2'', such that (k) $\sigma_2'' = \sigma_1'' \uplus \sigma'$ and (l) $\sigma_1'' \in P_{s1}{\cap}P_{s2}$, we have $A_1{\cap}A_2, G, P_{s1}{\cap}P_{s2}, Q_{s1}{\cap}Q_{s2} \models \langle\langle\!\langle T_1', T_2\rangle\!\rangle_{\mathsf{p}}\mathbf{skip}, \sigma_2''\rangle \rhd_{n-1} (Q_1{*}Q_2)$. From (c), (l) and (j), by Def. 8.1 (item 4.c), we know that (m) $A_1, G{\cap}A_2, P_{s1}, Q_{s1} \models \langle T_1', \sigma_1'' \uplus \sigma_1'\rangle \rhd_{n-1} Q_1$. From (d), using Lemma 8.2, we know that (n) $A_2, G{\cap}A_1, P_{s2}, Q_{s2} \models \langle T_2, \sigma_2\rangle \rhd_{n-1} Q_2$. From (a), (b), (m), and (n), by the induction hypothesis, we conclude

    * If both T'_1 and T_2 are 0-atomic, and T_1 is 1-atomic, we have to show that for all σ'', such that (k) $\sigma'' \in P_{s1} \cap P_{s2}$, there exists P'_s, where $(P_{s1} \cap P_{s2}) \times P'_s \subseteq G$ and $P'_s \subseteq P'_s \circ (A_1 \cap A_2)$, and a pair of states, σ''_1 and σ''_2, such that $\sigma' = \sigma''_1 \uplus \sigma''_2$, $\sigma''_1 \in P'_s$, and $A_1 \cap A_2, G, P'_s, Q_{s1} \cap Q_{s2} \models \langle \langle\!\langle T'_1, T_2 \rangle\!\rangle_{\mathsf{p}} \textbf{skip}, \sigma''_2 \rangle \rhd_{n-1} (Q_1 * Q_2)$. From (c), (k), and (j), by Def. 8.1 (item 4.d), we know there exists P_{s1}', where (l) $P_{s1} \times P_{s1}' \subseteq G \cap A_2$ and (m) $P_{s1}' \subseteq P_{s1}' \circ A_1$, and a pair of states, σ'''_1 and σ'''_2, such that (n) $\sigma'_1 = \sigma'''_1 \uplus \sigma'''_2$, (o) $\sigma'''_1 \in P_{s1}'$, and (p) $A_1, G \cap A_2, P'_{s1}, Q_{s1} \models \langle T'_1, \sigma'''_2 \rangle \rhd_{n-1} Q_1$. From (d), using Lemma 8.2, we know that (q) $A_2, G \cap A_1, P_{s2}, Q_{s2} \models \langle T_2, \sigma_2 \rangle \rhd_{n-1} Q_2$. From (l), we have (r) $(P_{s1} \cap P_{s2}) \times (P_{s1}' \cap P_{s2}) \subseteq G$. From (b), and (m), we have (s) $(P_{s1}' \cap P_{s2}) \subseteq (P_{s1}' \cap P_{s2}) \circ (A_1 \cap A_2)$. From (k), (o), (l), and (b), we have (t) $\sigma'''_1 \in P_{s2}$. From (m), (b), (p), and (q), by the induction hypothesis, we have (u) $A_1 \cap A_2, G, P_{s1}' \cap P_{s2}, Q_{s1} \cap Q_{s2} \models \langle \langle\!\langle T'_1, T_2 \rangle\!\rangle_{\mathsf{p}} \textbf{skip}, \sigma'''_2 \uplus \sigma_2 \rangle \rhd_{n-1} (Q_1 * Q_2)$. Instantiating the goal with $P_{s1}' \cap P_{s2}$, σ'''_1, and $\sigma'''_2 \uplus \sigma_2$, from (r), (s), (n), (o), (t), and (u), we conclude

  – or, $T' = \langle\!\langle T_1, T'_2 \rangle\!\rangle_{\mathsf{p}} \textbf{skip}$ and $\langle T_2, \sigma_1 \uplus \sigma_2 \rangle \longmapsto \langle T'_2, \sigma' \rangle$. The proof is symmetric to the previous case

- $\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathsf{p}} \textbf{skip} \neq \textbf{skip}$ □

An atomic block can be introduced using the following lemma.

Lemma 8.9. If $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T, \sigma \rangle \rhd_n (Q_s * Q)$, $P_s \times Q_s \subseteq G$, and $Q_s \subseteq Q_s \circ A$, then $A, G, P_s, Q_s \models \langle \langle\!\langle T \rangle\!\rangle_{\mathsf{a}} \textbf{skip}, \sigma \rangle \rhd_n Q$

Proof. By induction over n. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, assuming (a) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T, \sigma \rangle \rhd_n (Q_s * Q)$, (b) $P_s \times Q_s \subseteq G$, and (c) $Q_s \subseteq Q_s \circ A$, we have 5 cases:

- By Def. 3.1, we know that (d) $\langle\!\langle T \rangle\!\rangle_{\mathsf{a}} \textbf{skip}$ is 1-atomic

- From the semantics, if we assume $\langle \langle\!\langle T \rangle\!\rangle_{\mathsf{a}} \textbf{skip}, \sigma \rangle \longmapsto \mathsf{abort}$, then we know that (e) $\langle T, \sigma \rangle \longmapsto \mathsf{abort}$. From (a), by Def. 8.1 (item 2), we know that (e) is false

- From the semantics, if we assume $\langle\langle\!\langle T \rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma\rangle \longmapsto \mathsf{race}$, then we know that (e) $\langle T, \sigma\rangle \longmapsto \mathsf{race}$. From (a), by Def. 8.1 (item 3), we know that (e) is false
- If (e) $\langle\langle\!\langle T \rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma\rangle \longmapsto \langle T', \sigma'\rangle$, given (d) and from the semantics, we have 2 cases:
  - We have $T' = \langle\!\langle T'' \rangle\!\rangle_{\mathsf{a}}\mathbf{skip}$, which is 1-atomic, (f) $\langle T, \sigma\rangle \longmapsto \langle T'', \sigma'\rangle$, and we need to show that $A, G, P_s, Q_s \models \langle\langle\!\langle T'' \rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma'\rangle \rhd_{n-1} Q$. From (a) and (f), by Def. 8.1 (items 4.a through 4.b), we know that (g) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T'', \sigma'\rangle \rhd_{n-1} (Q_s * Q)$. From (g), by the induction hypothesis, we conclude
  - We have (f) $T = \mathbf{skip}$, $T' = \mathbf{skip}$ which is 0-atomic, $\sigma = \sigma'$, and we need to show that for all σ'', such that (g) $\sigma'' \in P_s$, there exists P_s', where $P_s \times P_s' \subseteq G$ and $P_s' \subseteq P_s' \circ A$, and a pair of states, σ_1 and σ_2, such that $\sigma = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in P_s'$, and $A, G, P_s', Q_s \models \langle \mathbf{skip}, \sigma_2\rangle \rhd_{n-1} Q$. From (a) and (f), by Def. 8.1 (item 5), we know that (h) $\sigma \in Q_s * Q$. From (h), we know that there exists σ_1' and σ_2' such that (i) $\sigma = \sigma_1' \uplus \sigma_2'$, (j) $\sigma_1' \in Q_s$, and (k) $\sigma_2' \in Q$. From (k), using Lemma 8.4, we know (l) $A, G, Q_s, Q_s \models \langle \mathbf{skip}, \sigma_2'\rangle \rhd_{n-1} Q$. Instantiating the goal with Q_s, σ_1', and σ_2', from (b), (c), (i), (j), and (l), we conclude
- We know that $\langle\!\langle T \rangle\!\rangle_{\mathsf{a}}\mathbf{skip} \neq \mathbf{skip}$ □

The following lemma is used for framing a sixtuple into a larger shared memory.

Lemma 8.10. If $A, G, P_s, Q_s \models \langle T, \sigma\rangle \rhd_n Q$, and either I is precise or $\mathsf{img}(G)$ and $\mathsf{dom}(A)$ coincide, then

1. If T is 0-atomic, then $I^2 * A, I^2 * G, I * P_s, I * Q_s \models \langle T, \sigma\rangle \rhd_n Q$
2. If T is 1-atomic, and $\sigma' \in I$, then $I^2 * A, I^2 * G, I * P_s, I * Q_s \models \langle T, \sigma' \uplus \sigma\rangle \rhd_n Q$

Proof. By induction over n. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, assuming (a) $A, G, P_s, Q_s \models \langle T, \sigma\rangle \rhd_n Q$, and (b) I is precise or $\mathsf{img}(G)$ and $\mathsf{dom}(A)$ coincide, we have 2 cases:

- If (c) T is 0-atomic, we need to show $I^2 * A, I^2 * G, I * P_s, I * Q_s \models \langle T, \sigma\rangle \rhd_n Q$. By Def. 8.1, we have 5 cases:

  - From (c), we know that T is 0-atomic
  - From (a), by Def. 8.1 (item 2), we know that $\langle T, \sigma\rangle \longmapsto \textsf{abort}$ is false
  - From (a), by Def. 8.1 (item 3), we know that $\langle T, \sigma\rangle \longmapsto \textsf{race}$ is false
  - If (d) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'\rangle$, given (c) the we have 2 cases:
    * If T' is 0-atomic, we need to show that $\textsf{dom}(\sigma) = \textsf{dom}(\sigma')$ and $I^2 * A, I^2 * G, I * P_s, I * Q_s \models \langle T', \sigma'\rangle \rhd_{n-1} Q$. From (a) and (d), by Def. 8.1 (item 4.b), we know that (e) $\textsf{dom}(\sigma) = \textsf{dom}(\sigma')$ and (f) $A, G, P_s, Q_s \models \langle T', \sigma'\rangle \rhd_{n-1} Q$. From (f) and (b), by the induction hypothesis (item 1), we know that (g) $I^2 * A, I^2 * G, I * P_s, I * Q_s \models \langle T', \sigma'\rangle \rhd_{n-1} Q$. From (e) and (g) we conclude
    * If T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (e) $\sigma_2 = \sigma_1 \uplus \sigma'$ and (f) $\sigma_1 \in I * P_s$, we have $I^2 * A, I^2 * G, I * P_s, I * Q_s \models \langle T', \sigma_2\rangle \rhd_{n-1} Q$. From (f), we know exists σ_1' and σ_1'', such that $\sigma_1 = \sigma_1' \uplus \sigma_1''$, (g) $\sigma_1' \in I$, and (h) $\sigma_1'' \in P_s$. From (a), (d), and (h), by Def. 8.1 (item 4.c), we know that (i) $A, G, P_s, Q_s \models \langle T', \sigma_1'' \uplus \sigma'\rangle \rhd_{n-1} Q$. From (i), (b), and (g), by the induction hypothesis (item 2), we conclude
  - We assume (d) $T = \textbf{skip}$. From (a) and (d), by Def. 8.1 (item 5), we know (e) $P_s \subseteq Q_s$ and (f) $\sigma \in Q$. From (e), we have (g) $I * P_s \subseteq I * Q_s$. From (g), and (f), we conclude

- If (c) T is 1-atomic, and (d) $\sigma' \in I'$, we need to show $I^2 * A, I^2 * G, I * P_s, I * Q_s \models \langle T, \sigma' \uplus \sigma\rangle \rhd_n Q$. By Def. 8.1, we have 5 cases:
  - From (c), we know that T is 1-atomic
  - From (a), by Def. 8.1 (item 2), we know that (e) $\langle T, \sigma\rangle \longmapsto \textsf{abort}$ is false. From (e), using Lemma 3.22 (item 1), we conclude
  - From (a), by Def. 8.1 (item 3), we know that (f) $\langle T, \sigma\rangle \longmapsto \textsf{race}$ is false. From (e) and (f), using Lemma 3.22 (item 2), we conclude
  - We know (g) $\langle T, \sigma' \uplus \sigma\rangle \longmapsto \langle T', \sigma''\rangle$. From (e) and (g), using Lemma 3.22 (item 3), we know exists σ''' such that (h) $\sigma'' = \sigma' \uplus \sigma'''$ and (i) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'''\rangle$.

Then we have 2 cases:

  * If T' is 1-atomic, we need to show that $I^2{*}A, I^2{*}G, I{*}P_s, I{*}Q_s \models \langle T', \sigma' \uplus \sigma''' \rangle \rhd_{n-1} Q$. From (a) and (i), by Def. 8.1 (item 4.a), we know that (j) $A, G, P_s, Q_s \models \langle T', \sigma''' \rangle \rhd_{n-1} Q$. From (j), (b), and (d), by the induction hypothesis (item 2), we conclude
  * If T' is 0-atomic, we need to show that for all σ'''', such that (j) $\sigma'''' \in I{*}P_s$, there exists P'_s, where $(I{*}P_s) \times P'_s \subseteq (I^2{*}G)$ and $P'_s \subseteq P'_s \circ (I^2{*}A)$, and a pair of states, σ_1 and σ_2, such that $\sigma'' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in P'_s$, and $I^2{*}A, I^2{*}G, P'_s, I{*}Q_s \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q$. From (j), we know there exists σ'_1 and σ'_2, such that $\sigma'''' = \sigma'_1 \uplus \sigma'_2$, (k) $\sigma'_1 \in I$, and (l) $\sigma'_2 \in P_s$. From (a), (i) and (l), by Def. 8.1 (item 4.d), we know there exists P''_s, where (m) $P_s \times P''_s \subseteq G$ and (n) $P''_s \subseteq P''_s \circ A$, and a pair of states, σ''_1 and σ''_2, such that (o) $\sigma''' = \sigma''_1 \uplus \sigma''_2$, (p) $\sigma''_1 \in P''_s$, and (q) $A, G, P''_s, Q_s \models \langle T', \sigma''_2 \rangle \rhd_{n-1} Q$. From (q) and (b), by the induction hypothesis (item 1), we have (r) $I^2{*}A, I^2{*}G, I{*}P''_s, I{*}Q_s \models \langle T', \sigma''_2 \rangle \rhd_{n-1} Q$. From (m), we have (s) $(I{*}P_s) \times (I{*}P''_s) \subseteq (I^2{*}G)$. From (b), (l), and (m), we have (t) $(I{*}P''_s) \subseteq (I{*}P''_s) \circ (I^2{*}A)$. From (d), (o), and (p), we have (u) $\sigma' \uplus \sigma''_1 \in I{*}P_s$. Instantiating the goal with $I{*}P''_s$, $\sigma' \uplus \sigma''_1$, and σ''_2, from (s), (t), (u), and (r), we conclude

- From (c), we know that $T \neq$ **skip** ☐

The following lemma is used to transfer a resource from shared to private in a sixtuple.

Lemma 8.11. If $A, G, P_s, Q_s \models \langle T, \sigma \rangle \rhd_n Q$, then

1. If T is 0-atomic, and $\sigma' \in P_s$, then $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T, \sigma' \uplus \sigma \rangle \rhd_n (Q_s{*}Q)$
2. If T is 1-atomic, and $P_s \neq \varnothing$, then $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T, \sigma \rangle \rhd_n (Q_s{*}Q)$

Proof. By induction over n. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, assuming (a) $A, G, P_s, Q_s \models \langle T, \sigma \rangle \rhd_n Q$, we have 2 cases:

- If (b) T is 0-atomic, and (c) $\sigma' \in P_s$, we need to show $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T, \sigma' \uplus \sigma \rangle \rhd_n (Q_s{*}Q)$. By Def. 8.1, we have 5 cases:

- From (b), we know that T is 0-atomic
- From (a), by Def. 8.1 (item 2), we know that (d) $\langle T, \sigma\rangle \longmapsto \mathsf{abort}$ is false. From (d), using Lemma 3.22 (item 1), we conclude
- From (a), by Def. 8.1 (item 3), we know that (e) $\langle T, \sigma\rangle \longmapsto \mathsf{race}$ is false. From (d) and (e), using Lemma 3.22 (item 2), we conclude
- We know (f) $\langle T, \sigma' \uplus \sigma\rangle \longmapsto \langle T', \sigma''\rangle$. From (d) and (f), using Lemma 3.22 (item 3), we know exists σ''' such that (g) $\sigma'' = \sigma' \uplus \sigma'''$ and (h) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'''\rangle$. Then we have 2 cases:
    - If T' is 0-atomic, we need to show that $\mathsf{dom}(\sigma' \uplus \sigma) = \mathsf{dom}(\sigma' \uplus \sigma''')$ and $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T', \sigma' \uplus \sigma'''\rangle \rhd_{n-1} (Q_s{*}Q)$. From (a) and (h), by Def. 8.1 (item 4.b), we know that (i) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma''')$ and (j) $A, G, P_s, Q_s \models \langle T', \sigma'''\rangle \rhd_{n-1} Q$. From (j) and (c), by the induction hypothesis (item 1), we know (k) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T', \sigma' \uplus \sigma'''\rangle \rhd_{n-1} (Q_s{*}Q)$. From (i) and (k), we conclude
    - If T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (i) $\sigma_2 = \sigma_1 \uplus \sigma' \uplus \sigma'''$ and (j) $\sigma_1 \in \mathsf{Emp}$, we have $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T', \sigma_2\rangle \rhd_{n-1} (Q_s{*}Q)$. From (i), and (j), we have (k) $\sigma_2 = \sigma'$. From (a), (h), (c), and (k), by Def. 8.1 (item 4.c), we know that (l) $A, G, P_s, Q_s \models \langle T', \sigma''\rangle \rhd_{n-1} Q$. From (l) and (c), by the induction hypothesis (item 2), we conclude
- We assume (d) $T = \textbf{skip}$. From (a) and (d), by Def. 8.1 (item 5), we know that (e) $P_s \subseteq Q_s$ and (f) $\sigma \in Q$. From (c), (e), and (f) we know that $\sigma' \uplus \sigma \in Q_s{*}Q$ and conclude

- If (b) T is 1-atomic, and (c) $P_s \neq \emptyset$, we need to show $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T, \sigma\rangle \rhd_n (Q_s{*}Q)$. By Def. 8.1, we have 5 cases:
    - From (b), we know that T is 1-atomic
    - From (a), by Def. 8.1 (item 2), we know that $\langle T, \sigma\rangle \longmapsto \mathsf{abort}$ is false
    - From (a), by Def. 8.1 (item 3), we know that $\langle T, \sigma\rangle \longmapsto \mathsf{race}$ is false

- If (d) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'\rangle$, given (b) the we have 2 cases:
    - If T' is 1-atomic, we need to show that $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T', \sigma'\rangle \rhd_{n-1} (Q_s{*}Q)$. From (a) and (d), by Def. 8.1 (item 4.a), we know that (d) $A, G, P_s, Q_s \models \langle T', \sigma'\rangle \rhd_{n-1} Q$. From (d) and (c), by the induction hypothesis (item 2), we conclude
    - If T' is 0-atomic, we need to show that for all σ'', such that (e) $\sigma'' \in \mathsf{Emp}$, exists P'_s, where $\mathsf{Emp} \times P'_s \subseteq \mathsf{Emp}^2$ and $P'_s \subseteq P'_s \circ \mathsf{Emp}^2$, and a pair of states, σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in P'_s$, and $\mathsf{Emp}^2, \mathsf{Emp}^2, P'_s, \mathsf{Emp} \models \langle T', \sigma_2\rangle \rhd_{n-1} (Q_s{*}Q)$. From (a), (d), and (c), by Def. 8.1 (item 4.d), we know there exists P''_s, where (f) $P_s \times P''_s \subseteq G$ and (g) $P''_s \subseteq P''_s \circ A$, and a pair of states, σ'_1 and σ'_2, such that (h) $\sigma' = \sigma'_1 \uplus \sigma'_2$, (i) $\sigma'_1 \in P''_s$, and (j) $A, G, P''_s, Q_s \models \langle T', \sigma'_2\rangle \rhd_{n-1} Q$. From (j) and (i), by the induction hypothesis (item 1), we have (j) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T', \sigma'\rangle \rhd_{n-1} (Q_s{*}Q)$. Instantiating the goal with Emp, $\varnothing$, and σ', from (j), and knowing that trivially $\mathsf{Emp} \times \mathsf{Emp} \subseteq \mathsf{Emp}^2$, $\mathsf{Emp} \subseteq \mathsf{Emp} \circ \mathsf{Emp}^2$, $\sigma' = \varnothing \uplus \sigma'$, and $\varnothing \in \mathsf{Emp}$ hold, we conclude
- From (b), we know that $T \neq$ **skip** □

The following lemma is used for the conjunction of triples.

Lemma 8.12. If $A, G, P_s, Q_s \models \langle T, \sigma\rangle \rhd_n Q_1$, $A, G, P_s, Q_s \models \langle T, \sigma\rangle \rhd_n Q_2$, and $\mathsf{img}(G)$ is precise, then $A, G, P_s, Q_s \models \langle T, \sigma\rangle \rhd_n (Q_1 \cap Q_2)$

Proof. By induction over n. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, assuming (a) $A, G, P_s, Q_s \models \langle T, \sigma\rangle \rhd_n Q_1$, (b) $A, G, P_s, Q_s \models \langle T, \sigma\rangle \rhd_n Q_2$, and (c) $\mathsf{img}(G)$ is precise, we have 5 cases:

- From (a), by Def. 8.1 (item 1), we know that T is 0- or 1-atomic
- From (a), by Def. 8.1 (item 2), we know that $\langle T, \sigma\rangle \longmapsto \mathsf{abort}$ is false
- From (a), by Def. 8.1 (item 3), we know that $\langle T, \sigma\rangle \longmapsto \mathsf{race}$ is false
- If (d) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'\rangle$, then we have 4 cases:

– If both T and T' are 1-atomic, we need to show that $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (d), by Def. 8.1 (item 4.a), we know that (e) $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_{n-1} Q_1$. From (b) and (d), by Def. 8.1 (item 4.a), we know that (f) $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_{n-1} Q_2$. From (e), (f), and (c), by the induction hypothesis, we conclude

– If both T and T' are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (d), by Def. 8.1 (item 4.b), we know already that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, and also that (e) $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_{n-1} Q_1$. From (b) and (d), by Def. 8.1 (item 4.b), we also know that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and (f) $A, G, P_s, Q_s \models \langle T', \sigma' \rangle \rhd_{n-1} Q_2$. From (e), (f), and (c), by the induction hypothesis, we conclude

– If T is 0-atomic and T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (e) $\sigma_2 = \sigma_1 \uplus \sigma'$ and (f) $\sigma_1 \in P_s$, we have $A, G, P_s, Q_s \models \langle T', \sigma_2 \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a), (d), (e), and (f), by Def. 8.1 (item 4.c), we know that (g) $A, G, P_s, Q_s \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q_1$. From (b), (d), (e), and (f), by Def. 8.1 (item 4.c), we know that (h) $A, G, P_s, Q_s \models \langle T', \sigma_2 \rangle \rhd_{n-1} Q_2$. From (g), (h), and (c), by the induction hypothesis, we conclude

– If T is 1-atomic and T' is 0-atomic, we need to show that for all σ'', such that (e) $\sigma'' \in P_s$, there exists P_s', where $P_s \times P_s' \subseteq G$ and $P_s' \subseteq P_s' \circ A$, and a pair of states, σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in P_s'$, and $A, G, P_s', Q_s \models \langle T', \sigma_2 \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a), (d), and (e), by Def. 8.1 (item 4.d), we know there exists ${P_s}_1'$, where (f) $P_s \times {P_s}_1' \subseteq G$ and (g) ${P_s}_1' \subseteq {P_s}_1' \circ A$, and a pair of states, σ_1' and σ_2', such that (h) $\sigma' = \sigma_1' \uplus \sigma_2'$, (i) $\sigma_1' \in {P_s}_1'$, and (j) $A, G, {P_s}_1', Q_s \models \langle T', \sigma_2' \rangle \rhd_{n-1} Q_1$. From (b), (d), and (e), by Def. 8.1 (item 4.d), we know there exists ${P_s}_2'$, where (k) $P_s \times {P_s}_2' \subseteq G$ and (l) ${P_s}_2' \subseteq {P_s}_2' \circ A$, and a pair of states, σ_1'' and σ_2'', such that (m) $\sigma' = \sigma_1'' \uplus \sigma_2''$, (n) $\sigma_1'' \in {P_s}_2'$, and (o) $A, G, {P_s}_2', Q_s \models \langle T', \sigma_2'' \rangle \rhd_{n-1} Q_2$. From (j), using Lemma 8.3, we have (p) $A, G, {P_s}_1' \cap {P_s}_2', Q_s \models \langle T', \sigma_2' \rangle \rhd_{n-1} Q_1$. From (o), using Lemma 8.3, we have (q) $A, G, {P_s}_1' \cap {P_s}_2', Q_s \models \langle T', \sigma_2'' \rangle \rhd_{n-1} Q_2$. From (f), we have (r) $P_s \times ({P_s}_1' \cap {P_s}_2') \subseteq G$. From (g) and (l), we have (s) $({P_s}_1' \cap {P_s}_2') \subseteq$

$(P_{s1}' \cap P_{s2}') \circ A$. From (e), (i), (n), (f), (k), (h), (m), and (c), we know that $\sigma_1' = \sigma_1''$ and $\sigma_2' = \sigma_2''$. From (p), (q), and (c), by the induction hypothesis, we have (t) $A, G, P_{s1}' \cap P_{s2}', Q_s \models \langle T', \sigma_2' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. Instantiating the goal with $P_{s1}' \cap P_{s2}'$, σ_1', and σ_2', from (r), (s), (h), (i), (n), and (t), we conclude

- We assume (d) $T = \textbf{skip}$. From (a) and (d), by Def. 8.1 (item 5), we know that (e) $P_s \subseteq Q_s$ and (f) $\sigma \in Q_1$. From (b) and (d), by Def. 8.1 (item 5), we know that (g) $P_s \subseteq Q_s$ and (h) $\sigma \in Q_2$. From (f) and (h), we know that (i) $\sigma \in Q_1 \cap Q_2$. From (e) and (i), we conclude $\square$

Semantics rules. The septuple $A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}$ is the semantic correspondent to a SAGL judgement $A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\}$. It is defined in terms of $A, G, P_s, Q_s \models \langle T, \sigma \rangle \rhd Q$ as show below:

Definition 8.13. $A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}$, if and only if, for all σ, such that $\sigma \in P$, we have $A, G, P_s, Q_s \models \langle c, \sigma \rangle \rhd Q$

From Def. 8.13, we can prove Lemma 8.14 which states more explicitly the properties guaranteed by the semantic septuple: safety, race-freedom, and partial correctness (items 1, 2, and 3, respectively).

Lemma 8.14. If $A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}$, then for all σ, such that $\sigma \in P_s * P$, we have:

1. $\neg \langle c, \sigma \rangle \longmapsto^* \mathsf{abort}$
2. $\neg \langle c, \sigma \rangle \longmapsto^* \mathsf{race}$
3. If $\langle c, \sigma \rangle \longmapsto^* \langle \textbf{skip}, \sigma' \rangle$, then $\sigma' \in Q_s * Q$

Proof. From $A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}$, using Lemma 8.26[1], we obtain (a) $\mathsf{Emp}^2, \mathsf{Emp}^2 \models \{\mathsf{Emp}, P_s * P\}\, c\, \{\mathsf{Emp}, Q_s * Q\}$. Given (a), and $\sigma \in P_s * P$, from Def. 8.13, we obtain (b) $Emp^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle c, \sigma \rangle \rhd (Q_s * Q)$. We then generalize the proof from command c to any 0- or 1-atomic thread tree T. Now we can consider each one of the goals:

[1]Although Lemma 8.26 is defined later on the text, there is no circularity.

- For goal 1, we need to show that for all n, (c) $\langle T, \sigma\rangle \longmapsto^{n}$ abort is false. From (b), we obtain (d) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T, \sigma\rangle \rhd_n (Q_s{*}Q)$, which is our sole assumption. By induction over n, we have two cases. The base case, where $n = 0$, is trivial as $\langle T, \sigma\rangle \neq \mathsf{abort}$. In the inductive case, where $n > 0$, we know there exists a configuration κ, such that (e) $\langle T, \sigma\rangle \longmapsto \kappa$ and (f) $\kappa \longmapsto^{n-1}$ abort. From (d), given that $n > 0$, we know, from items 2 and 3 of Def. 8.1, that by exclusion there must exists T' and σ', such that $\kappa = \langle T', \sigma'\rangle$; therefore we obtain (g) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'\rangle$. From (g), using Remark 3.17, we know that T' is either 0- or 1-atomic. Given (g), and (d), from items 4.a through 4.d of Def. 8.1, we know that (h) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T', \sigma'\rangle \rhd_{n-1} (Q_s{*}Q)$. From (h), and (f), using the induction hypothesis, we know that (i) $\langle T', \sigma'\rangle \longmapsto^{n-1}$ abort is false. From (g), and (i), we know $\langle T, \sigma\rangle \longmapsto^{n-1}$ abort is false, which was our goal.

- For goal 2, we need to show that for all n, (c) $\langle T, \sigma\rangle \longmapsto^{n}$ race is false. From (b), we obtain (d) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T, \sigma\rangle \rhd_n (Q_s * Q)$, which is our sole assumption. By induction over n, we have two cases. The base case, where $n = 0$, is trivial as $\langle T, \sigma\rangle \neq \mathsf{race}$. In the inductive case, where $n > 0$, we know there exists a configuration κ, such that (e) $\langle T, \sigma\rangle \longmapsto \kappa$ and (f) $\kappa \longmapsto^{n-1}$ race. From (d), given that $n > 0$, we know, from items 2 and 3 of Def. 8.1, that by exclusion there must exists T' and σ', such that $\kappa = \langle T', \sigma'\rangle$; therefore we obtain (g) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'\rangle$. From (g), using Remark 3.17, we know that T' is either 0- or 1-atomic. Given (g), and (d), from items 4.a through 4.d of Def. 8.1, we know that (h) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T', \sigma'\rangle \rhd_{n-1} (Q_s{*}Q)$. From (h), and (f), using the induction hypothesis, we know that (i) $\langle T', \sigma'\rangle \longmapsto^{n-1}$ race is false. From (g), and (i), we know $\langle T, \sigma\rangle \longmapsto^{n-1}$ race is false, which was our goal.

- For goal 3, we need to show that for all n, if (c) $\langle T, \sigma\rangle \longmapsto^{n} \langle \mathbf{skip}, \sigma'\rangle$, then $\sigma' \in Q_s{*}Q$. From (b), we obtain (d) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T, \sigma\rangle \rhd_n (Q_s{*}Q)$, which is our sole assumption. By induction over n, we have two cases. In base case, where $n = 0$, we know that $T = \mathbf{skip}$ and $\sigma' = \sigma$; given (d), from item 5 of Def. 8.1, we obtain the goal

$\sigma' \in Q_s * Q$. In the inductive case, where $n > 0$, we know there exists a configuration κ, such that (e) $\langle T, \sigma\rangle \longmapsto \kappa$ and (f) $\kappa \longmapsto^{n-1} \langle \mathbf{skip}, \sigma'\rangle$. From (d), given that $n > 0$, we know, from items 2 and 3 of Def. 8.1, that by exclusion there must exists T' and σ'', such that $\kappa = \langle T', \sigma''\rangle$; therefore we obtain (g) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma''\rangle$. From (g), using Remark 3.17, we know that T' is either 0- or 1-atomic. Given (g), and (d), from items 4.a through 4.d of Def. 8.1, we know that (h) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle T', \sigma''\rangle \rhd_{n-1} (Q_s * Q)$. From (h), and (f), using the induction hypothesis, we obtain the goal $\sigma' \in Q_s * Q$. □

In the following sequence of lemmas, Lemma 8.15 through Lemma 8.28, we show the correspondence between the SAGL rules from Fig. 8.3, and their semantic equivalent using definition Def. 8.13.

Lemma 8.15.

$$\frac{}{\mathsf{Emp}^2, \mathsf{Emp}^2 \models \{\mathsf{Emp}, Q \circ [\![\nu := e]\!]\}\ \nu := e\ \{\mathsf{Emp}, Q\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in Q \circ [\![\nu := e]\!]$, and we need to show that $\mathsf{Emp}^2, \mathsf{Emp}^2\mathsf{Emp}, \mathsf{Emp} \models \langle \nu := e, \sigma\rangle \rhd_n Q$. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, we have 5 cases:

- By Def. 3.1, we know that (b) $\nu := e$ is 0-atomic
- From the semantics, we know that $\langle \nu := e, \sigma\rangle \longmapsto \mathsf{abort}$ is false if $\langle \nu := e, \sigma\rangle \longrightarrow \mathsf{abort}$ is false. From the sequential semantics, we know $\langle \nu := e, \sigma\rangle \longrightarrow \mathsf{abort}$ is false if there exists σ' such that $(\sigma, \sigma') \in [\![a]\!]$. From (a), by Def. 6.2 (item 1), we conclude
- From the semantics, we know that $\langle \nu := e, \sigma\rangle \longmapsto \mathsf{race}$ is false
- From the semantics, we know that $\langle \nu := e, \sigma\rangle \longmapsto \langle \mathbf{skip}, \sigma'\rangle$, where (c) $\langle \nu := e, \sigma\rangle \longrightarrow \langle \mathbf{skip}, \sigma'\rangle$. From (b), and since $\mathbf{skip}$ is 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle \mathbf{skip}, \sigma'\rangle \rhd_{n-1} Q$. From the sequential semantics, and (c), we know (d) $(\sigma, \sigma') \in [\![\nu := e]\!]$. From (d), using Remark 3.13, we know (e) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$. From (a) and (d), by Def. 6.2 (item 2), we know that (f) $\sigma' \in Q$.

From (f), and Lemma 8.4, we know (g) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle \textbf{skip}, \sigma' \rangle \rhd_{n-1} Q$. From (e) and (g) we conclude

- We know that $(\nu := e) \neq \textbf{skip}$ $\square$

Lemma 8.16.

$$\frac{}{\mathsf{Emp}^2, \mathsf{Emp}^2 \models \{\mathsf{Emp}, Q \circ [\![a]\!]\} \, \langle a \rangle \, \{\mathsf{Emp}, Q\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in Q \circ [\![a]\!]$, and we need to show that $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle \textbf{atomic}\ a, \sigma \rangle \rhd_n Q$. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, we have 5 cases:

- By Def. 3.1, we know that (b) **atomic** a is 0-atomic
- From the semantics, we know that $\langle \textbf{atomic}\ a, \sigma \rangle \longmapsto \mathsf{abort}$ is false
- From the semantics, we know that $\langle \textbf{atomic}\ a, \sigma \rangle \longmapsto \mathsf{race}$ is false
- From the semantics, we know that $\langle \textbf{atomic}\ a, \sigma \rangle \longmapsto \langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \textbf{skip}, \sigma \rangle$, given that $\bullet$ is 0-atomic. From (b), and since $\langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \textbf{skip}$ is 1-atomic, we need to show that for all σ_1 and σ_2, such that (c) $\sigma_2 = \sigma_1 \uplus \sigma$ and (d) $\sigma_1 \in \mathsf{Emp}$, $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \textbf{skip}, \sigma_2 \rangle \rhd_{n-1} Q$. From (d), we know $\sigma_1 = \emptyset$, therefore, from (c), we know $\sigma_2 = \sigma$. If $n = 1$, by Def. 8.1, we conclude. If $n > 1$, by Def. 8.1, we have 5 cases:
  - By Def. 3.1, we know that (e) $\langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \textbf{skip}$ is 1-atomic
  - From the semantics, we know that $\langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \textbf{skip}, \sigma \rangle \longmapsto \mathsf{abort}$ is false if $\langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \textbf{skip}, \sigma \rangle \longrightarrow \mathsf{abort}$ is false. From the sequential semantics, we know $\langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \textbf{skip}, \sigma \rangle \longrightarrow \mathsf{abort}$ is false if there exists σ' such that $(\sigma, \sigma') \in [\![a]\!]$. From (a), by Def. 6.2 (item 1), we conclude
  - From the semantics, we know that $\langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \textbf{skip}, \sigma \rangle \longmapsto \mathsf{race}$ is false
  - From the semantics, we know that $\langle \langle\!\langle a \rangle\!\rangle_{\mathsf{a}} \textbf{skip}, \sigma \rangle \longmapsto \langle \langle\!\langle \textbf{skip} \rangle\!\rangle_{\mathsf{a}} \textbf{skip}, \sigma' \rangle$, where (f) $\langle a, \sigma \rangle \longrightarrow \langle \textbf{skip}, \sigma' \rangle$. From (e), and since $\langle\!\langle \textbf{skip} \rangle\!\rangle_{\mathsf{a}} \textbf{skip}$ is 1-atomic, we need to show that $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle \langle\!\langle \textbf{skip} \rangle\!\rangle_{\mathsf{a}} \textbf{skip}, \sigma' \rangle \rhd_{(n-2)} Q$. If $n = 2$, by Def. 8.1, we conclude. If $n > 2$, by Def. 8.1, we have 5 cases:

    * By Def. 3.1, we know that (g) $\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}$ is 1-atomic
    * From the semantics, we know that $\langle\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma'\rangle \longmapsto \mathsf{abort}$ is false
    * From the semantics, we know that $\langle\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma'\rangle \longmapsto \mathsf{race}$ is false
    * From the semantics, we know that $\langle\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip}, \sigma'\rangle \longmapsto \langle\mathbf{skip}, \sigma'\rangle$. From (g), and since **skip** is 0-atomic, we need to show that for all σ'', such that $\sigma'' \in \mathsf{Emp}$, there exists P'_s, where $\mathsf{Emp} \times P'_s \subseteq \mathsf{Emp}^2$ and $P'_s \subseteq P'_s \circ \mathsf{Emp}^2$, and a pair of states, σ'_1 and σ'_2, such that $\sigma' = \sigma'_1 \uplus \sigma'_2$, $\sigma'_1 \in P'_s$, and $\mathsf{Emp}^2, \mathsf{Emp}^2, P'_s, \mathsf{Emp} \models \langle\mathbf{skip}, \sigma'_2\rangle \rhd_{(n-3)} Q$.
    We instantiate P'_s as Emp, σ'_1 as $\emptyset$, and σ'_2 as σ', as we know that $\mathsf{Emp} \times \mathsf{Emp} \subseteq \mathsf{Emp}^2$, $\mathsf{Emp} \subseteq \mathsf{Emp} \circ \mathsf{Emp}^2$, $\sigma' = \emptyset \uplus \sigma'$, and $\emptyset \in \mathsf{Emp}$; it remains to show that $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle\mathbf{skip}, \sigma'\rangle \rhd_{(n-3)} Q$. From the sequential semantics, and (f), we know (h) $(\sigma, \sigma') \in [\![a]\!]$. From (a) and (h), by Def. 6.2 (item 2), we know that (i) $\sigma' \in Q$. From (i), and Lemma 8.4, we conclude
    * We know that $\langle\!\langle\mathbf{skip}\rangle\!\rangle_{\mathsf{a}}\mathbf{skip} \neq \mathbf{skip}$
  - We know that $\langle\!\langle a\rangle\!\rangle_{\mathsf{a}}\mathbf{skip} \neq \mathbf{skip}$
- We know that **atomic** $a \neq$ **skip** □

Lemma 8.17.

$$\frac{A, G \models \{P_s, P\}\, c_1\, \{P'_s, P'\} \quad A, G \models \{P'_s, P'\}\, c_2\, \{Q_s, Q\}}{A, G \models \{P_s, P\}\, c_1; c_2\, \{Q_s, Q\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $A, G, P_s, Q_s \models \langle c_1; c_2, \sigma\rangle \rhd_n Q$. From (a), and $A, G \models \{P_s, P\}\, c_1\, \{P'_s, P'\}$, by Def. 8.13, we know that (b) $A, G, P_s, P'_s \models \langle c_1, \sigma\rangle \rhd_n P'$. From $A, G \models \{P'_s, P'\}\, c_2\, \{Q_s, Q\}$, by Def. 8.13, we know that (c) for all $\sigma' \in P'$, we have $A, G, P'_s, Q_s \models \langle c_2, \sigma'\rangle \rhd_n Q$. From (b) and (c), using Lemma 8.5 (item 1), we conclude □

Lemma 8.18.

$$\frac{}{\mathsf{Emp}^2, \mathsf{Emp}^2 \models \{\mathsf{Emp}, P\}\, \mathbf{skip}\, \{\mathsf{Emp}, P\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle \textbf{skip}, \sigma \rangle \rhd_n P$. From (a), using Lemma 8.4, we conclude □

Lemma 8.19.

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad A, G \models \{P_s, P \cap \lfloor b \rfloor\}\, c_1\, \{Q_s, Q\} \quad A, G \models \{P_s, P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q_s, Q\}}{A, G \models \{P_s, P\}\, \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\, \{Q_s, Q\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $A, G, P_s, Q_s \models \langle \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2, \sigma \rangle \rhd_n Q$. From (a), and $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, we know (b) $\sigma \in \lfloor b \rfloor \cup \lfloor \neg b \rfloor$. From (b) we can consider two cases:

- If (c) $\sigma \in \lfloor b \rfloor$, with (a), we know (d) $\sigma \in P \cap \lfloor b \rfloor$. From (d), and $A, G \models \{P_s, P \cap \lfloor b \rfloor\}\, c_1\, \{Q_s, Q\}$, by Def. 8.13, we know that (e) $A, G, P_s, Q_s \models \langle c_1, \sigma \rangle \rhd_n Q$. From (e) and (c), using Lemma 8.6 (item 1), we conclude

- If (c) $\sigma \in \lfloor \neg b \rfloor$, with (a), we know (d) $\sigma \in P \cap \lfloor \neg b \rfloor$. From (d), and $A, G \models \{P_s, P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q_s, Q\}$, by Def. 8.13, we know that (e) $A, G, P_s, Q_s \models \langle c_2, \sigma \rangle \rhd_n Q$. From (e) and (c), using Lemma 8.6 (item 2), we conclude □

Lemma 8.20.

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad A, G \models \{P_s, P \cap \lfloor b \rfloor\}\, c\, \{P_s, P\}}{A, G \models \{P_s, P\}\, \textbf{while}\ b\ \textbf{do}\ c\, \{P_s, P \cap \lfloor \neg b \rfloor\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $A, G, P_s, P_s \models \langle \textbf{while}\ b\ \textbf{do}\ c, \sigma \rangle \rhd_n (P \cap \lfloor \neg b \rfloor)$. From $A, G \models \{P_s, P \cap \lfloor b \rfloor\}\, c\, \{P_s, P\}$, by Def. 8.13, we know that (b) for all $\sigma' \in P \cap \lfloor b \rfloor$, we have $A, G, P_s, P_s \models \langle c, \sigma' \rangle \rhd_n P$. From (a), (b), and $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, using Lemma 8.7, we conclude □

Lemma 8.21.

$$\frac{\begin{array}{cc} P_{s1} \subseteq P_{s1} \circ A_1 & P_{s2} \subseteq P_{s2} \circ A_2 \\ A_1, G \cap A_2 \models \{P_{s1}, P_1\}\, c_1\, \{Q_{s1}, Q_1\} & A_2, G \cap A_1 \models \{P_{s2}, P_2\}\, c_2\, \{Q_{s2}, Q_2\} \end{array}}{A_1 \cap A_2, G \models \{P_{s1} \cap P_{s2}, P_1 * P_2\}\, c_1 \parallel c_2\, \{Q_{s1} \cap Q_{s2}, Q_1 * Q_2\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in P_1 * P_2$, and we need to show that $A_1 \cap A_2, G, P_{s1} \cap P_{s2}, Q_{s1} \cap Q_{s2} \models \langle c_1 \parallel c_2, \sigma\rangle \rhd_n (Q_1 * Q_2)$. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, we have 5 cases:

- By Def. 3.1, we know that (b) $c_1 \parallel c_2$ is 0-atomic
- From the semantics, we know that $\langle c_1 \parallel c_2, \sigma\rangle \longmapsto \textsf{abort}$ is false
- From the semantics, we know that $\langle c_1 \parallel c_2, \sigma\rangle \longmapsto \textsf{race}$ is false
- From the semantics, we know that $\langle c_1 \parallel c_2, \sigma\rangle \longmapsto \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}} \mathbf{skip}, \sigma\rangle$. By Def. 3.1, we know (c) $\langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}} \mathbf{skip}$ is 0-atomic. From (b) and (c), we need to show that $\sigma = \sigma$, which is trivial, and that $A_1 \cap A_2, G, P_{s1} \cap P_{s2}, Q_{s1} \cap Q_{s2} \models \langle \langle\!\langle c_1, c_2 \rangle\!\rangle_{\mathbf{p}} \mathbf{skip}, \sigma\rangle \rhd_{n-1} (Q_1 * Q_2)$. From (a), we know there exists σ_1 and σ_2, such that $\sigma = \sigma_1 \uplus \sigma_2$, (d) $\sigma_1 \in P_1$, and (e) $\sigma_2 \in P_2$. From (d), and $A_1, G \cap A_2 \models \{P_{s1}, P_1\}\, c_1 \,\{Q_{s1}, Q_1\}$, by Def. 8.13, we know that (f) $A_1, G \cap A_2, P_{s1}, Q_{s1} \models \langle c_1, \sigma_1\rangle \rhd_{n-1} Q_1$. From (e), and $A_2, G \cap A_1 \models \{P_{s2}, P_2\}\, c_2 \,\{Q_{s2}, Q_2\}$, by Def. 8.13, we know that (g) $A_2, G \cap A_1, P_{s2}, Q_{s2} \models \langle c_2, \sigma_2\rangle \rhd_{n-1} Q_2$. From $P_{s1} \subseteq P_{s1} \circ A_1$, $P_{s2} \subseteq P_{s2} \circ A_2$, (f), (g), and (c), using Lemma 8.8, we conclude
- We know that $c_1 \parallel c_2 \neq \mathbf{skip}$ □

Lemma 8.22.

$$\frac{\mathsf{Emp}^2, \mathsf{Emp}^2 \models \{\mathsf{Emp}, P_s * P\}\, c \,\{\mathsf{Emp}, Q_s * Q\} \quad P_s \times Q_s \subseteq G \quad Q_s \subseteq Q_s \circ A}{A, G \models \{P_s, P\}\, \textbf{atomic}\, c \,\{Q_s, Q\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $A, G, P_s, Q_s \models \langle \textbf{atomic}\, c, \sigma\rangle \rhd_n Q$. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, we have 5 cases:

- By Def. 3.1, we know that (b) **atomic** c is 0-atomic
- From the semantics, we know that $\langle \textbf{atomic}\, c, \sigma\rangle \longmapsto \textsf{abort}$ is false
- From the semantics, we know that $\langle \textbf{atomic}\, c, \sigma\rangle \longmapsto \textsf{race}$ is false

- From the semantics, we know that $\langle \textbf{atomic}\ c, \sigma\rangle \longmapsto \langle \langle\!\langle c\rangle\!\rangle_{\mathsf{a}}\textbf{skip}, \sigma\rangle$, given that $\bullet$ is 0-atomic. From (b), and since $\langle\!\langle c\rangle\!\rangle_{\mathsf{a}}\textbf{skip}$ is 1-atomic, we need to show that for all σ_1 and σ_2, such that (c) $\sigma_2 = \sigma_1 \uplus \sigma$ and (d) $\sigma_1 \in P_s$, $A, G, P_s, Q_s \models \langle \langle\!\langle c\rangle\!\rangle_{\mathsf{a}}\textbf{skip}, \sigma_2\rangle \rhd_{n-1} Q$. From (a), (c), and (d), we know (e) $\sigma_2 \in P_s * P$. From (e), and $\mathsf{Emp}^2, \mathsf{Emp}^2 \models \{\mathsf{Emp}, P_s{*}P\}\, c\, \{\mathsf{Emp}, Q_s{*}Q\}$, by Def. 8.13, we know that (f) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle c, \sigma_2\rangle \rhd_{n-1} (Q_s{*}Q)$. From (f), $P_s \times Q_s \subseteq G$, and $Q_s \subseteq Q_s \circ A$, using Lemma 8.9, we conclude

- We know that $\textbf{atomic}\ c \neq \textbf{skip}$ □

Lemma 8.23.

$$\frac{A \subseteq A' \quad P_s \subseteq P_s' \quad P \subseteq P' \quad A', G' \models \{P_s', P'\}\, c\, \{Q_s', Q'\} \quad G' \subseteq G \quad Q_s' \subseteq Q_s \quad Q' \subseteq Q}{A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $A, G, P_s, Q_s \models \langle c, \sigma\rangle \rhd_n Q$. From (a), and $P \subseteq P'$, we get (b) $\sigma \in P'$. From (b), and $A', G', P_s', Q_s' \models \{P'\}\, c\, \{Q'\}$, by Def. 8.13, we know that (c) $A', G', P_s', Q_s' \models \langle c, \sigma\rangle \rhd_n Q'$. From (c), $A \subseteq A'$, $G' \subseteq G$, $P_s \subseteq P_s'$, $Q_s' \subseteq Q_s$, and $Q' \subseteq Q$, using Lemma 8.3, we conclude □

Lemma 8.24.

$$\frac{\forall x.\ A, G \models \{P_s, \mathcal{P}(x)\}\, c\, \{Q_s, \mathcal{Q}(x)\}}{A, G \models \{P_s, \exists x.\ \mathcal{P}(x)\}\, c\, \{Q_s, \exists x.\ \mathcal{Q}(x)\}}$$

Proof. From Def. 8.13, we assume there exist σ, n, and x, such that (a) $\sigma \in \mathcal{P}(x)$, and we need to show that $A, G, P_s, Q_s \models \langle c, \sigma\rangle \rhd_n (\exists x.\ \mathcal{Q}(x))$. From (a), and $\forall x.\ A, G \models \{P_s, \mathcal{P}(x)\}\, c\, \{Q_s, \mathcal{Q}(x)\}$, by Def. 8.13, we know that (b) $A, G, P_s, Q_s \models \langle c, \sigma\rangle \rhd_n (\mathcal{Q}(x))$. We also know that (c) $(\mathcal{Q}(x)) \subseteq (\exists x.\ \mathcal{Q}(x))$. From (b) and (c), using Lemma 8.3, we conclude □

Lemma 8.25.

$$\frac{A, G \models \{P_s, P\}\, c\, \{Q_s, Q\} \quad I \text{ is precise } \textbf{or}\ \mathsf{img}(G) \text{ and } \mathsf{dom}(A) \text{ coincide}}{I^2{*}A, I^2{*}G \models \{I{*}P_s, P\}\, c\, \{I{*}Q_s, Q\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in P$, and we need to show that $I^2 * A, I^2 * G, I * P_s, I * Q_s \models \langle c, \sigma \rangle \rhd_n Q$. From (a), and $A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}$, by Def. 8.13, we know that (b) $A, G, P_s, Q_s \models \langle c, \sigma \rangle \rhd_n Q$. We also know that (c) c is 0-atomic. From (b), and (c), given that I is precise or $\mathsf{img}(G)$ and $\mathsf{dom}(A)$ coincide, using Lemma 8.10 (item 1), we conclude □

Lemma 8.26.

$$\frac{A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}}{\mathsf{Emp}^2, \mathsf{Emp}^2 \models \{\mathsf{Emp}, P_s * P\}\, c\, \{Q_s * Q\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in P_s * P$, and we need to show that $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \langle c, \sigma \rangle \rhd_n (Q_s * Q)$. From (a) we know that there exists σ_1 and σ_2 such that $\sigma = \sigma_1 \uplus \sigma_2$, (b) $\sigma_1 \in P_s$, and (c) $\sigma_2 \in P$. From (c), and $A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}$, by Def. 8.13, we know that (d) $A, G, P_s, Q_s \models \langle c, \sigma_2 \rangle \rhd_n Q$. We also know that (e) c is 0-atomic. From (d), (e), and (b), using Lemma 8.11 (item 1), we conclude □

Lemma 8.27.

$$\frac{A, G \models \{P_s, P_1\}\, c\, \{Q_s, Q_1\} \quad A, G \models \{P_s, P_2\}\, c\, \{Q_s, Q_2\} \quad \mathsf{img}(G) \text{ is precise}}{A, G \models \{P_s, P_1 \cap P_2\}\, c\, \{Q_s, Q_1 \cap Q_2\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in P_1 \cap P_2$, and we need to show that $A, G, P_s, Q_s \models \langle c, \sigma \rangle \rhd_n (Q_1 \cap Q_2)$. From (a) we know that (b) $\sigma \in P_1$ and (c) $\sigma \in P_2$. From (b), and $A, G \models \{P_s, P_1\}\, c\, \{Q_s, Q_1\}$, by Def. 8.13, we know that (d) $A, G, P_s, Q_s \models \langle c, \sigma \rangle \rhd_n Q_1$. From (c), and $A, G \models \{P_s, P_2\}\, c\, \{Q_s, Q_2\}$, by Def. 8.13, we know that (e) $A, G, P_s, Q_s \models \langle c, \sigma \rangle \rhd_n Q_2$. From (d), (e), and knowing that $\mathsf{img}(G)$ is precise, using Lemma 8.12, we conclude □

Lemma 8.28.

$$\frac{A, G \models \{P_s, P_1\}\, c\, \{Q_s, Q_1\} \quad A, G \models \{P_s, P_2\}\, c\, \{Q_s, Q_2\}}{A, G \models \{P_s, P_1 \cup P_2\}\, c\, \{Q_s, Q_1 \cup Q_2\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in P_1 \cup P_2$, and we need to show that $A, G, P_s, Q_s \models \langle c, \sigma \rangle \rhd_n (Q_1 \cup Q_2)$. From (a) we know that either (b)

$\sigma \in P_1$ or (c) $\sigma \in P_2$. If we assume (b), then from $A, G \models \{P_s, P_1\}\, c\, \{Q_s, Q_1\}$, by Def. 8.13, we know that (d) $A, G, P_s, Q_s \models \langle c, \sigma\rangle \rhd_n Q_1$. We also know that (e) $Q_1 \subseteq Q_1 \cup Q_2$. From (d) and (e), using Lemma 8.3, we conclude. Similarly, if we assume (c), then from $A, G \models \{P_s, P_2\}\, c\, \{Q_s, Q_2\}$, by Def. 8.13, we know that (f) $A, G, P_s, Q_s \models \langle c, \sigma\rangle \rhd_n Q_2$. We also know that (g) $Q_2 \subseteq Q_1 \cup Q_2$. From (f) and (g), using Lemma 8.3, we conclude □

Soundness theorem. The proof structure of Theorem 8.29 is similar for all SAGL rules. It uses all lemmas from Lemma 8.15 to Lemma 8.28, one for each corresponding SAGL rule. The proof structure is modular, if an extra rule is added to Fig. 8.3, we just need to prove an extra lemma for its correspondent semantic rule.

Theorem 8.29. If $A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\}$, then $A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}$

Proof. By strong induction over the derivation tree depth of (a) $A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\}$. After inversion of (a), we have one case for each rule:

- ASSIGNMENT: we know $A = \mathsf{Emp}^2$, $G = \mathsf{Emp}^2$, $P_s = \mathsf{Emp}$, $Q_s = \mathsf{Emp}$, $P = Q \circ [\![\nu := e]\!]$, and $c = (\nu := e)$, using Lemma 8.15, we conclude
- ACTION: we know $A = \mathsf{Emp}^2$, $G = \mathsf{Emp}^2$, $P_s = \mathsf{Emp}$, $Q_s = \mathsf{Emp}$, $P = Q \circ [\![a]\!]$, and $c = \langle a\rangle$, using Lemma 8.16, we conclude.
- SEQUENTIAL: we know $c = (c_1; c_2)$ and that there exists P_s' and P' such that (a) $A, G \vdash \{P_s, P\}\, c_1\, \{P_s', P'\}$ and (b) $A, G \vdash \{P_s', P'\}\, c_2\, \{Q_s, Q\}$. From (a), using the induction hypothesis, we obtain (c) $A, G \models \{P_s, P\}\, c_1\, \{P_s', P'\}$. From (b), using the induction hypothesis, we obtain (d) $A, G \models \{P_s', P'\}\, c_2\, \{Q_s, Q\}$. From (c), and (d), using Lemma 8.17, we conclude
- SKIP: we know $A = \mathsf{Emp}^2$, $G = \mathsf{Emp}^2$, $P_s = \mathsf{Emp}$, $Q_s = \mathsf{Emp}$, $P = Q$, and $c = \mathbf{skip}$, using Lemma 8.18, we conclude.
- CONDITIONAL: we know $c = (\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2)$ and that (a) $P \subseteq \lfloor b\rfloor \cup \lfloor \neg b\rfloor$, (b) $A, G \vdash \{P_s, P \cap \lfloor b\rfloor\}\, c_1\, \{Q_s, Q\}$, and (c) $A, G \vdash \{P_s, P \cap \lfloor \neg b\rfloor\}\, c_2\, \{Q_s, Q\}$. From (b), using the induction hypothesis, we obtain (d) $A, G \models \{P_s, P \cap \lfloor b\rfloor\}\, c_1\, \{Q_s, Q\}$. From

(c), using the induction hypothesis, we obtain (e) $A, G \models \{P_s, P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q_s, Q\}$. From (a), (d), and (e), using Lemma 8.19, we conclude

- LOOP: we know $c = (\textbf{while}\ b\ \textbf{do}\ c)$, $Q = (P \cap \lfloor \neg b \rfloor)$, (a) $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, and (b) $A, G \vdash \{P_s, P \cap \lfloor b \rfloor\}\, c\, \{P_s, P\}$. From (b), using the induction hypothesis, we obtain (c) $A, G \models \{P_s, P \cap \lfloor b \rfloor\}\, c\, \{P_s, P\}$. From (a), and (c), using Lemma 8.20, we conclude

- PARALLEL: we know $A = (A_1 \cap A_2)$, $P_s = (P_{s1} \cap P_{s2})$, $Q_s = (Q_{s1} \cap Q_{s2})$, $P = (P_1 * P_2)$, $c = (c_1 \parallel c_2)$, $Q = (Q_1 * Q_2)$, and that (a) $P_{s1} \subseteq P_{s1} \circ A_1$, (b) $P_{s2} \subseteq P_{s2} \circ A_2$, (c) $A_1, G \cap A_2 \vdash \{P_{s1}, P_1\}\, c_1\, \{Q_{s1}, Q_1\}$ and (d) $A_2, G \cap A_1 \vdash \{P_{s2}, P_2\}\, c_2\, \{Q_{s2}, Q_2\}$. From (a), using the induction hypothesis, we obtain (e) $A_1, G \cap A_2 \models \{P_{s1}, P_1\}\, c_1\, \{Q_{s1}, Q_1\}$. From (b), using the induction hypothesis, we obtain (f) $A_2, G \cap A_1 \models \{P_{s2}, P_2\}\, c_2\, \{Q_{s2}, Q_2\}$. From (a), (b), (e), and (f), using Lemma 8.21, we conclude

- ATOMIC: we know $c = (\textbf{atomic}\ c)$, (a) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \vdash \{P_s * P\}\, c\, \{Q_s * Q\}$, (b) $P_s \times Q_s \subseteq G$, and (c) $Q_s \subseteq Q_s \circ A$. From (a), using the induction hypothesis, we obtain (d) $\mathsf{Emp}^2, \mathsf{Emp}^2, \mathsf{Emp}, \mathsf{Emp} \models \{P_s * P\}\, c\, \{Q_s * Q\}$. From (d), (b), and (c), using Lemma 8.22, we conclude

- CONSEQUENCE: we know that there exists A', P_s', P', G', Q_s', and Q', such that (a) $A \subseteq A'$, (b) $P_s \subseteq P_s'$, (c) $P \subseteq P'$, (d) $G' \subseteq G$, (e) $Q_s' \subseteq Q_s$, (f) $Q' \subseteq Q$, and (g) $A', G' \vdash \{P_s', P'\}\, c\, \{Q_s', Q'\}$. From (g), using the induction hypothesis, we obtain (h) $A', G' \models \{P_s', P'\}\, c\, \{Q_s', Q'\}$. From (a), (b), (c), (d), (e), (f), and (f), using Lemma 8.23, we conclude

- EXISTENTIAL: we know that $P = (\exists x.\ \mathcal{P}(x))$, $Q = (\exists x.\ \mathcal{Q}(x))$, and (a) $\forall x.\ A, G \vdash \{P_s, \mathcal{P}(x)\}\, c\, \{Q_s, \mathcal{Q}(x)\}$. From (a), using the induction hypothesis, we obtain (b) $\forall x.\ A, G \models \{\mathcal{P}_{\int}, \mathcal{P}(x)\}\, c\, \{Q_s, \mathcal{Q}(x)\}$. From (b), using Lemma 8.24, we conclude

- FRAME: we know $A = (I^2 * A')$, $G = (I^2 * G')$, $P_s = (I * P_s')$, $Q_s = (I * Q_s')$, and (a) $A', G' \vdash \{P_s', P\}\, c\, \{Q_s', Q\}$ and (b) either I' is precise or $\mathsf{img}(G')$ and $\mathsf{dom}(A')$ coincide. From (a), using the induction hypothesis, we obtain (c) $A', G' \models \{P_s', P\}\, c\, \{Q_s', Q\}$. From (c) and (b), using Lemma 8.25, we conclude

- RESOURCE: we know $A = \mathsf{Emp}^2$, $G = \mathsf{Emp}^2$, $P_s = \mathsf{Emp}$, $Q_s = \mathsf{Emp}$, $P = (P'_s * P')$, $Q = (Q'_s * Q')$, and (a) $A', G' \vdash \{P'_s, P'\}\, c\, \{Q'_s, Q'\}$. From (a), using the induction hypothesis, we obtain (b) $A', G' \models \{P'_s, P'\}\, c\, \{Q'_s, Q'\}$. From (b), using Lemma 8.26, we conclude

- CONJUNCTION: we know $P = (P_1 \cap P_2)$, $Q = (Q_1 \cap Q_2)$, and that (a) $A, G \vdash \{P_s, P_1\}\, c\, \{Q_s, Q_1\}$, (b) $A, G \vdash \{P_s, P_2\}\, c\, \{Q_s, Q_2\}$, and (c) $\mathsf{img}(G)$ is precise. From (a), using the induction hypothesis, we obtain (d) $A, G \models \{P_s, P_1\}\, c\, \{Q_s, Q_1\}$. From (b), using the induction hypothesis, we obtain (e) $A, G \models \{P_s, P_2\}\, c\, \{Q_s, Q_2\}$. From (d), (e), and (c), using Lemma 8.27, we conclude

- DISJUNCTION: we know $P = (P_1 \cup P_2)$, $Q = (Q_1 \cup Q_2)$, and that
(a) $A, G \vdash \{P_s, P_1\}\, c\, \{Q_s, Q_1\}$ and (b) $A, G \vdash \{P_s, P_2\}\, c\, \{Q_s, Q_2\}$. From (a), using the induction hypothesis, we obtain (c) $A, G \models \{P_s, P_1\}\, c\, \{Q_s, Q_1\}$. From (b), using the induction hypothesis, we obtain (d) $A, G \models \{P_s, P_2\}\, c\, \{Q_s, Q_2\}$. From (c), and (d), using Lemma 8.28, we conclude □

8.3.2 With Regard to the Parameterized Semantics

In this section, we proof the soundness of SAGL with regard to the parameterized semantics of Sec. 3.8. First, we need to define the semantic meaning of a Λ-parameterized CSL quadruple.

Definition 8.30. $A, G \models_{[\Lambda]} \{P_s, P\}\, c\, \{Q_s, Q\}$, if and only if, for all σ, such that $\sigma \in P_s * P$, we have:

1. $\neg [\Lambda]\ \langle c, \sigma \rangle \longmapsto^* \mathsf{abort}$

2. If $[\Lambda]\ \langle c, \sigma \rangle \longmapsto^* \langle \mathbf{skip}, \sigma' \rangle$, then $\sigma' \in Q_s * Q$

This definition is straightforward. The $A, G \models_{[\Lambda]} \{P_s, P\}\, c\, \{Q_s, Q\}$ septuple ensures that for any state satisfying the pre-condition $P_s * P$ will not abort, and, if it the execution completes, the final state will satisfy $Q_s * Q$. Given this definition, we can phrase and prove the soundness theorem below:

Theorem 8.31. If $A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\}$, and Λ provides the DRF-guarantee, then $A, G \models_{[\Lambda]} \{P_s, P\}\, c\, \{Q_s, Q\}$

Proof. From $A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\}$, using Theorem 8.29, we obtain (a) $A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}$. From Def. 8.30, we can prove the goal if, by assuming there is a state σ, such that (b) $\sigma \in P_s * P$, we can establish the following two conditions:

(c) $\neg [\Lambda]\ \langle c, \sigma\rangle \longmapsto^* \textsf{abort}$

(d) If $[\Lambda]\ \langle c, \sigma\rangle \longmapsto^* \langle \textbf{skip}, \sigma'\rangle$, then $\sigma' \in Q_s * Q$

From (a), and (b), using Lemma 8.14, we know that:

(e) $\neg\langle c, \sigma\rangle \longmapsto^* \textsf{abort}$

(f) $\neg\langle c, \sigma\rangle \longmapsto^* \textsf{race}$

(g) If $\langle c, \sigma\rangle \longmapsto^* \langle \textbf{skip}, \sigma'\rangle$, then $\sigma' \in Q_s * Q$

Since Λ provides the DRF-guarantee, from (e), and (f), based on Def. 5.2, we establish (c) and we know

(h) If $[\Lambda]\ \langle c, \sigma\rangle \longmapsto^* \langle \textbf{skip}, \sigma'\rangle$, then $\langle c, \sigma\rangle \longmapsto^* \langle \textbf{skip}, \sigma'\rangle$

From (h), and (g), we can establish (d) $\square$

8.3.3 With Regard to the Relaxed Semantics

The proof of soundness regarding the $\rightsquigarrow$-parameterized relaxed semantics of Sec. 3.10 is straightforward.

Theorem 8.32. If $A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\}$, then $A, G \models_{[\rightsquigarrow]} \{P_s, P\}\, c\, \{Q_s, Q\}$

Proof. From Theorem 5.12, we know that (a) $\rightsquigarrow$ provides the DRF-guarantee. From $A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\}$, and (a), using Theorem 8.31, we prove our goal $\square$

8.4 Extension Rules

In section Sec. 8.2, we have presented the SAGL rules that match the standard rules of CSL from Sec. 6.2. Here we present some extensions.

In Fig. 8.3, we have presented the following FRAME rule, that allows abstracting away a piece of shared memory

$$\frac{A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\} \quad \mathsf{img}(I) \text{ is precise } \textbf{or } \mathsf{img}(G) \text{ and } \mathsf{dom}(A) \text{ coincide}}{I^2 {*} A, I^2 {*} G \vdash \{I {*} Q_s, P\}\, c\, \{I {*} Q_s, Q\}}$$

In CSL, in order to obtain a frame rule for the private memory, we combine the FRAME rule with the RESOURCE rule. However, in SAGL, since both rules have some limitation, we would end up with the following derived rule

$$\frac{\mathsf{Emp}^2, \mathsf{Emp}^2 \vdash \{\mathsf{Emp}, P\}\, c\, \{\mathsf{Emp}, Q\}}{\mathsf{Emp}^2, \mathsf{Emp}^2 \vdash \{\mathsf{Emp}, I {*} P\}\, c\, \{\mathsf{Emp}, I {*} Q\}}$$

This rule is not very flexible as it requires the shared memory to be empty. We can, however, prove a better frame rule for the private memory directly from the semantic rule of Lemma 8.34 (which uses Lemma 8.33 as auxiliary).

Lemma 8.33. If $A, G, P_s, Q_s \models \langle T, \sigma\rangle \rhd_n Q$, and $\sigma' \in I$, then $A, G, P_s, Q_s \models \langle T, \sigma' \uplus \sigma\rangle \rhd_n I {*} Q$

Proof. By induction over n. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, assuming (a) $A, G, P_s, Q_s \models \langle T, \sigma\rangle \rhd_n Q$ and (b) $\sigma' \in I$, we have 5 cases:

- From (a), by Def. 8.1 (item 1), we know that T is 0- or 1-atomic
- From (a), by Def. 8.1 (item 2), we know that (c) $\langle T, \sigma\rangle \longmapsto \mathsf{abort}$ is false. From (c), using Lemma 3.22 (item 1), we conclude
- From (a), by Def. 8.1 (item 3), we know that (d) $\langle T, \sigma\rangle \longmapsto \mathsf{race}$ is false. From (c) and (d), using Lemma 3.22 (item 2), we conclude
- From (c), and $\langle T, \sigma' \uplus \sigma\rangle \longmapsto \langle T', \sigma''\rangle$, using Lemma 3.22 (item 3), we know there exists σ''' such that $\sigma'' = \sigma' \uplus \sigma'''$ and (e) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'''\rangle$. Then we have 4 cases:

– If both T and T' are 1-atomic, we need to show that $A, G, P_s, Q_s \models \langle T', \sigma' \uplus \sigma''' \rangle \rhd_{n-1} (I * Q)$. From (a) and (e), by Def. 8.1 (item 4.a), we know that (f) $A, G, P_s, Q_s \models \langle T', \sigma''' \rangle \rhd_{n-1} Q$. From (f) and (b), by the induction hypothesis, we conclude

– If both T and T' are 0-atomic, we need to show that $\mathsf{dom}(\sigma' \uplus \sigma) = \mathsf{dom}(\sigma' \uplus \sigma''')$ and $A, G, P_s, Q_s \models \langle T', \sigma' \uplus \sigma''' \rangle \rhd_{n-1} (I * Q)$. From (a) and (e), by Def. 8.1 (item 4.b), we know that (f) $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma''')$, and also that (g) $A, G, P_s, Q_s \models \langle T', \sigma''' \rangle \rhd_{n-1} Q$. From (g) and (b), by the induction hypothesis, we have (h) $A, G, P_s, Q_s \models \langle T', \sigma' \uplus \sigma''' \rangle \rhd_{n-1} Q$. From (f) and (h), we conclude

– If T is 0-atomic and T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (f) $\sigma_2 = \sigma_1 \uplus \sigma' \uplus \sigma'''$ and (g) $\sigma_1 \in P_s$, we have $A, G, P_s, Q_s \models \langle T', \sigma_2 \rangle \rhd_{n-1} (I * Q)$. From (a), (e), and (g), by Def. 8.1 (item 4.c), we know that (h) $A, G, P_s, Q_s \models \langle T', \sigma_1 \uplus \sigma''' \rangle \rhd_{n-1} Q$. From (h) and (b), by the induction hypothesis, we conclude

– If T is 1-atomic and T' is 0-atomic, we need to show that for all σ'''', such that (f) $\sigma'''' \in P_s$, there exists P_s', where $P_s \times P_s' \subseteq G$ and $P_s' \subseteq P_s' \circ A$, and a pair of states, σ_1 and σ_2, such that $\sigma' \uplus \sigma''' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in P_s'$, and $A, G, P_s', Q_s \models \langle T', \sigma_2 \rangle \rhd_{n-1} (I * Q)$. From (a), (e), and (f), by Def. 8.1 (item 4.d), we know there exists P_s'', where (g) $P_s \times P_s'' \subseteq G$ and (h) $P_s'' \subseteq P_s'' \circ A$, and a pair of states, σ_1' and σ_2', such that (i) $\sigma''' = \sigma_1' \uplus \sigma_2'$, (j) $\sigma_1' \in P_s''$, and (k) $A, G, P_s'', Q_s \models \langle T', \sigma_2' \rangle \rhd_{n-1} Q$. From (k) and (b), by the induction hypothesis, we have (l) $A, G, P_s'', Q_s \models \langle T', \sigma' \uplus \sigma_2' \rangle \rhd_{n-1} (I * Q)$. Instantiating the goal with P_s'', σ_1', and $\sigma' \uplus \sigma_2'$, from (g), (h), (i), (j), and (l), we conclude

- We assume (c) $T = \textbf{skip}$. From (a) and (c), by Def. 8.1 (item 5), we know that (d) $P_s \subseteq Q_s$ and (e) $\sigma \in Q$. From (b) and (e), we know that (f) $\sigma' \uplus \sigma \in I * Q$. From (d) and (f), we conclude $\square$

Lemma 8.34.

$$\frac{A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}}{A, G \models \{P_s, I * P\}\, c\, \{Q_s, I * Q\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in I{*}P$, and we need to show that $A, G, P_s, Q_s \models \langle c, \sigma\rangle \rhd_n (I{*}Q)$. From (a) we know that exists σ_1 and σ_2 such that $\sigma = \sigma_1 \uplus \sigma_2$, (b) $\sigma_1 \in I$, and (c) $\sigma_2 \in P$. From (c), and $A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}$, by Def. 8.13, we know that (d) $A, G, P_s, Q_s \models \langle c, \sigma_2\rangle \rhd_n Q$. From (d), and (a), using Lemma 8.33, we conclude $\square$

In Fig. 8.3, we have presented the following CONJUNCTION rule

$$\frac{A, G \vdash \{P_s, P_1\}\, c\, \{Q_s, Q_1\} \quad A, G \vdash \{P_s, P_2\}\, c\, \{Q_s, Q_2\} \quad \mathsf{img}(G) \text{ is precise}}{A, G \vdash \{P_s, P_1 {\cap} P_2\}\, c\, \{Q_s, Q_1 {\cap} Q_2\}}$$

which has the constraint that $\mathsf{img}(G)$ is precise. We can generalize this rule to the following

$$\frac{\begin{array}{c} A_1, G_1 \vdash \{P_{s1}, P_1\}\, c\, \{Q_{s1}, Q_1\} \quad A_2, G_2 \vdash \{P_{s2}, P_2\}\, c\, \{Q_{s2}, Q_2\} \\ \mathsf{img}(G_1) \text{ and } \mathsf{img}(G_2) \text{ coincide} \end{array}}{A_1 {\cap} A_2, G_1 {\cap} G_2 \vdash \{P_{s1} {\cap} P_{s2}, P_1 {\cap} P_2\}\, c\, \{Q_{s1} {\cap} Q_{s2}, Q_1 {\cap} Q_2\}} \ (\text{CONJUNCTION})$$

which requires the image of G_1 and G_2 to coincide (as in Def. 6.35).

In order to prove the soundness of this generalized rule, we need the following two lemmas:

Lemma 8.35. If $A_1, G_1, P_{s1}, Q_{s1} \models \langle T, \sigma\rangle \rhd_n Q_1$, $A_2, G_2, P_{s2}, Q_{s2} \models \langle T, \sigma\rangle \rhd_n Q_2$, and $\mathsf{img}(G_1)$ and $\mathsf{img}(G_2)$ coincide, then $A_1{\cap}A_2, G_1{\cap}G_2, P_{s1}{\cap}P_{s2}, Q_{s1}{\cap}Q_{s2} \models \langle T, \sigma\rangle \rhd_n (Q_1{\cap}Q_2)$

Proof. By induction over n. If $n = 0$, by Def. 8.1, we conclude. If $n > 0$, by Def. 8.1, assuming (a) $A_1, G_1, P_{s1}, Q_{s1} \models \langle T, \sigma\rangle \rhd_n Q_1$ and (b) $A_2, G_2, P_{s2}, Q_{s2} \models \langle T, \sigma\rangle \rhd_n Q_2$, we have 5 cases:

- From (a), by Def. 8.1 (item 1), we know that T is 0- or 1-atomic
- From (a), by Def. 8.1 (item 2), we know that $\langle T, \sigma\rangle \longmapsto \mathsf{abort}$ is false
- From (a), by Def. 8.1 (item 3), we know that $\langle T, \sigma\rangle \longmapsto \mathsf{race}$ is false

- If (c) $\langle T, \sigma\rangle \longmapsto \langle T', \sigma'\rangle$, then we have 4 cases:
  - If both T and T' are 1-atomic, we need to show that $A_1 \cap A_2, G_1 \cap G_2, P_{s1} \cap P_{s2}, Q_{s1} \cap Q_{s2} \models \langle T', \sigma'\rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (c), by Def. 8.1 (item 4.a), we know that (d) $A_1, G_1, P_{s1}, Q_{s1} \models \langle T', \sigma'\rangle \rhd_{n-1} Q_1$. From (b) and (c), by Def. 8.1 (item 4.a), we know that (e) $A_2, G_2, P_{s2}, Q_{s2} \models \langle T', \sigma'\rangle \rhd_{n-1} Q_2$. From (d) and (e), by the induction hypothesis, we conclude
  - If both T and T' are 0-atomic, we need to show that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $A_1 \cap A_2, G_1 \cap G_2, P_{s1} \cap P_{s2}, Q_{s1} \cap Q_{s2} \models \langle T', \sigma'\rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (a) and (c), by Def. 8.1 (item 4.b), we know already that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$, and also that (d) $A_1, G_1, P_{s1}, Q_{s1} \models \langle T', \sigma'\rangle \rhd_{n-1} Q_1$. From (b) and (c), by Def. 8.1 (item 4.b), we also know that $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and (e) $A_2, G_2, P_{s2}, Q_{s2} \models \langle T', \sigma'\rangle \rhd_{n-1} Q_2$. From (d) and (e), by the induction hypothesis, we conclude
  - If T is 0-atomic and T' is 1-atomic, we need to show that for all σ_1 and σ_2, such that (d) $\sigma_2 = \sigma_1 \uplus \sigma'$ and (e) $\sigma_1 \in P_{s1} \cap P_{s2}$, we have $A_1 \cap A_2, G_1 \cap G_2, P_{s1} \cap P_{s2}, Q_{s1} \cap Q_{s2} \models \langle T', \sigma_2\rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (e), we know that (f) $\sigma_1 \in P_{s1}$ and (g) $\sigma_1 \in P_{s2}$. From (a), (c), (d), and (f), by Def. 8.1 (item 4.c), we know that (h) $A_1, G_1, P_{s1}, Q_{s1} \models \langle T', \sigma_2\rangle \rhd_{n-1} Q_1$. From (b), (c), (d), and (g), by Def. 8.1 (item 4.c), we know that (i) $A_2, G_2, P_{s2}, Q_{s2} \models \langle T', \sigma_2\rangle \rhd_{n-1} Q_2$. From (h) and (i), by the induction hypothesis, we conclude
  - If T is 1-atomic and T' is 0-atomic, we need to show that for all σ'', such that (d) $\sigma'' \in P_{s1} \cap P_{s2}$, there exists P'_s, where $(P_{s1} \cap P_{s2}) \times P'_s \subseteq (G_1 \cap G_2)$ and $P'_s \subseteq P'_s \circ (A_1 \cap A_2)$, and a pair of states, σ_1 and σ_2, such that $\sigma' = \sigma_1 \uplus \sigma_2$, $\sigma_1 \in P'_s$, and $A_1 \cap A_2, G_1 \cap G_2, P'_s, Q_{s1} \cap Q_{s2} \models \langle T', \sigma_2\rangle \rhd_{n-1} (Q_1 \cap Q_2)$. From (d), we know (e) $\sigma'' \in P_{s1}$ and (f) $\sigma'' \in P_{s2}$. From (a), (c), and (e), by Def. 8.1 (item 4.d), we know there exists P'_{s1}, where (g) $P_{s1} \times P'_{s1} \subseteq G_1$ and (h) $P'_{s1} \subseteq P'_{s1} \circ A_1$, and a pair of states, σ'_1 and σ'_2, such that (i) $\sigma' = \sigma'_1 \uplus \sigma'_2$, (j) $\sigma'_1 \in P'_{s1}$, and (k) $A_1, G_1, P'_{s1}, Q_{s1} \models \langle T', \sigma'_2\rangle \rhd_{n-1} Q_1$. From (b), (c), and (f), by Def. 8.1 (item 4.d), we know there exists P'_{s2}, where (l) $P_{s2} \times P'_{s2} \subseteq G_2$ and (m) $P'_{s2} \subseteq P'_{s2} \circ A_2$, and

a pair of states, σ_1'' and σ_2'', such that (n) $\sigma' = \sigma_1'' \uplus \sigma_2''$, (o) $\sigma_1'' \in P_{s2}'$, and (p) $A_2, G_2, P_{s2}', Q_{s2} \models \langle T', \sigma_2'' \rangle \rhd_{n-1} Q_2$. From (e), (f), (j), (o), (g), (l), (i), and (n) given that $\mathsf{img}(G_1)$ and $\mathsf{img}(G_2)$ coincide, we know that $\sigma_1' = \sigma_1''$ and $\sigma_2' = \sigma_2''$. From (g) and (l), we have (q) $(P_{s1} \cap P_{s2}) \times (P_{s1}' \cap P_{s2}') \subseteq (G_1 \cap G_2)$. From (h) and (m), we have (r) $(P_{s1}' \cap P_{s2}') \subseteq (P_{s1}' \cap P_{s2}') \circ (A_1 \cap A_2)$. From (k) and (p), by the induction hypothesis, we have (s) $A_1 \cap A_2, G_1 \cap G_2, P_{s1}' \cap P_{s2}', Q_{s1} \cap Q_{s2} \models \langle T', \sigma_2' \rangle \rhd_{n-1} (Q_1 \cap Q_2)$. Instantiating the goal with $(P_{s1}' \cap P_{s2}')$, σ_1', and σ_2', from (q), (r), (i), (j), (o), and (s), we conclude

- We assume (c) $T = \textbf{skip}$. From (a) and (c), by Def. 8.1 (item 5), we know that (d) $P_{s1} \subseteq Q_{s1}$ and (e) $\sigma \in Q_1$. From (b) and (c), by Def. 8.1 (item 5), we know that (f) $P_{s2} \subseteq Q_{s2}$ and (g) $\sigma \in Q_2$. From (d) and (f), we know that (h) $(P_{s1} \cap P_{s2}) \subseteq (Q_{s1} \cap Q_{s2})$. From (e) and (g), we know that (i) $\sigma \in Q_1 \cap Q_2$. From (h) and (i), we conclude $\square$

Lemma 8.36.

$$\frac{A_1, G_1 \models \{P_{s1}, P_1\}\, c\, \{Q_{s1}, Q_1\} \quad A_2, G_2 \models \{P_{s2}, P_2\}\, c\, \{Q_{s2}, Q_2\} \qquad \mathsf{img}(G_1) \text{ and } \mathsf{img}(G_2) \text{ coincide}}{A_1 \cap A_2, G_1 \cap G_2 \models \{P_{s1} \cap P_{s2}, P_1 \cap P_2\}\, c\, \{Q_{s1} \cap Q_{s2}, Q_1 \cap Q_2\}}$$

Proof. From Def. 8.13, we assume there exist σ and n, such that (a) $\sigma \in P_1 \cap P_2$, and we need to show that $A_1 \cap A_2, G_1 \cap G_2, P_{s1} \cap P_{s2}, Q_{s1} \cap Q_{s2} \models \langle c, \sigma \rangle \rhd_n (Q_1 \cap Q_2)$. From (a) we know that (b) $\sigma \in P_1$ and (c) $\sigma \in P_2$. From (b), and $A_1, G_1 \models \{P_{s1}, P_1\}\, c\, \{Q_{s1}, Q_1\}$, by Def. 8.13, we know that (d) $A_1, G_1, P_{s1}, Q_{s1} \models \langle c, \sigma \rangle \rhd_n Q_1$. From (c), and $A_2, G_2 \models \{P_{s2}, P_2\}\, c\, \{Q_{s2}, Q_2\}$, by Def. 8.13, we know that (e) $A_2, G_2, P_{s2}, Q_{s2} \models \langle c, \sigma \rangle \rhd_n Q_2$. From (d), (e), and knowing that $\mathsf{img}(G_1)$ and $\mathsf{img}(G_2)$ coincide, using Lemma 8.35, we conclude $\square$

8.5 Embedding CSL in SAGL

It turns out that SAGL is a generalization of CSL. Many of the CSL rules from Fig. 6.3 can be embedded in SAGL by providing a simple interpretation of CSL judgments into SAGL judgements as show in Fig. 8.4.

$$[\![I \vdash \{P\}\, c\, \{Q\}]\!] \overset{\text{def}}{=} I^2, I^2 \vdash \{I, P\}\, c\, \{I, Q\}$$

Figure 8.4: Interpretation of CSL into SAGL

We can then prove most of CSL rules as lemmas, just as they are.

Lemma 8.37.

$$\frac{}{[\![\mathsf{Emp} \vdash \{Q \circ [\![\nu := e]\!]\}\, \nu := e\, \{Q\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $\mathsf{Emp}^2, \mathsf{Emp}^2 \vdash \{\mathsf{Emp}, Q \circ [\![\nu := e]\!]\}\, \nu := e\, \{\mathsf{Emp}, Q\}$, which we obtain directly from rule ASSIGNMENT □

Lemma 8.38.

$$\frac{}{[\![\mathsf{Emp} \vdash \{Q \circ [\![a]\!]\}\, \langle a \rangle\, \{Q\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $\mathsf{Emp}^2, \mathsf{Emp}^2 \vdash \{\mathsf{Emp}, Q \circ [\![a]\!]\}\, \langle a \rangle\, \{\mathsf{Emp}, Q\}$, which we obtain directly from rule ACTION □

Lemma 8.39.

$$\frac{[\![I \vdash \{P\}\, c_1\, \{P'\}]\!] \quad [\![I \vdash \{P'\}\, c_2\, \{Q\}]\!]}{[\![I \vdash \{P\}\, c_1; c_2\, \{Q\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $I^2, I^2 \vdash \{I, P\}\, c_1; c_2\, \{I, Q\}$. From $[\![I \vdash \{P\}\, c_1\, \{P'\}]\!]$, unfolding the interpretation, we have (a) $I^2, I^2 \vdash \{I, P\}\, c_1\, \{I, P'\}$. From $[\![I \vdash \{P'\}\, c_2\, \{Q\}]\!]$, unfolding the interpretation, we have (b) $I^2, I^2 \vdash \{I, P'\}\, c_2\, \{I, Q\}$. From (a) and (b), using rule SEQUENTIAL, we conclude □

Lemma 8.40.

$$\frac{}{[\![\mathsf{Emp} \vdash \{P\}\, \textbf{skip}\, \{P\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that
$\mathsf{Emp}^2, \mathsf{Emp}^2 \vdash \{\mathsf{Emp}, P\}\, \textbf{skip}\, \{\mathsf{Emp}, P\}$, which we obtain directly from rule SKIP □

Lemma 8.41.

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad [\![I \vdash \{P \cap \lfloor b \rfloor\}\, c_1\, \{Q\}]\!] \quad [\![I \vdash \{P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q\}]\!]}{[\![I \vdash \{P\}\, \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\, \{Q\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $I^2, I^2 \vdash \{I, P\}\, \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\, \{I, Q\}$. From $[\![I \vdash \{P \cap \lfloor b \rfloor\}\, c_1\, \{Q\}]\!]$, unfolding the interpretation, we have (a) $I^2, I^2 \vdash \{I, P \cap \lfloor b \rfloor\}\, c_1\, \{I, Q\}$. From $[\![I \vdash \{P \cap \lfloor \neg b \rfloor\}\, c_2\, \{Q\}]\!]$, unfolding the interpretation, we have (b) $I^2, I^2 \vdash \{I, P \cap \lfloor \neg b \rfloor\}\, c_2\, \{I, Q\}$. From (a), (b), and $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, using rule CONDITIONAL, we conclude □

Lemma 8.42.

$$\frac{P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor \quad [\![I \vdash \{P \cap \lfloor b \rfloor\}\, c\, \{P\}]\!]}{[\![I \vdash \{P\}\, \mathbf{while}\ b\ \mathbf{do}\ c\, \{P \cap \lfloor \neg b \rfloor\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $I^2, I^2 \vdash \{I, P\}\, \mathbf{while}\ b\ \mathbf{do}\ c\, \{I, P \cap \lfloor \neg b \rfloor\}$. From $[\![I \vdash \{P \cap \lfloor b \rfloor\}\, c\, \{P\}]\!]$, unfolding the interpretation, we have (a) $I^2, I^2 \vdash \{I, P \cap \lfloor b \rfloor\}\, c\, \{I, P\}$. From (a) and $P \subseteq \lfloor b \rfloor \cup \lfloor \neg b \rfloor$, using rule LOOP, we conclude □

Lemma 8.43.

$$\frac{[\![I \vdash \{P_1\}\, c_1\, \{Q_1\}]\!] \quad [\![I \vdash \{P_2\}\, c_2\, \{Q_2\}]\!]}{[\![I \vdash \{P_1 * P_2\}\, c_1 \parallel c_2\, \{Q_1 * Q_2\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $I^2, I^2 \vdash \{I, P_1 * P_2\}\, c_1 \parallel c_2\, \{I, Q_1 * Q_2\}$, which is equivalent to $I^2 \cap I^2, I^2 \vdash \{I \cap I, P_1 * P_2\}\, c_1 \parallel c_2\, \{I \cap I, Q_1 * Q_2\}$. From $[\![I \vdash \{P_1\}\, c_1\, \{Q_1\}]\!]$, unfolding the interpretation, we have $I^2, I^2 \vdash \{I, P_1\}\, c_1\, \{I, Q_1\}$, which is equivalent to (a) $I^2, I^2 \cap I^2 \vdash \{I, P_1\}\, c_1\, \{I, Q_1\}$. From $[\![I \vdash \{P_2\}\, c_2\, \{Q_2\}]\!]$, unfolding the interpretation, we have $I^2, I^2 \vdash \{I, P_2\}\, c_2\, \{I, Q_2\}$, which is equivalent to (b) $I^2, I^2 \cap I^2 \vdash \{I, P_2\}\, c_2\, \{I, Q_2\}$. From (a), (b), and $I \subseteq I \circ I^2$, using rule PARALLEL, we conclude □

Lemma 8.44.

$$\frac{[\![\mathsf{Emp} \vdash \{I * P\}\, c\, \{I * Q\}]\!]}{[\![I \vdash \{P\}\, \mathbf{atomic}\ c\, \{Q\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $I^2, I^2 \vdash \{I, P\}\, \mathbf{atomic}\ c\, \{I, Q\}$. From $[\![\mathsf{Emp} \vdash \{I * P\}\, c\, \{I * Q\}]\!]$, unfolding the interpretation, we have (a) $\mathsf{Emp}^2, \mathsf{Emp}^2 \vdash \{\mathsf{Emp}, I * P\}\, c\, \{\mathsf{Emp}, I * Q\}$. From (a), $I \times I \subseteq I^2$, and $I \subseteq I \circ I^2$, using rule ATOMIC, we conclude □

Lemma 8.45.

$$\frac{P \subseteq P' \quad [\![I \vdash \{P'\}\, c\, \{Q'\}]\!] \quad Q' \subseteq Q}{[\![I \vdash \{P\}\, c\, \{Q\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $I^2, I^2 \vdash \{I, P\}\, c\, \{I, Q\}$. From $[\![I \vdash \{P'\}\, c\, \{Q'\}]\!]$, unfolding the interpretation, we have (a) $I^2, I^2 \vdash \{I, P'\}\, c\, \{I, Q'\}$. From (a), $P \subseteq P'$, and $Q' \subseteq Q$, using rule CONSEQUENCE, we conclude □

Lemma 8.46.

$$\frac{\forall x.\ [\![I \vdash \{\mathcal{P}(x)\}\, c\, \{\mathcal{Q}(x)\}]\!]}{[\![I \vdash \{\exists x.\ \mathcal{P}(x)\}\, c\, \{\exists x.\ \mathcal{Q}(x)\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $I^2, I^2 \vdash \{I, \exists x.\ \mathcal{P}(x)\}\, c\, \{I, \exists x.\ \mathcal{Q}(x)\}$. From $\forall x.\ [\![I \vdash \{\mathcal{P}(x)\}\, c\, \{\mathcal{Q}(x)\}]\!]$, unfolding the interpretation, we have (a) $\forall x.\ I^2, I^2 \vdash \{I, \mathcal{P}(x)\}\, c\, \{I, \mathcal{Q}(x)\}$. From (a), using rule EXISTENTIAL, we conclude □

The CSL frame rule can be embedded but needs and extra precision requirement over I' or I.

Lemma 8.47.

$$\frac{[\![I \vdash \{P\}\, c\, \{Q\}]\!] \quad \text{I}' \text{ is precise } \textbf{or} \text{ I is precise}}{[\![I' {*} I \vdash \{P\}\, c\, \{Q\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $(I'{*}I)^2, (I'{*}I)^2 \vdash \{I'{*}I, P\}\, c\, \{I'{*} I, Q\}$, which is equivalent to $I'^2 {*} I^2, I'^2 {*} I^2 \vdash \{I' {*} I, P\}\, c\, \{I' {*} I, Q\}$. From $[\![I \vdash \{P\}\, c\, \{Q\}]\!]$, unfolding the interpretation, we have (a) $I^2, I^2 \vdash \{I, P\}\, c\, \{I, Q\}$. Given that I' is precise or I is precise, using Remark 6.36, we know that (b) either I' is precise or $\mathsf{img}(I^2)$ and $\mathsf{dom}(I^2)$ coincide. From (a), and (b), using rule FRAME, we conclude □

Due to the limited SAGL resource rule, the embedding of the CSL resource rule also needs to be limited.

Lemma 8.48.

$$\frac{[\![I \vdash \{P\}\, c\, \{Q\}]\!]}{[\![\mathsf{Emp} \vdash \{I {*} P\}\, c\, \{I {*} Q\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $\mathsf{Emp}^2, \mathsf{Emp}^2 \vdash \{\mathsf{Emp}, I * P\}\, c\, \{\mathsf{Emp}, I * Q\}$. From $[\![I \vdash \{P\}\, c\, \{Q\}]\!]$, unfolding the interpretation, we have (a) $I^2, I^2 \vdash \{I, P\}\, c\, \{I, Q\}$. From (a), using rule RESOURCE, we conclude □

Lemma 8.49.

$$\frac{[\![I \vdash \{P_1\}\, c\, \{Q_1\}]\!] \quad [\![I \vdash \{P_2\}\, c\, \{Q_2\}]\!] \quad I \text{ is precise}}{[\![I \vdash \{P_1 \cap P_2\}\, c\, \{Q_1 \cap Q_2\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $I^2, I^2 \vdash \{I, P_1 \cap P_2\}\, c\, \{I, Q_1 \cap Q_2\}$. From $[\![I \vdash \{P_1\}\, c\, \{Q_1\}]\!]$, unfolding the interpretation, we have
(a) $I^2, I^2 \vdash \{I, P_1\}\, c\, \{I, Q_1\}$. From $[\![I \vdash \{P_2\}\, c\, \{Q_2\}]\!]$, unfolding the interpretation, we have (b) $I^2, I^2 \vdash \{I, P_2\}\, c\, \{I, Q_2\}$. Given that I is precise, we know that (c) $\mathsf{img}(I^2)$ is precise. From (a), (b), and (c), using rule CONJUNCTION, we conclude □

Lemma 8.50.

$$\frac{[\![I \vdash \{P_1\}\, c\, \{Q_1\}]\!] \quad [\![I \vdash \{P_2\}\, c\, \{Q_2\}]\!]}{[\![I \vdash \{P_1 \cup P_2\}\, c\, \{Q_1 \cup Q_2\}]\!]}$$

Proof. If we unfold the interpretation, we need to show that $I^2, I^2 \vdash \{I, P_1 \cup P_2\}\, c\, \{I, Q_1 \cup Q_2\}$. From $[\![I \vdash \{P_1\}\, c\, \{Q_1\}]\!]$, unfolding the interpretation, we have
(a) $I^2, I^2 \vdash \{I, P_1\}\, c\, \{I, Q_1\}$. From $[\![I \vdash \{P_2\}\, c\, \{Q_2\}]\!]$, unfolding the interpretation, we have (b) $I^2, I^2 \vdash \{I, P_2\}\, c\, \{I, Q_2\}$. From (a), and (b), using rule DISJUNCTION, we conclude □

8.6 Verification Examples

In this section, we provide some examples of programs verified using SAGL.

8.6.1 Dekker's Algorithm

Here we provide a proof for the Dekker's algorithm implementation presented in Fig. 7.6 in Sec. 7.4.2. The proof idea is similar but, instead of using partial permissions to prevent some of the memory locations from being modified, we use rely-guarantee instead.

In order to define the assume and guarantee conditions for both threads, we define $\mathsf{dekkers}(I)$ which is a shared-memory invariant very similar to the one presented in

Sec. 7.4.2, except that we discarded the partial permissions. We carry out this invariant through out the verification of both threads, however we strengthen it, by perform a conjunction, where required. Based on the shared invariant we define the rely for the left hand side thread ($\mathsf{A1}$) by simply forcing the maintenance of the invariant plus the preservation of `cs1`, `a1`, and the location pointed by `a1`. The guarantee of the left hand side thread ($\mathsf{G1}$) also forces the maintenance of the invariant plus the preservation of `cs2`, `a2`, and the location pointed by `a2`. Naturally, the assumption of the left hand side thread is the guarantee of the right hand side thread, and vice versa.

$$
\begin{array}{lcl}
\mathsf{dekkers}(I) & \stackrel{\text{def}}{=} & \exists v_1, v_2, c_1, c_2.\ \texttt{cs1} \mapsto c_1 * \texttt{cs2} \mapsto c_2 \\
& & \quad * ((c_1 \neq 0 \cup c_2 \neq 0) \cap \mathsf{Emp} \cup (c_1 = 0 \cap c_2 = 0) \cap I) \\
& & \quad * (\exists p.\ \texttt{a1} \mapsto p * p \mapsto v_1) * (\exists p.\ \texttt{a2} \mapsto p * p \mapsto v_2) \\
& & \quad \cap (v_1 = 0 \cap c_1 = 0 \cup v_1 = 1) \cap (v_2 = 0 \cap c_2 = 0 \cup v_2 = 1) \\
\mathsf{A1}(I), \mathsf{G2}(I) & \stackrel{\text{def}}{=} & \exists p_1, v_1, c_1.\ (\texttt{a1} \mapsto p_1 * p_1 \mapsto v_1 * \texttt{cs1} \mapsto c_1 * \mathsf{True}) \\
& & \quad \times ((\texttt{a1} \mapsto p_1 * p_1 \mapsto v_1 * \texttt{cs1} \mapsto c_1 * \mathsf{True}) \cap \mathsf{dekkers}(I)) \\
\mathsf{A2}(I), \mathsf{G1}(I) & \stackrel{\text{def}}{=} & \exists p_2, v_2, c_2.\ (\texttt{a2} \mapsto p_2 * p_2 \mapsto v_2 * \texttt{cs2} \mapsto c_2 * \mathsf{True}) \\
& & \quad \times ((\texttt{a2} \mapsto p_2 * p_2 \mapsto v_2 * \texttt{cs2} \mapsto c_2 * \mathsf{True}) \cap \mathsf{dekkers}(I))
\end{array}
$$

Now, with these definitions in mind, we can show the verification of Dekker's algorithm using SAGL. Note that precision is required for I. Here we only show the verification for the left hand side thread. The verification of the right hand side thread is symmetric.

```
$\forall I.\ \mathsf{A1}(I), \mathsf{G1}(I) \vdash$
    $\{(\exists p.\ \texttt{a1} \mapsto p * p \mapsto 0 * \texttt{cs1} \mapsto 0 * \mathsf{True}) \cap \mathsf{dekkers}(I), \texttt{v1} \mapsto \_\}$
    **atomic** $[\texttt{a1}] := 1$;
    $\{(\exists p.\ \texttt{a1} \mapsto p * p \mapsto 1 * \texttt{cs1} \mapsto 0 * \mathsf{True}) \cap \mathsf{dekkers}(I), \texttt{v1} \mapsto \_\}$
    **atomic** $(\texttt{v1} := [\texttt{a2}]; \texttt{cs1} := -(\texttt{v1} - 1))$;
    $\{(\exists p.\ \texttt{a1} \mapsto p * p \mapsto 1 * \texttt{cs1} \mapsto \_ * \mathsf{True}) \cap \mathsf{dekkers}(I), \texttt{v1} \mapsto 1 \cup (\texttt{v1} \mapsto 0 * I)\}$
    **if** $\texttt{v1} = 0$ **then**
        $\{(\exists p.\ \texttt{a1} \mapsto p * p \mapsto 1 * \texttt{cs1} \mapsto \_ * \mathsf{True}) \cap \mathsf{dekkers}(I), \texttt{v1} \mapsto 0 * I\}$
        /* *critical section* */;
        $\{(\exists p.\ \texttt{a1} \mapsto p * p \mapsto 1 * \texttt{cs1} \mapsto \_ * \mathsf{True}) \cap \mathsf{dekkers}(I), \texttt{v1} \mapsto 0 * I\}$
    $\{(\exists p.\ \texttt{a1} \mapsto p * p \mapsto 1 * \texttt{cs1} \mapsto \_ * \mathsf{True}) \cap \mathsf{dekkers}(I), \texttt{v1} \mapsto 1 \cup (\texttt{v1} \mapsto 0 * I)\}$
    **atomic** $([\texttt{a1}] := 0; \texttt{cs1} := 0)$
    $\{(\exists p.\ \texttt{a1} \mapsto p * p \mapsto 0 * \texttt{cs1} \mapsto 0 * \mathsf{True}) \cap \mathsf{dekkers}(I), \texttt{v1} \mapsto \_\}$
```

Chapter 9

Cross Rewriting and Partial Barriers

In this chapter, we provide a summary of two extensions to our framework. These are here to enlighten the reader by showing that the ideas presented can be supported for more sophisticated goals. The first extension is *cross rewriting* that allows private values to be propagated around an atomic block or parallel composition. The second extension is the support for partial barriers where a command might be swapped with an atomic block in one direction.

9.1 Cross Rewriting

Before we present partial barriers we need to present cross rewriting. Cross rewriting allows the rewriting an atomic block based on the private operations that precede it. Or the other way around, it allows rewriting private operations based on the atomic block that precedes them. For example, subsumption (as defined in Def. 3.33) does not allow the following kind of transformations:

$$(\mathtt{x}:=3;\langle\mathtt{y}:=\mathtt{x}\rangle;\mathtt{z}:=\mathtt{y})\not\rightsquigarrow(\mathtt{x}:=3;\langle\mathtt{y}:=3\rangle;\mathtt{z}:=3)$$

Here we provide an extended version of subsumption, which we call cross-rewrite subsumption. Def. 9.2 redefines the semantics of $\rightsquigarrow$ in order to support on cross-rewrite. Def. 9.1 defines ϕ_c which is the set of states that result from the evaluation of $c's$ private "tail", i.e. the commands that follow the last barrier of c.

Definition 9.1. $\sigma \in \phi_c$ if, and only if, either

1. exists σ' such that $\langle c, \sigma' \rangle \Downarrow \langle \textbf{skip}, \sigma \rangle$
2. exists σ', $\mathbf{S}$, c', and σ'' such that $\langle c, \sigma' \rangle \Downarrow \langle \mathbf{S}[\,\textbf{atomic}\ c'\,], \sigma'' \rangle$ and $\sigma \in \phi_{\mathbf{S}[\,\textbf{skip}\,]}$
3. exists σ', $\mathbf{S}$, c_1, c_2, and σ'' such that $\langle c, \sigma' \rangle \Downarrow \langle \mathbf{S}[\,c_1 \parallel c_2\,], \sigma'' \rangle$ and $\sigma \in \phi_{\mathbf{S}[\,\textbf{skip}\,]}$

Definition 9.2. $c_1 \rightsquigarrow_0^\sigma c_2$ always holds; $c_1 \rightsquigarrow_{n+1}^\sigma c_2$ holds if, and only if, the following are true:

1. If $\langle c_2, \sigma \rangle \longrightarrow^* \textsf{abort}$, then $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$;
2. If $\langle c_2, \sigma \rangle \Downarrow \langle c_2', \sigma' \rangle$, then either $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$,
   or there exists c_1' such that $\langle c_1, \sigma \rangle \Downarrow \langle c_1', \sigma' \rangle$ and the following constraints hold:
   (a) if $c_2' = \textbf{skip}$, then $c_1' = \textbf{skip}$;
   (b) if $c_2' = \mathbf{S}_2[\,c_2'' \parallel c_2'''\,]$, there exist $\mathbf{S}_1$, c_1'' and c_1''' such that
      i. $c_1' = \mathbf{S}_1[\,c_1'' \parallel c_1'''\,]$;
      ii. $c_1'' \rightsquigarrow_n^{\sigma'} c_2''$;
      iii. $c_1''' \rightsquigarrow_n^{\sigma'} c_2'''$;
      iv. If $\sigma'' \in \phi_{c_2''} \cap \phi_{c_2'''}$, then $\mathbf{S}_1[\,\textbf{skip}\,] \rightsquigarrow_n^{\sigma''} \mathbf{S}_2[\,\textbf{skip}\,]$;
   (c) if $c_2' = \mathbf{S}_2[\,\textbf{atomic}\ c_2''\,]$, there exist $\mathbf{S}_1$ and c_1'' such that
      i. $c_1' = \mathbf{S}_1[\,\textbf{atomic}\ c_1''\,]$;
      ii. $c_1'' \rightsquigarrow_n^{\sigma'} c_2''$;
      iii. If $\sigma'' \in \phi_{c_2''}$, then $\mathbf{S}_1[\,\textbf{skip}\,] \rightsquigarrow_n^{\sigma''} \mathbf{S}_2[\,\textbf{skip}\,]$;
3. If $\langle c_2, \sigma \rangle \Uparrow$, then either $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$, or $\langle c_1, \sigma \rangle \Uparrow$.
4. If $\langle c_2, \sigma \rangle \underset{\delta_2}{\longrightarrow^*} \kappa_2$, then either $\langle c_1, \sigma \rangle \longrightarrow^* \textsf{abort}$, or there exists δ_1 and κ_1 such that $\langle c_1, \sigma \rangle \underset{\delta_1}{\longrightarrow^*} \kappa_1$ and $\delta_2 \subseteq \delta_1$.

We define $c_1 \rightsquigarrow^\sigma c_2$ as $\forall n.\ c_1 \rightsquigarrow_n^\sigma c_2$. We define $c_1 \rightsquigarrow c_2$ as $\forall \sigma.\ c_1 \rightsquigarrow^\sigma c_2$.

We also need lift cross-rewriting subsumption to thread trees in order to define the relaxed semantics.

Definition 9.3. $\sigma \in \Phi_T$ if, and only if, either

1. $T = c$ and $\sigma \in \phi_c$
2. $T = \langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathbf{p}}\, c$ and $\sigma \in \phi_c$
3. $T = \langle\!\langle T \rangle\!\rangle_{\mathbf{a}}\, c$ and $\sigma \in \phi_c$

Definition 9.4. The binary relation between thread trees $\rightsquigarrow^{\sigma}_{\mathbf{t}}$ is defined as follows:

$$\frac{c_1 \rightsquigarrow^{\sigma} c_2}{c_1 \rightsquigarrow^{\sigma}_{\mathbf{t}} c_2} \qquad \frac{\begin{array}{c} T_1 \rightsquigarrow^{\sigma}_{\mathbf{t}} T_2 \quad T_1' \rightsquigarrow^{\sigma}_{\mathbf{t}} T_2' \\ (\forall \sigma' \in \Phi_{T_2} \cap \Phi_{T_2'}.\ c_1 \rightsquigarrow^{\sigma'} c_2) \end{array}}{\langle\!\langle T_1, T_1' \rangle\!\rangle_{\mathbf{p}} c_1 \rightsquigarrow^{\sigma}_{\mathbf{t}} \langle\!\langle T_2, T_2' \rangle\!\rangle_{\mathbf{p}} c_2} \qquad \frac{\begin{array}{c} T_1 \rightsquigarrow^{\sigma} T_2 \\ (\forall \sigma' \in \Phi_{T_2}.\ c_1 \rightsquigarrow^{\sigma'} c_2) \end{array}}{\langle\!\langle T_1 \rangle\!\rangle_{\mathbf{a}} c_1 \rightsquigarrow^{\sigma}_{\mathbf{t}} \langle\!\langle T_2 \rangle\!\rangle_{\mathbf{a}} c_2}$$

We believe that the properties of subsumption also hold for cross-rewrite subsumption, however we leave it as future work.

9.2 Partial Barriers

Partial barriers, are memory barriers that enforce partial ordering of operations. In the semantics presented in Chapter 3, we can see that atomic blocks work as a total barrier. In this section, we present the semantic extension to support partial barriers. In particular, we are interested in *acquire* and *release* barriers.

An acquire barrier, is used when the footprint of the private memory increases, i.e. the program performs a shared memory operation to acquire some of that memory to its private use. Similarly, a release barrier is used to do the reverse, transfer a portion of private memory to shared memory, decreasing the private footprint. Here we assume acquire barriers are implemented by annotating atomic blocks with a **.a** modifier. Similarly, we use a **.r** for release.

$$\frac{\mathsf{Emp} \vdash \{I{*}P\}\, c\, \{I{*}Q\} \quad P \trianglelefteq Q}{I \vdash \{P\}\, \textbf{atomic.a}\, c\, \{Q\}}\ (\text{ACQUIRE})$$

$$\frac{\mathsf{Emp} \vdash \{I{*}P\}\, c\, \{I{*}Q\} \quad P \trianglerighteq Q}{I \vdash \{P\}\, \textbf{atomic.r}\, c\, \{Q\}}\ (\text{RELEASE})$$

Figure 9.1: CSL inference rules for partial barriers

CSL rules. In Figure 9.1, we present the prototype CSL inference rules for acquire and release. These rules are exactly like the rule presented in Chapter 6, except that we have an extra constraint over the pre- and post-conditions.

The constraint $P \trianglelefteq Q$ ensures the footprint increase or decrease based on the domain of the states described by P and Q.

Definition 9.5. $P \trianglelefteq Q$, if and only if, for all σ and σ', if $\sigma \in P$ and $\sigma' \in Q$, then $\nabla(\sigma) \subseteq \nabla(\sigma')$. We define $P \trianglerighteq Q$ as $Q \trianglelefteq P$.

Subsumption extensions. The following two definitions are used to extend the subsumption relation to support the reordering with regard to partial barriers. We only reorder commands that are purely sequential and completely data independent from the code of the barrier.

Definition 9.6. A command c is sequential if, and only if,

1. $c = a$ or $c = \textbf{skip}$
2. $c = (c_1; c_2)$ or $c = (\textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2)$, where both c_1 and c_2 are sequential
3. $c = (\textbf{while}\ b\ \textbf{do}\ c')$, where c' is also sequential

Definition 9.7. Two commands c_1 and c_2 are independent if, and only if,

1. If $\langle c_1, \sigma\rangle \underset{\delta_1}{\longmapsto}^* \langle c_1', \sigma'\rangle$, and $\langle c_2, \sigma'\rangle \underset{\delta_2}{\longmapsto}^* \kappa$, then $\delta_1 \overset{\frown}{\smile} \delta_2$
2. If $\langle c_2, \sigma\rangle \underset{\delta_2}{\longmapsto}^* \langle c_2', \sigma'\rangle$, and $\langle c_1, \sigma'\rangle \underset{\delta_1}{\longmapsto}^* \kappa$, then $\delta_2 \overset{\frown}{\smile} \delta_1$

In the dynamic semantics, we allow operations to be reordered with atomics following their acquire or release semantics. A new command relation $\underset{\partial}{\rightsquigarrow}$ is defined below

$$\frac{c_1 \rightsquigarrow c_2}{c_1 \underset{\partial}{\rightsquigarrow} c_2}$$

$$\frac{\begin{array}{cc} c_1' \text{ is sequential} & c_1' \text{ and } c_2' \text{ are independent} \\ c_1 \underset{\partial}{\rightsquigarrow} \mathcal{C}[\, c_1'; \textbf{atomic.a}\ c_2' \,] & \mathcal{C}[\, \textbf{atomic.a}\ c_2'; c_1' \,] \underset{\partial}{\rightsquigarrow} c_2 \end{array}}{c_1 \underset{\partial}{\rightsquigarrow} c_2}$$

$$\frac{\begin{array}{cc} c_2' \text{ is sequential} & c_1' \text{ and } c_2' \text{ are independent} \\ c_1 \underset{\partial}{\rightsquigarrow} \mathcal{C}[\, \textbf{atomic.r}\ c_1'; c_2' \,] & \mathcal{C}[\, c_2'; \textbf{atomic.r}\ c_1' \,] \underset{\partial}{\rightsquigarrow} c_2 \end{array}}{c_1 \underset{\partial}{\rightsquigarrow} c_2}$$

In order to establish the soundness of CSL given the extended semantics of partial barriers, one must either extend the definition of DRF-program to account for partial barriers, or establish that the $\underset{\partial}{\rightsquigarrow}$ relation preserves the DRF of a program verified using CSL.

In the first case, we must observe that, differently from other memory models such as Release-Consistency [36] and the Java Memory Model [54], the definition of DRF needs to be extended to consider the **.a** and **.r** modifiers. This is due to the fact that, in our setting, acquire and release can implement arbitrary communication protocols. The technical reason for that cannot be explained trivially but we can give a intuition based on the JMM. In the JMM, a lock operation has acquire semantics, i.e. when we lock an object we gain access to some memory. Naturally, to embed the lock operations in our language we would have to use the **.a** modifier. However, if by mistake we label lock with a **.r** modifier, we might have an operation, that is accessing the memory acquired by the lock, being executed prior to the lock. This "early" access might easily become race, even when the program is DRF according to the current definition. Such effect does not happen in the JMM because the shared-memory operations, such as lock, unlock, and volatile read and write, are built-in operations designed to be compatible with the usual notion of DRF. In our case, we must extend this definition. By extending the definition, we can keep the orthogonality of current soundness proof structure where we establish the soundness with regard to the interleaved semantics and rely solely in the DRF-guarantee to establish the soundness with regard to the relaxed semantics.

In the second case, if we show that the $\underset{\partial}{\rightsquigarrow}$ relation preserves the DRF property of programs verified with CSL, we can establish the soundness of CSL in the presence of partial barriers without the need to redefine the DRF-guarantee. This will produce a mixed soundness proof that will need an auxiliary lemma in the same grounds of Hypothesis 9.8.

Hypothesis 9.8. If $I \vdash \{P\}\, c\, \{Q\}$, $\sigma \in I * P$, and $c \underset{\partial}{\rightsquigarrow} c'$, then $\neg\langle c', \sigma\rangle \longmapsto^{*}$ race

Chapter 10

Related Work and Conclusion

This thesis connects two previously unconnected related fields: memory consistency models and program verification.

The first field, memory consistency models, was usually related to the implementation of shared memory multi-processors, and more recently connected to language semantics to encompass for compiler optimizations as well. Memory model descriptions are traditionally axiomatic and trace-based.

The second field, program verification, involves proving properties of programs, statically, before they execute. Most of the work in program verification uses some form of operational semantics and assume a non-deterministic interleaving to capture concurrency.

It is not hard to see there is a gap between the two fields: first, the way memory models are described makes them harder to reason about as we are used to think of programs semantics as state transitions described operationally; second, when we reason about concurrency as an interleaved execution we neglect many aspects of the execution that might very easily compromise the reasoning.

10.1 Related Work

Here we will present the most significant work, on both memory consistency models and concurrency verification, that inspired this thesis work. We will try be comprehensive and point-out connections as we do the presentation.

10.1.1 Memory Consistency Models

The memory consistency model defines how a machine deals with concurrent memory accesses. Early on, the models were very specific to machine features like write buffers and cache coherence. Later on, more abstract, axiomatic, descriptions appeared. More recently, as the memory model becomes part of language semantics, we have seen attempts to describe them operationally.

Tutorials and Terminology. Adve and Gharachorloo [3] provide a good tutorial describing the relationship between architecture features and hardware memory models. A more recent update to the tutorial is provided by Adve et al. [6].

Mostberger [57] provides a good survey of hardware memory models. Higham et al. [40] and de Melo [23] both present formalizations of memory hardware memory models within the same framework.

The definition of the term *sequential consistency* is due to Lamport [51]. Both the *happens-before* relation and the term *causality* were proposed also by Lamport [50].

Weak Consistency. Adve and Hill [4] describe *weak ordering*, which is also called *weak consistency*. Their work is presented as a refinement of ideas introduced by Dubois et al. [26, 27].

Release Consistency. Gharachorloo et al. [36] present an extension of weak consistency that classifies synchronized accesses further into acquire and release. Their model is referred to as release consistency, and serves as base for describing partial barriers and the DRF-guarantee when such barriers are available. Partial barriers are interesting from the point of view of performance as it gives to the hardware better opportunities for buffering and pipelining.

Java Memory Model. The Java Memory Model (JMM) [54] attempts to draw a line between acceptable and unacceptable behaviors coming from high-level memory models in the context of a typed language. It attempts to define the legal effects of optimizations in the presence of races. It is based on the happens-before model with extra constrains. The happens-before model is clearly undesirable as it allows self-justified executions to happen, breaking causality which is not intuitive for programming language semantics. Most of the effort and complications of the JMM are due to the fact that raceful programs must not break the language safety. The solution is ad-hoc and does not capture some reasonable sequential optimizations. The official description of the JMM was carried out as a Java community process [47].

Aspinall and Sevcik [10, 11] formalized the JMM in Isabelle/HOL and provide a discussion about JMM idiosyncrasies with many examples. Huisman and Petri [43] provide a formalization of the JMM in Coq, also discussing curious aspects of the model.

Pietro et al. [21] present a framework that captures the JMM combining operational, denotational and axiomatic semantic approaches. They separate the memory model specification from the runtime semantics and within the same framework show their interplay. Some examples of optimizations not supported by the JMM are described and possible modifications to the memory model are suggested.

Saraswat et al. [65] provide a theory for memory models based on similar requirements as the JMM. Their framework approaches the problem from the program transformation point of view. Is is not clearly connected to an operational semantics though. The term *fundamental property* is used to refer to the DRF-guarantee.

Itanium/x86 Memory Model. Both Intel and AMD have recently released documents describing their memory models [45, 1] which are informal and ambiguous. It turns out that in practice x86 machines uses a TSO like memory model as advocated by Owens et al. [59]. They provide both an operational and axiomatic model, formalized in HOL4. Their model is much simpler then previous attempts to formalized the x86 memory models [66]. Intel has also released a memory model for the Itanium Architecture [44], which was based in release consistency and described using a less informal framework.

Semantic Approach. Recently, Boudol and Petri [15] presented an operational semantics for a functional language with explicit store buffers. Their semantics have a very concrete feature and therefore captures a very constrained set of relaxations. The presentation is very nice and they prove the DRF-guarantee of their semantics following an approach that is more familiar to the program verification community.

Other Literature. Both Shasha and Snir [67] and Krishnamurthy and Yelick [49] presented early approaches to the problem of memory reordering affecting concurrent execution by finding program cycles when accessing memory.

Hill [41] advocates for hardware implementing sequential consistency to avoid problems with relaxed memory models.

Lipton and Sandberg [53] introduced the PRAM memory model.

Processor consistency was proposed by Goodman [37], and its characteristics are discussed by Ahamad et al. [7].

Ahamad and Hutto [8], introduced *causal consistency* for distributed shared memory and proved the DRF-guarantee of the model.

Modern ideas on hardware memory models where presented by Adve [2] and Gharachorloo [34] in their Ph.D. theses. Gharachorloo et al. [35] present reasoning techniques to program various memory consistency models.

Gao and Sarkar [32] define a hardware memory model that does not rely on cache coherence called *location consistency* which they claim to be weaker than any existing models.

Yang [74] presented a formalization of memory models for program analysis as his Ph.D. thesis. With others he introduced Nemos [77], a parameterized operational framework for describing and analyzing memory models suitable for model checking, and using it to study both Itanium [76] and Java [75] concurrency issues. Park and Dill [60, 25] presented executable specifications for relaxed memory models. Roychoudhury [64] implemented executable specifications to test programs under the JMM and other hardware memory models.

Condon et al. [22] used Lamport clocks to reason about relaxed memory model at the architectural level. Shen et al [68] introduced the *commit-reconcile fences* as a new memory model. Arvind and Maessen [9] present a graph-based framework to define hardware memory models in terms of two aspects: memory ordering and store atomicity.

Adve and Hill [5] provide an unified description for four shared-memory models. Steinke and Nutt [69] also provide an unified theory to describe shared memory consistency.

Boehm [13] discusses the importance of having the memory model as part of the language semantics.

Midkiff et al. [56] present a compiler that produces code for multiple memory models. Sura et al. [70] and Fang et al. [28] study approaches for automatic fence insertion by the compiler.

Burckhardt et al [19] describe a static approach to check the proper use of memory fences in C code.

Bronevetsky and de Supinski [16] provide an operational semantics for the OpenMP memory model.

Boehm and Adve [12] describe the key aspects of the new C++ memory model. Differently from Java, C++ is not a safe language, therefore programs with races have undefined behavior. This simplifies the memory model in a great deal.

10.1.2 Program Verification

Concurrency semantics. The CSL operational semantics [18] assumes sequential consistency. It has explicit race detection, and raceful programs are regarded as unsafe. Vafeiadis and Parkinson [71] present a semantics with explicit memory separation; also avoiding races. Brookes [17] presented a grainless semantics that merges adjacent actions into larger-steps; inspired by earlier work on a concurrent grainless semantics by Reynolds [63]. Hobor et al. [42] presents an oracle semantics for CSL. Gastin and Mislove [33] provide a deterministic operational semantics for a concurrent language with a notion of weak sequential composition, which allows actions that do not share resources to be executed concurrently. Unfortunately, none of these semantics attempted to address objectively memory model issues.

Concurrency verification. Separation logic was developed by Reynolds [62]. CSL [58, 18] was the result of extending separation logic principles to a concurrent setting. The incorporation of permission accounting into CSL to support shared-read only accesses was presented by Bornat et al. [14]. Logics that combine CSL and rely-guarantee reasoning [48, 72] were developed recently [30, 71]. Work on concurrency verification of assembly code using rely-guarantee is available [79, 31] as extensions of *certified assembly programing* (CAP) [39, 78].

Program equivalence. Leroy [52] introduces some intuitive notions of program equivalence required to define the correctness of an optimizing compiler. Calcagno and O'Hearn [20] define a semantic hoare triple that captures programs for which some notion of equivalence can be derived. The command subsumption relation was inspired and can be related by these work.

10.2 Conclusion

In this thesis, we have presented a simple operational semantics to formalize memory models. First, we present a footprint sequential semantics and and interleaved concurrent semantics which are used to define sequential consistency and what we regard as race-free programs. The semantics allow free nesting of atomic blocks and parallel composition. Furthermore, it is a small-step semantics that exhibits weak atomicity behaviors where unprotected code may interleave with atomic block execution. Then, we present a very simple semantics which is parameterized on a binary relation over programs. This semantics is based on the interleaved semantics but chooses any program from the binary relation to execute, which in fact simulates the execution of any of them. This binary relation should be seen as a compiler or hardware that is optimizing the program on the fly. Naturally, for a given parameter, i.e. a given set of program transformations, we will obtain a given memory model.

By instantiating the parameter with a specific relation, which we call subsumption ($\rightsquigarrow$), we have obtained a particular memory model that is weaker than many existing ones. This memory model also captures many optimizations that maintain some notion of sequential equivalence, and preserve the displacement of atomic blocks in the code. We show that the relaxed semantics obtained from subsumption has the DRF-guarantee. This means that programs that originally are absent of data-races, according to the interleaved semantics, will not exhibit new behaviors when optimized. This proof uses an intermediate mixed-step semantics that executes sequential code in big-steps, making a very intuitive connection between the interleaved semantics and the subsumption definition.

Finally, we prove the soundness of CSL and extensions with regard to the relaxed semantics. This is simply done by first establishing the soundness with regard to the interleaved semantics, and later using the DRF-guarantee theorem. However, in order to use the DRF-guarantee, the CSL semantic model must also prevent races, which is an extra condition carried over the soundness proof. In practical terms, we observe that as long as we properly synchronize the program, such that concurrent allocation and deallocation are always protected by an atomic block, CSL is indeed race-free.

Appendix A

Notation Summary

Syntactic Category	Description	Where	Page
c	command	Fig. 3.1	18
a	action	Fig. 3.1	18
ν	l-value	Fig. 3.1	18
e	expression	Fig. 3.1	18
b	condition	Fig. 3.1	18
κ	program configuration	Fig. 3.3	20
σ	program state	Fig. 3.3	20
$\bar{\sigma}$	program state with permissions	Fig. 7.1	171
Π	state permissions	Fig. 7.1	171
π	location permission	Fig. 7.1	171
ℓ	memory location	Fig. 3.3	20
T	thread tree	Fig. 3.3	20
δ	memory footprint	Fig. 3.4	22
rs/ws	read/write set of locations	Fig. 3.4	22
$\mathcal{C}$	program context	Sec. 3.4	23
$\mathbf{S}$	sequential context	Fig. 3.6	23
$\mathbf{T}$	thread three context	Fig. 3.6	23
Λ	command transformation	Sec. 3.8	42

Notation	Description	Where	Page
$\langle T, \sigma \rangle$	normal program configuration	Fig. 3.3	20
abort	abort program configuration	Fig. 3.3	20
race	race program configuration	Fig. 3.3	20
$\langle\!\langle T_1, T_2 \rangle\!\rangle_{\mathsf{p}}\, c$	thread tree node for parallel composition	Fig. 3.3	20
$\langle\!\langle T \rangle\!\rangle_{\mathsf{a}}\, c$	thread tree node for atomic block	Fig. 3.3	20
$\bullet$	program hole	Sec. 3.4	23

Notation	Description	Where	Page
T is n-atomic	thread tree class	Def. 3.1	21
$\mathbf{T}$ is n-atomic	thread tree context class	Def. 3.3	24
emp	empty footprint	Fig. 3.5	22
$\delta_1 \cup \delta_2$	footprint union	Fig. 3.5	22
$\delta_1 \subseteq \delta_2$	sub-footprint relation	Fig. 3.5	22
$\delta_1 \equiv \delta_2$	footprint equivalence	Fig. 3.5	22
$\delta_1 \subset \delta_2$	proper sub-footprint relation	Fig. 3.5	22
$\delta_1 \overset{\rightarrow}{\smile} \delta_2$	unidirectional footprint non-interference	Fig. 3.5	22
$\delta_1 \smile \delta_2$	bidirectional footprint non-interference	Fig. 3.5	22
$\bar{\sigma}_1 \uplus \bar{\sigma}_2$	disjoint union of states with permissions	Def. 7.1	172
$\bar{\sigma}_1 \subseteq \bar{\sigma}_2$	subs Tate relation for states with permissions	Def. 7.1	172
$\bar{\sigma}\|_{\Pi}$	permissions extraction	Def. 7.2	172
$\mathcal{C}[\,-\,]$	program context substitution	Sec. 3.4	23
$\mathbf{S}[\,-\,]$	sequential context substitution	Sec. 3.4	23
$\mathbf{T}[\,-\,]$	thread tree context substitution	Sec. 3.4	23
$[\![\nu]\!]$	l-value interpretation	Fig. 3.7	25
$[\![e]\!]$	expression interpretation	Fig. 3.7	25
$[\![b]\!]$	condition interpretation	Fig. 3.7	25
$[\![a]\!]$	action interpretation	Fig. 3.7	25
$\mathsf{L}_{\sigma}^{\nu}$	l-value location set	Fig. 3.8	26
Δ_{σ}^{ν}	l-value footprint	Fig. 3.8	26
Δ_{σ}^{e}	expression footprint	Fig. 3.8	26
Δ_{σ}^{b}	condition footprint	Fig. 3.8	26
Δ_{σ}^{a}	action footprint	Fig. 3.8	26
$\nabla(\sigma)$	largest compatible footprint for state	Def. 3.4	27
$\nabla(\bar{\sigma})$	largest compatible footprint for state with permissions	Def. 7.3	172
$\lfloor\Lambda\rfloor$	lift transformation from command to tree	Def. 3.23	43
Λ provides the DRF-guarantee	DRF-guarantee property	Def. 5.2	91

Notation	Description	Where	Page
$\kappa \longrightarrow \kappa'$	sequential small-step	Fig. 3.9	28
$\kappa \underset{\delta}{\longrightarrow} \kappa'$	sequential small-step with footprint	Fig. 3.10	30
$\kappa \underset{\delta}{\longrightarrow}^{n} \kappa'$	sequential multi-step with footprint	Def. 3.7	32
$\kappa \underset{\delta}{\longrightarrow}^{*} \kappa'$	reflexive transitive closure of step with footprint	Def. 3.7	32
$\kappa \longmapsto \kappa'$	concurrent small-step	Fig. 3.11	36
$\kappa \longmapsto^{\infty}$	concurrent small-step divergence	Def. 3.19	39
$\kappa \underset{\delta}{\longmapsto}^{n} \kappa'$	concurrent multi-step with footprint	Def. 3.20	39
$\kappa \underset{\delta}{\longmapsto}^{*} \kappa'$	reflexive transitive closure of step with footprint	Def. 3.20	39
$[\Lambda]\ \kappa \longmapsto \kappa'$	parameterized small-step	Fig. 3.13	42
$[\Lambda]\ \kappa \longmapsto^{n} \kappa'$	parameterized multi-step	Def. 3.24	43
$[\Lambda]\ \kappa \longmapsto^{*} \kappa'$	reflexive transitive closure of parameterized step	Def. 3.24	43
$[\Lambda]\ \kappa \longmapsto^{\infty}$	parameterized small-step divergence	Def. 3.25	43
$\kappa \Downarrow \kappa'$	sequential evaluation	Fig. 3.14	45
$\kappa \Uparrow$	sequential divergence	Fig. 3.14	45
$c_1 \leadsto_n c_2$	indexed command subsumption	Def. 3.33	47
$c_1 \leadsto c_2$	command subsumption	Def. 3.33	47
$T_1 \leadsto_{\mathsf{t}} T_2$	thread tree subsumption	Def. 3.33	47
$\kappa \Longrightarrow \kappa'$	concurrent mixed-step	Fig. 5.1	92
$\kappa \Longrightarrow^{\infty}$	concurrent mixed-step divergence	Def. 5.3	92

Syntactic Category	Description	Where	Page
P, Q, I	assertions	Fig. 6.1/Fig. 7.2	128/173
$\mathcal{P}, \mathcal{Q}, \mathcal{I}$	assertion formulae	Fig. 6.1/Fig. 7.2	128/173
A, G	binary assertions	Fig. 8.1	217

Notation	Description	Where	Page
$\lfloor b \rfloor$	condition lifted to assertion	Fig. 6.2/Fig. 7.3	129/173
$P_1 * P_2$	separating conjunction for assertions	Fig. 6.2/Fig. 7.3	129/173
$G_1 * G_2$	separating conjunction for binary assertions	Fig. 8.2	218
$P \times Q$	cartesian product of assertions	Fig. 8.2	218
P^2	cartesian product with itself	Fig. 8.2	218
$P \circ G$	assertion composition	Fig. 8.2	218
Emp	assertion for empty state	Fig. 6.2/Fig. 7.3	129/173
Id	identity binary assertion	Fig. 8.2	218
$\ell \mapsto i$	assertion for singleton state	Fig. 6.2/Fig. 7.3	129/173
$\ell \overset{\pi}{\mapsto} i$	assertion for singleton state with permission	Fig. 7.3	173
P is precise	precision for assertions	Def. 6.1/Def. 7.4	129/173
$Q \circ [\![a]\!]$	weakest pre-condition for action	Def. 6.2/Def. 7.5	131/175
P_1 and P_2 coincide	coincidence for assertions	Def. 6.35	165

Notation	Description	Where	Page
$I \vdash \{P\}\, c\, \{Q\}$	CSL judgement	Fig. 6.3/Fig. 7.4	130/174
$A, G \vdash \{P_s, P\}\, c\, \{Q_s, Q\}$	SAGL judgement	Fig. 8.3	219
$I \models \langle T, \sigma \rangle \triangleright_n Q$	indexed CSL semantic triple	Def. 6.3	132
$I \models \langle T, \sigma \rangle \triangleright Q$	CSL semantic triple	Def. 6.3	132
$I \models \langle T, \bar{\sigma} \rangle \triangleright_n Q$	indexed CSL with permissions semantic triple	Def. 7.8	175
$I \models \langle T, \bar{\sigma} \rangle \triangleright Q$	CSL with permissions semantic triple	Def. 7.8	175
$A, G, P_s, Q_s \models \langle T, \sigma \rangle \triangleright_n Q$	indexed SAGL semantic sixtuple	Def. 8.1	221
$A, G, P_s, Q_s \models \langle T, \sigma \rangle \triangleright Q$	SAGL semantic sixtuple	Def. 8.1	221
$I \models \{P\}\, c\, \{Q\}$	CSL semantic model	Def. 6.15/Def. 7.20	152/197
$A, G \models \{P_s, P\}\, c\, \{Q_s, Q\}$	SAGL semantic model	Def. 8.13	243
$I \models_{[\Lambda]} \{P\}\, c\, \{Q\}$	parameterized CSL semantic model	Def. 6.32/Def. 7.37	163/207
$A, G \models_{[\Lambda]} \{P_s, P\}\, c\, \{Q_s, Q\}$	parameterized SAGL semantic model	Def. 8.30	254

Appendix B

Guide to Coq Proofs

In this appendix, we provide a high-level guide to the Coq implementation of the theory presented in this thesis. We have formalized the theory for the language syntax and semantics (Chapter 3), the proof of the DRF-guarantee (Chapter 5), and the soundness proofs for both CSL (Chapter 6) and SAGL (Chapter 8).

B.1 Organization

The Coq proof is organized into several files. Here we present the file name and a description of its contents:

- `cntx.v`: sequential context definitions, on page 288
- `csl.v`: CSL inference rules, on page 308
- `csl_in_sagl.v`: Embedding of CSL in SAGL, on page 321
- `csl_soundness.v`: CSL soundness proof, on page 310
- `deltasem.v`: definition of footprint functions, on page 294
- `denosem.v`: semantics of expressions and actions, on page 295
- `foot.v`: footprint definitions, on page 291

B.2 Contents

In this section, we present, for each Coq file, a summary of important definitions and associated theory. We only describe here the most relevant parts of the implementation in order to keep the presentation concise. Whenever possible, we will provide pointers to their counterparts as introduced in the preceding chapters.

`set.v`

In this file, we define a library of operations and properties to manipulate a finite set of elements of polymorphic type `A`. Here we present the opaque interface to this library, the actual implementation uses a sorted list as encoding for the finite set.

```
Section set.

(* requirements over polymorphic type A *)
Variable A: Set.
Variable cmp_A: A -> A -> comparison.
Variable cmp_A_equ: forall a1 a2, a1 = a2 <-> cmp_A a1 a2 = Eq.
Variable cmp_A_rev: forall a1 a2, CompOpp (cmp_A a1 a2) = cmp_A a2 a1.

(* sample definitions *)
Definition set: Set. (* set type *)
Definition emp: set. (* empty set *)
Definition sin (a: A): set. (* singleton set *)
Definition ins (s: set) (a : A): set. (* insertion *)
Definition del (s: set) (a : A): set. (* deletion *)
Definition has (s: set) (a : A): bool. (* membership check *)
Definition cup (s1 s2: set): set. (* union *)
Definition cap (s1 s2: set): set. (* intersection *)
Definition min (s1 s2: set): set. (* subtraction *)
Definition dis (s1 s2: set): Prop. (* disjoint relation *)
Definition sub (s1 s2: set): Prop. (* subset relation *)

(* sample properties *)
Lemma cup_ident: forall s, cup emp s = s.
Lemma cup_equal: forall s, cup s s = s.
Lemma cup_commu: forall s1 s2, cup s1 s2 = cup s2 s1.
Lemma cup_assoc: forall s1 s2 s3,
                 cup (cup s1 s2) s3 = cup s1 (cup s2 s3).
Lemma cap_empty: forall s, cap emp s = emp.
Lemma cap_equal: forall s, cap s s = s.
Lemma cap_commu: forall s1 s2, cap s1 s2 = cap s2 s1.
Lemma cap_assoc: forall s1 s2 s3,
                 cap (cap s1 s2) s3 = cap s1 (cap s2 s3).
Lemma dis_empty: forall s, dis emp s.
Lemma dis_commu: forall s1 s2, dis s1 s2 -> dis s2 s1.
Lemma sub_empty: forall s, sub emp s.
Lemma sub_reflx: forall s, sub s s.
Lemma sub_trans: forall s1 s2 s3, sub s1 s2 -> sub s2 s3 -> sub s1 s3.
Lemma sub_cup: forall s1 s2 s1' s2',
               sub s1 s1' -> sub s2 s2' ->
               sub (cup s1 s2) (cup s1' s2').
Lemma sub_cup2: forall s1 s2, sub s1 (cup s1 s2).
Lemma sub_dis: forall s1 s2 s1' s2',
               sub s1 s1' -> sub s2 s2' -> dis s1' s2' -> dis s1 s2.
Lemma sub_min: forall s1 s2, sub (min s1 s2) s1.

End set.
```

---

map.v

In this file, we define a library of operations and properties to manipulate a finite partial map of elements of polymorphic type `A` to elements of polymorphic type `B`. Here we present the opaque interface to this library, the actual implementation uses a sorted list as encoding for the finite map.

```
Section map.

(* requirements over polymorphic types A and B *)
Variable A B: Set.
Variable cmp_A: A -> A -> comparison.
Variable cmp_A_equ: forall a1 a2, a1 = a2 <-> cmp_A a1 a2 = Eq.
Variable cmp_A_rev: forall a1 a2, CompOpp (cmp_A a1 a2) = cmp_A a2 a1.
Variable uni_B: B -> B -> B.
Variable uni_B_commu: forall b1 b2, uni_B b1 b2 = uni_B b2 b1.
Variable dis_B: B -> B -> Prop.
Variable dis_B_commu: forall b1 b2, dis_B b1 b2 -> dis_B b2 b1.

(* sample definitions *)
Definition map: Set. (* map type *)
Definition emp: map. (* empty map *)
Definition sin (a: A) (b: B): map. (* singleton map *)
Definition ins (m: map) (a: A) (b: B): map. (* insertion *)
Definition del (m: map) (a: A): map. (* deletion *)
Definition get (m: map) (a: A): option B. (* retrieval *).
Definition has (m: map) (a: A): bool. (* membership check *)
Definition dom (m: map): set A. (* domain extraction *)
Definition uni (m1 m2: map): map. (* union *)
Definition dis (m1 m2: map): Prop. (* disjoint relation *)

(* sample properties *)
Lemma uni_ident: forall m, uni emp m = m.
Lemma uni_commu: forall m1 m2, uni m1 m2 = uni m2 m1.
Lemma uni_assoc: forall m1 m2 m3,
                 uni (uni m1 m2) m3 = uni m1 (uni m2 m3).
Lemma dis_empty: forall m, dis emp m.
Lemma dis_commu: forall m1 m2, dis m1 m2 -> dis m2 m1.

End map.
```

---

lang.v

In this file, we define the syntax of our programming language as close as possible to the one presented in Fig. 3.1. There is only one simplification: instead of $\nu := \textbf{cons}(e_1, \ldots, e_n)$, that takes a arbitrary number of arguments and initialize each new memory location with the result of evaluation of the respective argument, we define a simpler primitive that only allocates a pair of uninitialized memory cells.

```
Inductive Expr: Set := (* expressions *)
  | econ (n: nat)
  | elod (lv: Lval)
  | eadd (e1 e2: Expr)
  | eneg (e: Expr)
with Lval: Set := (* l-values *)
  | lvar (v: string)
  | lmem (e: Expr).

Inductive Cond: Set := (* conditions *)
  | bcon (z: bool)
  | band (b1 b2: Cond)
  | bneg (b: Cond)
  | bequ (e1 e2: Expr)
  | bltn (e1 e2: Expr).

Inductive Actn: Set := (* actions *)
  | asgn (lv: Lval) (e: Expr)
  | allc (lv: Lval)
  | free (lv: Lval).

Inductive Comm: Set := (* commands *)
  | actn (a: Actn)
  | sequ (c1 c2: Comm)
  | skip
  | cond (b: Cond) (c1 c2: Comm)
  | loop (b: Cond) (c: Comm)
  | para (c1 c2: Comm)
  | atom (c: Comm).
```

In this file, we also define the syntactic sugar presented in Fig. 3.2.

```
Definition esub (e1 e2: Expr): Expr := eadd e1 (eneg e2).
Definition bor (b1 b2: Cond): Cond := bneg (band (bneg b1) (bneg b2)).
Definition bneq (e1 e2: Expr): Cond := bneg (bequ e1 e2).
Definition bleq (e1 e2: Expr): Cond := bltn e1 (eadd e2 (econ 1)).
Definition bgeq (e1 e2: Expr): Cond := bneg (bltn e1 e2).
Definition bgtn (e1 e2: Expr): Cond :=
  bneg (bltn e1 (eadd e2 (econ 1))).
Definition pcnd (b: Cond) (c: Comm): Comm := cond b c skip.
Definition repe (c: Comm) (b: Cond): Comm := sequ c (loop (bneg b) c).
Definition forv (lv: Lval) (e1 e2: Expr) (c: Comm): Comm :=
  sequ (actn (asgn lv e1))
       (loop (bltn (elod lv) (eadd e2 (econ 1)))
             (sequ c (actn (asgn lv (eadd (elod lv) (econ 1)))))).
Definition wait (b: Cond): Comm := loop (bneg b) skip.
Definition when (b: Cond) (c: Comm): Comm :=
  atom (sequ (loop (bneg b) skip) c).
```

`runtime.v`

In this file, we define thread trees and program configurations as shown in Fig. 3.3.

```
Inductive Tree: Set := (* thread trees *)
  | tcomm (c: Comm)
  | tpara (t1 t2: Tree) (c: Comm)
  | tatom (t: Tree) (c: Comm).

Inductive Conf : Set := (* program configurations *)
  | conf (t: Tree) (s: State)
  | abrt
  | race.
```

We also define a predicate to characterize n-atomic thread trees (Def. 3.1).

```
Inductive ATOM : nat -> Tree -> Prop := (* n-atomic predicate *)
  | ATOM_tcomm : forall c, ATOM 0 (tcomm c)
  | ATOM_tatom : forall t c, ATOM 1 (tatom t c)
  | ATOM_tpara : forall n1 n2 t1 t2 c,
                 ATOM n1 t1 ->
                 ATOM n2 t2 ->
                 ATOM (n1+n2) (tpara t1 t2 c).
```

`cntx.v`

In this file, we define command contexts and associated properties. We also define command transformations as a relation.

```
(* command transformations *)
Definition Trfrm: Type := Comm -> Comm -> Prop.
```

```
Inductive Cntx: Set := (* command contexts *)
  | hole
  | sequ_1 (C: Cntx) (c: Comm)
  | sequ_2 (c: Comm) (C: Cntx)
  | cond_1 (b: Cond) (C: Cntx) (c: Comm)
  | cond_2 (b: Cond) (c: Comm) (C: Cntx)
  | loop_1 (b: Cond) (C: Cntx)
  | para_1 (C: Cntx) (c: Comm)
  | para_2 (c: Comm) (C: Cntx)
  | atom_1 (C: Cntx).

(* context/command substitution *)
Fixpoint Subst (C: Cntx) (c': Comm): Comm :=
  match C with
  | hole => c'
  | sequ_1 C c => sequ (Subst C c') c
  | sequ_2 c C => sequ c (Subst C c')
  | cond_1 b C c => cond b (Subst C c') c
  | cond_2 b c C => cond b c (Subst C c')
  | loop_1 b C => loop b (Subst C c')
  | para_1 C c => para (Subst C c') c
  | para_2 c C => para c (Subst C c')
  | atom_1 C => atom (Subst C c')
  end.
(* context/context substitution *)
Fixpoint Substx (C: Cntx) (C': Cntx): Cntx :=
  match C with
  | hole => C'
  | sequ_1 C c => sequ_1 (Substx C C') c
  | sequ_2 c C => sequ_2 c (Substx C C')
  | cond_1 b C c => cond_1 b (Substx C C') c
  | cond_2 b c C => cond_2 b c (Substx C C')
  | loop_1 b C => loop_1 b (Substx C C')
  | para_1 C c => para_1 (Substx C C') c
  | para_2 c C => para_2 c (Substx C C')
  | atom_1 C => atom_1 (Substx C C')
  end.
Fixpoint Flat (C: Cntx): Comm := (* context flattening *)
  match C with
  | hole => skip
  | sequ_1 hole c => c
  | sequ_1 C' c => sequ (Flat C') c
  | sequ_2 c hole => c
  | sequ_2 c C' => sequ c (Flat C')
  | cond_1 b C' c => cond b (Flat C') c
  | cond_2 b c C' => cond b c (Flat C')
  | loop_1 b C' => loop b (Flat C')
  | para_1 C' c => para (Flat C') c
  | para_2 c C' => para c (Flat C')
  | atom_1 C' => atom (Flat C')
  end.
```

The following predicate, characterizes sequential contexts as defined in Fig 3.6.

```
Inductive SEQU : Cntx -> Prop := (* sequential contexts *)
  | SEQU_hole : SEQU hole
  | SEQU_sequ : forall S c, SEQU S -> SEQU (sequ_1 S c).
```

We define the weaker notion of command equality, used to establish Lemma 3.46, that allows a command to be considered equal to itself prepended with **skip**.

```
Inductive c_eq_n: nat -> Comm -> Comm -> Prop :=
  | c_eq_O: forall c, c_eq_n O c c
  | c_eq_S1: forall n c1 c2,
             (exists S, SEQU S /\
              c1 = Subst S skip /\ c_eq_n n (Flat S) c2) ->
             c_eq_n (S n) c1 c2.
  | c_eq_S2: forall n c1 c2,
             (exists S, SEQU S /\
              c2 = Subst S skip /\ c_eq_n n c1 (Flat S)) ->
             c_eq_n (S n) c1 c2.
Definition c_eq (c1 c2: Comm): Prop := exists n, c_eq_n n c1 c2.
```

We also define a generalized proof of contextualization that takes a relation `R` as parameter. This is the basis for the proof of Lemma 3.36.

```
Theorem CntxEquiv:
  forall (R: Trfrm) C c1 c2,
  (forall c1 c2 c, R c1 c2 -> R (sequ c1 c) (sequ c2 c)) ->
  (forall c1 c2 c, R c1 c2 -> R (sequ c c1) (sequ c c2)) ->
  (forall c1 c2 b c, R c1 c2 -> R (cond b c1 c) (cond b c2 c)) ->
  (forall c1 c2 b c, R c1 c2 -> R (cond b c c1) (cond b c c2)) ->
  (forall c1 c2 b, R c1 c2 -> R (loop b c1) (loop b c2)) ->
  (forall c1 c2 c, R c1 c2 -> R (para c1 c) (para c2 c)) ->
  (forall c1 c2 c, R c1 c2 -> R (para c c1) (para c c2)) ->
  (forall c1 c2, R c1 c2 -> R (atom c1) (atom c2)) ->
  R c1 c2 ->
  R (Subst C c1) (Subst C c2).
```

---

loc.v

In this file, we define locations as shown in Fig. 3.3. We also define operations for comparison of locations, as well as helper injection functions.

```
Definition Loc: Set := (Z + string)%type. (* locations *)

Definition beq (l1 l2: Loc): bool := (* equality *)
  match l1,l2 with
  | inl i1,inl i2 => Zeq_bool i1 i2
  | inr v1,inr v2 => prefix v1 v2 && prefix v2 v1
  | _,_ => false
  end.
Definition cmp (l1 l2: Loc): comparison := (* comparison *)
  match l1,l2 with
  | inl i,inr v => Lt
  | inl i1,inl i2 => Zcompare i1 i2
  | inr v1,inr v2 => cmp_string v1 v2
  | inr v,inl i => Gt
  end.

Definition inl (i: Z): Loc := inl string i. (* left injection *)
Definition inr (v: string): Loc := inr Z v. (* right injection *)
```

---

locset.v

In this file, we conveniently define a location set as a specialization of a set as defined in file `set.v`.

```
(* common definitions *)
Definition LocSet: Set := set Loc. (* location set type *)
```

---

foot.v

In this file, we define a footprint as show in Fig 3.4, and the associated definitions as shown in Fig 3.5.

```
(* common definitions *)
Definition Foot: Set := (LocSet * LocSet)%type. (* footprint type *)
Definition emp: Foot := (locset.emp,locset.emp). (* empty footprint *)
Definition cup (f1 f2: Foot): Foot := (* footprint union *)
  match f1,f2 with
  | (rs1,ws1),(rs2,ws2) => (locset.cup rs1 rs2,locset.cup ws1 ws2)
  end.
Definition sub (f1 f2: Foot): Prop := (* sub-footprint relation *)
  match f1,f2 with
  | (rs1,ws1),(rs2,ws2) => locset.sub rs1 (locset.cup rs2 ws2)
                        /\ locset.sub ws1 ws2
  end.
(* unidirectional non-interference *)
Definition rni (f1 f2: Foot): Prop :=
  match f1,f2 with
  | (rs1,ws1),(rs2,ws2) => locset.dis ws1 (locset.cup rs2 ws2)
  end.
(* bidirectional non-interference *)
Definition ni (f1 f2: Foot): Prop :=
  rni f1 f2 /\ rni f2 f1.
```

```
(* common properties *)
Lemma cup_ident: forall f, cup emp f = f.
Lemma cup_equal: forall f, cup f f = f.
Lemma cup_commu: forall f1 f2, cup f1 f2 = cup f2 f1.
Lemma cup_assoc: forall f1 f2 f3,
                 cup (cup f1 f2) f3 = cup f1 (cup f2 f3).
Lemma sub_empty: forall f, sub emp f.
Lemma sub_reflx: forall f, sub f f.
Lemma sub_trans: forall f1 f2 f3, sub f1 f2 -> sub f2 f3 -> sub f1 f3.
Lemma sub_cup: forall f1 f2 f1' f2',
               sub f1 f1' -> sub f2 f2' ->
               sub (cup f1 f2) (cup f1' f2').
Lemma sub_cup2: forall f1 f2, sub f1 (cup f1 f2).
Lemma rni_dec: forall f1 f2, rni f1 f2 \/ ~rni f1 f2.
Lemma rni_sub: forall f1 f1' f2 f2',
               sub f1 f1' -> sub f2 f2' -> rni f1' f2' -> rni f1 f2.
Lemma rni_cup_left: forall f1 f2 f,
                    rni f1 f -> rni f2 f -> rni (cup f1 f2) f.
Lemma rni_cup_rght: forall f1 f2 f,
                    rni f f1 -> rni f f2 -> rni f (cup f1 f2).
Lemma ni_emp: forall f, ni f emp.
```

state.v

In this file, we define the program state as a finite partial map from locations to integers (as shown in Fig 3.3). We also define the largest compatible footprint of a state (Def. 3.4) and associated properties.

```
(* common definitions *)
Definition State: Set := map Loc Z. (* state type *)
(* splitting *)
Definition spl (s s1 s2: State): Prop := s = uni s1 s2 /\ dis s1 s2.
(* largest compatible footprint *)
Definition footstate (s: State): Foot := (locset.emp,dom s).

(* common properties *)
Lemma spl_commu: forall s s1 s2, spl s s1 s2 -> spl s s2 s1.
Lemma spl_assoc: forall s s1 s2 s3,
                 (exists s23, spl s s1 s23 /\ spl s23 s2 s3) ->
                 (exists s12, spl s s12 s3 /\ spl s12 s1 s2).
Lemma spl_det2: forall s s1 s2 s2',
                spl s s1 s2 -> spl s s1 s2' -> s2 = s2'.
Lemma spl_emp: forall s, spl s emp s.
Lemma spl_emp_det: forall s s', spl s emp s' -> s = s'.
Lemma dom_footstate: forall s s',
                     dom s = dom s' -> footstate s = footstate s'.
Lemma rni_footstate: forall s s',
                     locset.dis (dom s) (dom s') ->
```

```
                    rni (footstate s) (footstate s').
```

---

multi.v

In this file, we define multi-step, evaluation, and divergence, in a general way for both labeled and unlabeled transitions.

```
Definition Strel: Type := (* unlabeled transition type *)
  Conf -> Conf -> Prop.
Definition LbStrel: Type :=  (* labeled transition type *)
  Conf -> Foot -> Conf -> Prop.
Definition Lb (Step: Strel): LbStrel := (* lifting *)
  fun k l k' => Step k k'.
Definition Un (Step: LbStrel): Strel := (* flattening *)
  fun k k' => exists l, Step k l k'.

(* labeled transition definitions *)
Section lb.

(* polymorphic labeled stepping *)
Variable Step: LbStrel.

Inductive LbMultin: nat -> LbStrel := (* fixed size multi-step *)
  | LbMultin_O: forall k,
                LbMultin O k emp k
  | LbMultin_S: forall n k l1 l2 k',
                (exists k'', Step k l1 k'' /\ LbMultin n k'' l2 k') ->
                LbMultin (S n) k (cup l1 l2) k'.
Definition LbMulti: LbStrel := (* arbitrary size multi-step *)
  fun k l k' => exists n, LbMultin n k l k'.
Definition LbEvalu: LbStrel := (* evaluation *)
  fun k l k' => LbMulti k l k'
             /\ ~exists l', exists k'', Step k' l' k''.
Definition LbDiver: Conf -> Prop := (* divergence *)
  fun k => forall n, exists l, exists k', LbMultin n k l k'.

End lb.

(* unlabeled transition definitions *)
Section un.

(* polymorphic unlabeled stepping *)
Variable Step: Strel.

Definition Multin: nat -> Strel := (* fixed size multi-step *)
  fun n => Un (LbMultin (Lb Step) n).
Definition Multi: Strel := (* arbitrary size multi-step *)
  Un (LbMulti (Lb Step)).
Definition Evalu: Strel := (* evaluation *)
  Un (LbEvalu (Lb Step)).
```

```
Definition Diver: Conf -> Prop := (* divergence *)
  LbDiver (Lb Step).

End un.
```

---

deltasem.v

In this file, we define the footprint extraction functions as shown in Fig. 3.8.

```
Definition L_l (lv: Lval) (s: State): LocSet :=
  match Deno_l lv s with
  | Some l => locset.sin l
  | None => locset.emp
  end.
Fixpoint Delta_e (e: Expr) (s: State): Foot :=
  match e with
  | econ i => emp
  | elod lv => cup (Delta_l lv s) (L_l lv s,locset.emp)
  | eadd e1 e2 => cup (Delta_e e1 s) (Delta_e e2 s)
  | eneg e => Delta_e e s
  end
with Delta_l (lv: Lval) (s: State): Foot :=
  match lv with
  | lvar v => emp
  | lmem e => Delta_e e s
  end.
Fixpoint Delta_b (b: Cond) (s: State): Foot :=
  match b with
  | bcon z => emp
  | band b1 b2 => cup (Delta_b b1 s) (Delta_b b2 s)
  | bneg b => Delta_b b s
  | bequ e1 e2 => cup (Delta_e e1 s) (Delta_e e2 s)
  | bltn e1 e2 => cup (Delta_e e1 s) (Delta_e e2 s)
  end.
Fixpoint Delta_a (a: Actn) (s: State): Foot :=
  match a with
  | asgn lv e => cup (Delta_l lv s)
                     (cup (Delta_e e s) (locset.emp,L_l lv s))
  | allc lv => cup (Delta_l lv s) (locset.emp,L_l lv s)
  | free lv => cup (Delta_l lv s) (locset.emp,L_l lv s)
  end.
```

We also establish the framing properties of the footprint extraction functions as stated by Remark 3.15 and Remark 3.14, respectively.

```
Lemma Delta_b_framing: forall b s s1 s2,
                       spl s s1 s2 ->
                       (exists z, Deno_b b s2 = Some z) ->
                       Delta_b b s = Delta_b b s2.
```

```
Lemma Delta_a_framing: forall a s s1 s2,
                       spl s s1 s2 ->
                       (exists s2', Deno_a a s2 s2') ->
                       Delta_a a s = Delta_a a s2.
```

`denosem.v`

In this file, we define the semantics of l-values, expressions, conditions, and actions, as shown in Fig. 3.7.

```
Fixpoint Deno_e (e: Expr) (s: State): option Z :=
  match e with
  | econ n => Some (Z_of_nat n)
  | elod lv => match Deno_l lv s with
               | Some l => get s l
               | None => None
               end
  | eadd e1 e2 => match Deno_e e1 s,Deno_e e2 s with
                  | Some i1,Some i2 => Some (i1+i2)%Z
                  | _,_ => None
                  end
  | eneg e => match Deno_e e s with
              | Some i => Some (-i)%Z
              | None => None
              end
  end
with Deno_l (lv: Lval) (s: State): option Loc :=
  match lv with
  | lvar v => Some (inr v)
  | lmem e => match Deno_e e s with
              | Some i => Some (inl i)
              | None => None
              end
  end.
Fixpoint Deno_b (b: Cond) (s: State): option bool :=
  match b with
  | bcon z => Some z
  | band b1 b2 => match Deno_b b1 s,Deno_b b2 s with
                  | Some z1,Some z2 => Some (z1 && z2)
                  | _,_ => None
                  end
  | bneg b => match Deno_b b s with
              | Some z => Some (negb z)
              | None => None
              end
  | bequ e1 e2 => match Deno_e e1 s,Deno_e e2 s with
                  | Some i1,Some i2 => Some (Zeq_bool i1 i2)
                  | _,_ => None
                  end
  | bltn e1 e2 => match Deno_e e1 s,Deno_e e2 s with
```

```
                  | Some i1,Some i2 => Some (Zlt_bool i1 i2)
                  | _,_ => None
                  end
  end.
Inductive Deno_a: Actn -> State -> State -> Prop :=
  | Deno_asgn: forall lv e s l i,
               Deno_l lv s = Some l ->
               (exists i, get s l = Some i) ->
               Deno_e e s = Some i ->
               Deno_a (asgn lv e) s (ins s l i)
  | Deno_allc: forall lv s l p i1 i2,
               Deno_l lv s = Some l ->
               (exists i, get s l = Some i) ->
               p > 0 ->
               get s (inl (Z_of_nat p)) = None ->
               get s (inl (Z_of_nat (p+1))) = None ->
               Deno_a (allc lv) s
                 (ins (ins (ins s
                   (inl (Z_of_nat (p+1))) i2)
                   (inl (Z_of_nat p)) i1) l (Z_of_nat p))
  | Deno_free: forall lv s l,
               Deno_l lv s = Some l ->
               (exists i, get s l = Some i) ->
               Deno_a (free lv) s (del s l).
```

We also establish Remark 3.13, as well as the framing properties stated by Remark 3.12 and Remark 3.11, respectively.

```
Lemma Deno_asgn_dom: forall lv e s s',
                     Deno_a (asgn lv e) s s' -> dom s = dom s'.
Lemma Deno_b_framing: forall b s s1 s2 z,
                      spl s s1 s2 ->
                      Deno_b b s2 = Some z -> Deno_b b s = Some z.
Lemma Deno_a_framing: forall a s s1 s2,
                      spl s s1 s2 ->
                      (exists s2', Deno_a a s2 s2') ->
                      (exists s', Deno_a a s s') /\
                      (forall s', Deno_a a s s' ->
                         exists s2', spl s' s1 s2' /\ Deno_a a s2 s2').
```

---

seqsem.v

In this file, we define the sequential semantics as shown in Fig 3.9.

```
Inductive Seqstep: Strel :=
  | Seqstep_actn_1:
      forall S a s,
      SEQU S ->
      ~(exists s', Deno_a a s s') ->
      Seqstep (conf (tcomm (Subst S (actn a))) s) abrt
  | Seqstep_actn_2:
      forall S a s s',
      SEQU S ->
      Deno_a a s s' ->
      Seqstep (conf (tcomm (Subst S (actn a))) s)
              (conf (tcomm (Subst S skip)) s')
  | Seqstep_skip_1:
      forall S c s,
      SEQU S ->
      Seqstep (conf (tcomm (Subst S (sequ skip c))) s)
              (conf (tcomm (Subst S c)) s)
  | Seqstep_cond_1:
      forall S b c1 c2 s,
      SEQU S ->
      ~(exists z, Deno_b b s = Some z) ->
      Seqstep (conf (tcomm (Subst S (cond b c1 c2))) s) abrt
  | Seqstep_cond_2:
      forall S b c1 c2 s,
      SEQU S ->
      Deno_b b s = Some true ->
      Seqstep (conf (tcomm (Subst S (cond b c1 c2))) s)
              (conf (tcomm (Subst S c1)) s)
  | Seqstep_cond_3:
      forall S b c1 c2 s,
      SEQU S ->
      Deno_b b s = Some false ->
      Seqstep (conf (tcomm (Subst S (cond b c1 c2))) s)
              (conf (tcomm (Subst S c2)) s)
  | Seqstep_loop_1:
      forall S b c s,
      SEQU S ->
      Seqstep (conf (tcomm (Subst S (loop b c))) s)
        (conf (tcomm (Subst S (cond b (sequ c (loop b c)) skip))) s).
```

---

seqsemfoot.v

In this file, we define the sequential semantics with footprints as shown in Fig 3.10.

```
Inductive Seqstepfoot: LbStrel :=
  | Seqstepfoot_actn_1:
      forall S a s,
      SEQU S ->
      ~(exists s', Deno_a a s s') ->
      Seqstepfoot (conf (tcomm (Subst S (actn a))) s)
                  (Delta_a a s) abrt
  | Seqstepfoot_actn_2:
      forall S a s s',
      SEQU S ->
      Deno_a a s s' ->
      Seqstepfoot (conf (tcomm (Subst S (actn a))) s)
```

```
           (cup (Delta_a a s) (locset.emp,locset.min (dom s') (dom s)))
           (conf (tcomm (Subst S skip)) s')
  | Seqstepfoot_skip_1:
      forall S c s,
      SEQU S ->
      Seqstepfoot (conf (tcomm (Subst S (sequ skip c))) s)
              emp (conf (tcomm (Subst S c)) s)
  | Seqstepfoot_cond_1:
      forall S b c1 c2 s,
      SEQU S ->
      ~(exists z, Deno_b b s = Some z) ->
      Seqstepfoot (conf (tcomm (Subst S (cond b c1 c2))) s)
                  (Delta_b b s) abrt
  | Seqstepfoot_cond_2:
      forall S b c1 c2 s,
      SEQU S ->
      Deno_b b s = Some true ->
      Seqstepfoot (conf (tcomm (Subst S (cond b c1 c2))) s)
                  (Delta_b b s) (conf (tcomm (Subst S c1)) s)
  | Seqstepfoot_cond_3:
      forall S b c1 c2 s,
      SEQU S ->
      Deno_b b s = Some false ->
      Seqstepfoot (conf (tcomm (Subst S (cond b c1 c2))) s)
                  (Delta_b b s) (conf (tcomm (Subst S c2)) s)
  | Seqstepfoot_loop_1:
      forall S b c s,
      SEQU S ->
      Seqstepfoot (conf (tcomm (Subst S (loop b c))) s) emp
        (conf (tcomm (Subst S (cond b (sequ c (loop b c)) skip))) s).
```

We establish Remark 3.9, as well as the framing properties given by Lemma 3.16.

```
Lemma Seqstep_equiv:
  forall k k', Seqstep k k' <-> exists f, Seqstepfoot k f k'.
Lemma Seqstep_framing:
  forall c s s1 s2,
  spl s s1 s2 ->
  ~Seqstep (conf (tcomm c) s2) abrt ->
  ~Seqstep (conf (tcomm c) s) abrt /\
  (forall f c' s',
     Seqstepfoot (conf (tcomm c) s) f (conf (tcomm c') s') ->
     exists s2', spl s' s1 s2' /\
     Seqstepfoot (conf (tcomm c) s2) f (conf (tcomm c') s2')).
```

We also define the auxiliary properties required to establish the DRF-guarantee.

```
Lemma Seq_foot:
  forall c s f c' s',
  Seqstepfoot (conf (tcomm c) s) f (conf (tcomm c') s') ->
  sub f (cup (footstate s) (footstate s')).
```

```
Lemma Seq_Seq_rni_abrt:
  forall c1 c1' c2 s f1 f2,
  (exists s', Seqstepfoot (conf (tcomm c1) s) f1 (conf (tcomm c1') s')
           /\ Seqstepfoot (conf (tcomm c2) s') f2 abrt) ->
  rni f1 f2 ->
  Seqstepfoot (conf (tcomm c2) s) f2 abrt.
Lemma Seq_Seq_ni:
  forall c1 c1' c2 c2' s s'' f1 f2,
  (exists s', Seqstepfoot (conf (tcomm c1) s) f1 (conf (tcomm c1') s')
    /\ Seqstepfoot (conf (tcomm c2) s') f2 (conf (tcomm c2') s'')) ->
  ni f1 f2 ->
  (exists s', Seqstepfoot (conf (tcomm c2) s) f2 (conf (tcomm c2') s')
    /\ Seqstepfoot (conf (tcomm c1) s') f1 (conf (tcomm c1') s'')).
Lemma MultiSeq_MultiSeq_earliest_race:
  forall c1 c1' c2 s s' f1 f2 k2,
  LbMulti Seqstepfoot (conf (tcomm c1) s) f1 (conf (tcomm c1') s') ->
  LbMulti Seqstepfoot (conf (tcomm c2) s') f2 k2 ->
  ~rni f1 f2 ->
  exists c1', exists c2', exists s',
  (exists f1, exists f2, exists s'',
   LbMulti Seqstepfoot (conf (tcomm c1) s) f1 (conf (tcomm c1') s'') /\
   LbMulti Seqstepfoot (conf (tcomm c2) s'') f2 (conf (tcomm c2') s')
    /\ ni f1 f2) /\
  ((exists f1, exists c1'', exists s'', exists f2, exists k2,
     Seqstepfoot (conf (tcomm c1') s') f1 (conf (tcomm c1'') s'') /\
     Seqstepfoot (conf (tcomm c2') s'') f2 k2 /\ ~rni f1 f2) \/
   (exists f2, exists c2'', exists s'', exists f1, exists k1,
     Seqstepfoot (conf (tcomm c2') s') f2 (conf (tcomm c2'') s'') /\
     Seqstepfoot (conf (tcomm c1') s'') f1 k1 /\ ~rni f2 f1)).
```

subsumpn.v

In this file, we define the indexed subsumption relation (Def. 3.33).

```
(* indexed command subsumption *)
Inductive Subsumpn: nat -> Comm -> Comm -> Prop :=
  | Subsumpn_O:
      forall c1 c2, Subsumpn O c1 c2
  | Subsumpn_S:
      forall n c1 c2,
      (forall s, Multi Seqstep (conf (tcomm c2) s) abrt ->
                 Multi Seqstep (conf (tcomm c1) s) abrt) ->
      (forall s c2' s',
       Evalu Seqstep (conf (tcomm c2) s) (conf (tcomm c2') s') ->
       Multi Seqstep (conf (tcomm c1) s) abrt
       \/ exists c1',
          Evalu Seqstep (conf (tcomm c1) s) (conf (tcomm c1') s') /\
          ((c1' = skip /\ c2' = skip) \/
           (exists S1', exists c1'', exists c1''',
            exists S2', exists c2'', exists c2''',
            SEQU S1' /\ SEQU S2' /\
            c1' = Subst S1' (para c1'' c1''') /\
            c2' = Subst S2' (para c2'' c2''') /\
            Subsumpn n c1'' c2'' /\ Subsumpn n c1''' c2''' /\
            Subsumpn n (Subst S1' skip) (Subst S2' skip)) \/
           (exists S1', exists c1'', exists S2', exists c2'',
            SEQU S1' /\ SEQU S2' /\
            c1' = Subst S1' (atom c1'') /\
            c2' = Subst S2' (atom c2'') /\
            Subsumpn n c1'' c2'' /\
            Subsumpn n (Subst S1' skip) (Subst S2' skip)))) ->
      (forall s, Diver Seqstep (conf (tcomm c2) s) ->
                 Multi Seqstep (conf (tcomm c1) s) abrt
                 \/ Diver Seqstep (conf (tcomm c1) s)) ->
      (forall s f2 k2, LbMulti Seqstepfoot (conf (tcomm c2) s) f2 k2 ->
                       Multi Seqstep (conf (tcomm c1) s) abrt
                       \/ exists f1, exists k1,
                          LbMulti Seqstepfoot (conf (tcomm c1) s) f1 k1
                            /\ sub f2 f1) ->
      Subsumpn (S n) c1 c2.
```

We also establish many auxiliary properties of indexed subsumption, in special Lemma 3.37.

```
Lemma Subsumpn_le:
  forall n n' c1 c2, Subsumpn n c1 c2 -> n' <= n -> Subsumpn n' c1 c2.
```

---

subsump.v

In this file, we define the subsumption relation (Def. 3.33).

```
(* command subsumption *)
Definition Subsump (c1 c2: Comm): Prop := forall n, Subsumpn n c1 c2.
```

We establish the important properties of subsumption such as reflexitivity (Lemma 3.34), transitivity (Lemma 3.35), and contextualization (Lemma 3.36).

```
Lemma Subsump_reflx:
  forall c, Subsump c c.
Lemma Subsump_trans:
  forall c1 c2 c3, Subsump c1 c2 -> Subsump c2 c3 -> Subsump c1 c3.
Theorem Subsump_cntxequiv:
  forall C c1 c2, Subsump c1 c2 -> Subsump (Subst C c1) (Subst C c2).
```

We also establish Lemma 3.47.

```
Lemma Subsump_commu:
  forall c s c' s',
  (exists c'', Seqstep (conf (tcomm c) s) (conf (tcomm c'') s') /\
   Subsump c'' c') ->
  (exists c'', exists c''', Subsump c c'' /\
   Seqstep (conf (tcomm c'') s) (conf (tcomm c''') s') /\
   c_eq c''' c').
```

`tntx.v`

In this file, we define thread tree contexts, as shown in Fig. 3.6, and associated properties.

```
Inductive Tntx: Set := (* thread tree contexts *)
  | hole
  | tpara_1 (T: Tntx) (t: Tree) (c: Comm)
  | tpara_2 (t: Tree) (T: Tntx) (c: Comm)
  | tatom_1 (T: Tntx) (c: Comm).

(* context/tree substitution *)
Fixpoint Subst (T: Tntx) (t': Tree): Tree :=
  match T with
  | hole => t'
  | tpara_1 T t c => tpara (Subst T t') t c
  | tpara_2 t T c => tpara t (Subst T t') c
  | tatom_1 T c => tatom (Subst T t') c
  end.
(* context/context substitution *)
Fixpoint Substx (T: Tntx) (T': Tntx): Tntx :=
  match T with
  | hole => T'
  | tpara_1 T t c => tpara_1 (Substx T T') t c
  | tpara_2 t T c => tpara_2 t (Substx T T') c
  | tatom_1 T c => tatom_1 (Substx T T') c
  end.
```

We define a predicate to characterize n-atomic thread tree contexts (Def. 3.3).

```
Inductive ATOM: nat -> Tntx -> Prop :=
  | ATOM_hole: ATOM 0 hole
  | ATOM_tpara_1: forall n1 n2 T t c,
                  ATOM n1 T ->
                  runtime.ATOM n2 t ->
                  ATOM (n1+n2) (tpara_1 T t c)
  | ATOM_tpara_2: forall n1 n2 t T c,
                  runtime.ATOM n1 t ->
                  ATOM n2 T ->
                  ATOM (n1+n2) (tpara_2 t T c)
  | ATOM_tatom_1: forall n T c,
                  ATOM n T ->
                  ATOM n (tatom_1 T c).
```

And we also extend the weaker notion of equality to thread trees and configurations.

```
Fixpoint t_eq (t1 t2: Tree): Prop :=
  match t1,t2 with
  | tcomm c1,tcomm c2 => c_eq c1 c2
  | tpara t1 t1' c1,tpara t2 t2' c2 => t_eq t1 t2 /\ t_eq t1' t2'
                                    /\ c_eq c1 c2
  | tatom t1 c1,tatom t2 c2 => t_eq t1 t2 /\ c_eq c1 c2
  | _,_ => False
  end.
Definition k_eq (k1 k2: Conf): Prop :=
  match k1,k2 with
  | conf t1 s1,conf t2 s2 => t_eq t1 t2 /\ s1 = s2
  | abrt,abrt => True
  | race,race => True
  | _,_ => False
  end.
```

---

smlsem.v

In this file, we define the small-step interleaved semantics as shown in Fig 3.11.

```
Inductive Smlstep: Strel :=
  | Smlstep_comm_1:
      forall T c s,
      Seqstep (conf (tcomm c) s) abrt ->
      Smlstep (conf (Subst T (tcomm c)) s) abrt
  | Smlstep_comm_2:
      forall T c s c' s',
      Seqstep (conf (tcomm c) s) (conf (tcomm c') s') ->
      Smlstep (conf (Subst T (tcomm c)) s)
              (conf (Subst T (tcomm c')) s')
  | Smlstep_para_1:
      forall T S c1 c2 s,
      SEQU S ->
      Smlstep (conf (Subst T (tcomm (cntx.Subst S (para c1 c2)))) s)
              (conf (Subst T (tpara (tcomm c1)
                                    (tcomm c2) (cntx.Subst S skip))) s)
  | Smlstep_para_2:
      forall T c s,
      Smlstep (conf (Subst T (tpara (tcomm skip) (tcomm skip) c)) s)
              (conf (Subst T (tcomm c)) s)
  | Smlstep_atom_1:
      forall T S c s,
      ATOM 0 T ->
      SEQU S ->
      Smlstep (conf (Subst T (tcomm (cntx.Subst S (atom c)))) s)
              (conf (Subst T (tatom (tcomm c) (cntx.Subst S skip))) s)
  | Smlstep_atom_2:
      forall T c s,
      Smlstep (conf (Subst T (tatom (tcomm skip) c)) s)
              (conf (Subst T (tcomm c)) s)
  | Smlstep_race_1:
      forall T T1 c1 T2 c2 c s,
      (exists f1, exists c1', exists s', exists f2, exists k,
       Seqstepfoot (conf (tcomm c1) s) f1 (conf (tcomm c1') s') /\
       Seqstepfoot (conf (tcomm c2) s') f2 k /\ ~rni f1 f2) ->
      Smlstep (conf (Subst T (tpara (Subst T1 (tcomm c1))
                               (Subst T2 (tcomm c2)) c)) s) race
  | Smlstep_race_2:
      forall T T1 c1 T2 c2 c s,
      (exists f2, exists c2', exists s', exists f1, exists k,
       Seqstepfoot (conf (tcomm c2) s) f2 (conf (tcomm c2') s') /\
       Seqstepfoot (conf (tcomm c1) s') f1 k /\ ~rni f2 f1) ->
      Smlstep (conf (Subst T (tpara (Subst T1 (tcomm c1))
                               (Subst T2 (tcomm c2)) c)) s) race.
```

We also define Remark 3.17 and Lemma 3.22.

```
Lemma Smlstep_ATOM:
  forall t s t' s',
  (runtime.ATOM 0 t \/ runtime.ATOM 1 t) ->
  Smlstep (conf t s) (conf t' s') ->
  (runtime.ATOM 0 t' \/ runtime.ATOM 1 t').
Lemma Smlstep_framing:
  forall t s s1 s2,
  spl s s1 s2 ->
  ~Smlstep (conf t s2) abrt ->
  ~Smlstep (conf t s) abrt /\
  (~Smlstep (conf t s2) race -> ~Smlstep (conf t s) race) /\
  (forall t' s', Smlstep (conf t s) (conf t' s') ->
     exists s2', spl s' s1 s2' /\ Smlstep (conf t s2) (conf t' s2')).
```

---

mixsem.v

In this file, we define the mixed-step interleaved semantics, as shown in Fig 5.1, and the mixed-step divergence (Def. 5.3).

```
Inductive Mixstep: Strel :=
  | Mixstep_comm_1:
      forall T c s,
      Multi Seqstep (conf (tcomm c) s) abrt ->
      Mixstep (conf (Subst T (tcomm c)) s) abrt
  | Mixstep_comm_2:
      forall T c s c' s',
      Evalu Seqstep (conf (tcomm c) s) (conf (tcomm c') s') ->
      c <> c' ->
      Mixstep (conf (Subst T (tcomm c)) s)
              (conf (Subst T (tcomm c')) s')
  | Mixstep_para_1:
      forall T S c1 c2 s,
      SEQU S ->
      Mixstep (conf (Subst T (tcomm (cntx.Subst S (para c1 c2)))) s)
              (conf (Subst T (tpara (tcomm c1)
                                    (tcomm c2) (cntx.Subst S skip))) s)
  | Mixstep_para_2:
      forall T c s,
      Mixstep (conf (Subst T (tpara (tcomm skip) (tcomm skip) c)) s)
              (conf (Subst T (tcomm c)) s)
  | Mixstep_atom_1:
      forall T S c s,
      ATOM 0 T ->
      SEQU S ->
      Mixstep (conf (Subst T (tcomm (cntx.Subst S (atom c)))) s)
              (conf (Subst T (tatom (tcomm c) (cntx.Subst S skip))) s)
  | Mixstep_atom_2:
      forall T c s,
      Mixstep (conf (Subst T (tatom (tcomm skip) c)) s)
              (conf (Subst T (tcomm c)) s)
  | Mixstep_race_1:
      forall T T1 c1 T2 c2 c s,
      (exists f1, exists c1', exists s', exists f2, exists k,
       LbMulti Seqstepfoot (conf (tcomm c1) s) f1 (conf (tcomm c1') s')
         /\ LbMulti Seqstepfoot (conf (tcomm c2) s') f2 k
         /\ ~rni f1 f2) ->
      Mixstep (conf (Subst T (tpara (Subst T1 (tcomm c1))
                                    (Subst T2 (tcomm c2)) c)) s) race
  | Mixstep_race_2:
      forall T T1 c1 T2 c2 c s,
      (exists f2, exists c2', exists s', exists f1, exists k,
       LbMulti Seqstepfoot (conf (tcomm c2) s) f2 (conf (tcomm c2') s')
         /\ LbMulti Seqstepfoot (conf (tcomm c1) s') f1 k
         /\ ~rni f2 f1) ->
      Mixstep (conf (Subst T (tpara (Subst T1 (tcomm c1))
                                    (Subst T2 (tcomm c2)) c)) s) race.

Definition DiverMixstep (k: Conf): Prop :=
  Diver Mixstep k \/
  exists T, exists c, exists s,
  Multi Mixstep k (conf (Subst T (tcomm c)) s)
    /\ Diver Seqstep (conf (tcomm c) s).
```

We also establish many important properties such as the ones given by Lemma 5.4, Lemma 5.9, and Lemma 5.5.

```
Lemma MultiMix_MultiSml:
  forall k k', Multi Mixstep k k' -> Multi Smlstep k k'.
Lemma DiverMix_DiverSml:
  forall k, DiverMixstep k -> Diver Smlstep k.

Lemma Sml_Mix_compo:
  forall k k' k'',
  Smlstep k k' ->
  Mixstep k' k'' ->
  Mixstep k race
    \/ Multi Mixstep k k''
    \/ (exists k', Mixstep k k' /\ Smlstep k' k'').

Lemma MultiSml_MultiMix_race:
  forall k,
  Multi Smlstep k race ->
  Multi Mixstep k race.
Lemma MultiSml_MultiMix_abrt:
  forall k,
  ~Multi Smlstep k race ->
  Multi Smlstep k abrt ->
  Multi Mixstep k abrt.
Lemma MultiSml_MultiMix_skip:
  forall k s,
  ~Multi Smlstep k race ->
  Multi Smlstep k (conf (tcomm skip) s) ->
  Multi Mixstep k (conf (tcomm skip) s).
Lemma DiverSml_DiverMix:
  forall k,
  ~Multi Smlstep k race ->
  Diver Smlstep k ->
  DiverMixstep k.
```

---

prmsem.v

In this file, we define the lifting of transformations from commands to thread trees (Def. 3.23), the parameterized semantics (Figure. 3.13), and the DRF-guarantee (Def. 5.2).

```
Definition TrfrmT: Type := Tree -> Tree -> Prop.

Fixpoint Lift (L: Trfrm) (t1 t2: Tree): Prop :=
  match t1,t2 with
  | tcomm c1,tcomm c2 => L c1 c2
  | tpara t1 t1' c1,tpara t2 t2' c2 => Lift L t1 t2
                                 /\ Lift L t1' t2' /\ L c1 c2
  | tatom t1 c1,tatom t2 c2 => Lift L t1 t2 /\ L c1 c2
  | _,_ => False
  end.

Inductive Prmstep: Trfrm -> Strel :=
  | Prmstep_trfrm:
      forall L t s k,
      (exists t', Lift L t t' /\ Smlstep (conf t' s) k) ->
      Prmstep L (conf t s) k.

Definition DRF (L: Trfrm): Prop :=
  forall k, ~Multi Smlstep k abrt ->
            ~Multi Smlstep k race ->
```

```
          ~Multi (Prmstep L) k abrt /\
          (forall s, Multi (Prmstep L) k (conf (tcomm skip) s) ->
                    Multi Smlstep k (conf (tcomm skip) s)).
```

---

rlxsem.v

In this file, we establish the important properties of the relaxed semantics given by Lemma 5.10, Corollary 5.11, Lemma 3.46, and Theorem 5.12.

```
Definition SubsumpT: TrfrmT := Lift Subsump.

Lemma MultiMix_MultiMix_abrt:
  forall t1 t2 s,
  SubsumpT t1 t2 ->
  ~Multi Mixstep (conf t1 s) abrt ->
  ~Multi Mixstep (conf t2 s) abrt.
Lemma MultiMix_MultiMix_race:
  forall t1 t2 s,
  SubsumpT t1 t2 ->
  ~Multi Mixstep (conf t1 s) abrt ->
  Multi Mixstep (conf t2 s) race ->
  Multi Mixstep (conf t1 s) race.
Lemma MultiMix_MultiMix_skip:
  forall t1 t2 s s',
  SubsumpT t1 t2 ->
  ~Multi Mixstep (conf t1 s) abrt ->
  Multi Mixstep (conf t2 s) (conf (tcomm skip) s') ->
  Multi Mixstep (conf t1 s) (conf (tcomm skip) s').
Lemma DiverMix_DiverMix_dive:
  forall t1 t2 s,
  SubsumpT t1 t2 ->
  ~Multi Mixstep (conf t1 s) abrt ->
  DiverMixstep (conf t2 s) ->
  DiverMixstep (conf t1 s).

Lemma MultiSml_MultiSml_race:
  forall t1 t2 s,
  SubsumpT t1 t2 ->
  ~Multi Smlstep (conf t1 s) abrt ->
  ~Multi Smlstep (conf t1 s) race ->
  ~Multi Smlstep (conf t2 s) race.
Lemma MultiSml_MultiSml_abrt:
  forall t1 t2 s,
  SubsumpT t1 t2 ->
  ~Multi Smlstep (conf t1 s) race ->
  ~Multi Smlstep (conf t1 s) abrt ->
  ~Multi Smlstep (conf t2 s) abrt.
Lemma MultiSml_MultiSml_skip:
  forall t1 t2 s s',
  SubsumpT t1 t2 ->
  ~Multi Smlstep (conf t1 s) abrt ->
  ~Multi Smlstep (conf t1 s) race ->
  Multi Smlstep (conf t2 s) (conf (tcomm skip) s') ->
  Multi Smlstep (conf t1 s) (conf (tcomm skip) s').
Lemma DiverSml_DiverSml:
  forall t1 t2 s,
  SubsumpT t1 t2 ->
  ~Multi Smlstep (conf t1 s) abrt ->
  ~Multi Smlstep (conf t1 s) race ->
  Diver Smlstep (conf t2 s) ->
  Diver Smlstep (conf t1 s).

Lemma MultiRlx_equiv:
  forall t s k,
  Multi (Prmstep Subsump) (conf t s) k <->
    exists t', exists k', SubsumpT t t' /\
      Multi Smlstep (conf t' s) k' /\ k_eq k' k.

Theorem DRF_Subsump: DRF Subsump.
```

---

specs.v

In this file, we define the assertion language used by both CSL (Fig. 6.1 and Fig. 6.2) and SAGL (Fig. 8.1 and Fig. 8.2) as a shallow embedding in the Coq metalogic.

```
Definition Assert: Type := State -> Prop. (* assertions *)
Definition Action: Type := State -> State -> Prop. (* bin assertions *)

(* common definitions *)
Definition crs (p q: Assert): Action := fun s s' => p s /\ q s'.
Definition dom (g: Action): Assert := fun s => exists s', g s s'.
Definition img (g: Action): Assert := fun s' => exists s, g s s'.
Definition land (p1 p2: Assert): Assert := fun s => p1 s /\ p2 s.
Definition lor (p1 p2: Assert): Assert := fun s => p1 s \/ p2 s.
Definition lex (A: Type) (P: A -> Assert): Assert :=
  fun s => exists a, P a s.
Definition dand (p1 p2: Assert): Assert :=
  fun s => exists s1, exists s2, spl s s1 s2 /\ p1 s1 /\ p2 s2.
Definition land2 (g1 g2: Action): Action :=
  fun s s' => g1 s s' /\ g2 s s'.
Definition lor2 (g1 g2: Action): Action :=
  fun s s' => g1 s s' \/ g2 s s'.
Definition lex2 (A: Type) (G: A -> Action): Action :=
  fun s s' => exists a, G a s s'.
Definition dand2 (g1 g2: Action): Action :=
  fun s s' => exists s1, exists s2, exists s1', exists s2',
              spl s s1 s2 /\ spl s' s1' s2' /\ g1 s1 s1' /\ g2 s2 s2'.
Definition pog (p: Assert) (g: Action) : Assert :=
  fun s => forall s', g s s' -> p s'.
Definition gog (g2 g1: Action): Action :=
  fun s s' => exists s'', g1 s s'' /\ g2 s'' s'.

(* constants *)
Definition ID: Action := fun s s' => s = s'.
Definition TRUE: Assert := fun s => True.
Definition FALSE: Assert := fun s => False.
Definition EMP: Assert := fun s => s = emp.
Definition EMP2: Action := crs EMP EMP.

(* implication *)
Definition aimp (p1 p2: Assert): Prop := forall s, p1 s -> p2 s.
Definition aimp2 (g1 g2: Action): Prop := forall s s', g1 s s' -> g2 s s'.
```

We also define the notions of precision (Def. 6.1) and coincidence (Def. 6.35).

```
Definition Prec (p: Assert): Prop := (* precision *)
  forall s s1 s2 s1' s2',
  spl s s1 s2 -> spl s s1' s2' -> p s1 -> p s1' -> s1 = s1'.
Definition Coin (p1 p2: Assert): Prop := (* coincidence *)
  forall s s1 s2 s1' s2',
  spl s s1 s2 -> spl s s1' s2' -> p1 s1 -> p2 s1' -> s1 = s1'.
```

---

logic_common.v

In this file, we define common logic constructs and properties that are tied to the language and shared by both CSL and SAGL. In particular, we define lifting of conditions into assertions (Fig. 6.2) and the weakest precondition of an action (Def. 6.2).

```
Definition lfb (b: Cond): Assert :=
  fun s => Deno_b b s = Some true.
Definition qoa (q: Assert) (a: Actn): Assert :=
  land (dom (Deno_a a)) (pog q (Deno_a a)).
```

---

csl.v

In this file, we define the set of inference rules for CSL, as presented in Fig 6.3 (including the generalized conjunction rule presented in Sec. 6.4). The definition carries the maximum depth of the derivation tree, which is useful when performing induction in the proofs.

```
Inductive WFCommn: nat -> Assert -> Assert -> Comm -> Assert -> Prop :=
  | Rulen_asgn:
      forall lv e q,
      WFCommn O EMP (qoa q (asgn lv e)) (actn (asgn lv e)) q
```

```
| Rulen_actn:
    forall a q,
    WFCommn O EMP (qoa q a) (atom (actn a)) q
| Rulen_sequ:
    forall n1 n2 I p c1 c2 q,
    (exists p', WFCommn n1 I p c1 p' /\ WFCommn n2 I p' c2 q) ->
    WFCommn (S (max n1 n2)) I p (sequ c1 c2) q
| Rulen_skip:
    forall I p,
    WFCommn O I p skip p
| Rulen_cond:
    forall n1 n2 I p b c1 c2 q,
    aimp p (lor (lfb b) (lfb (bneg b))) ->
    WFCommn n1 I (land p (lfb b)) c1 q ->
    WFCommn n2 I (land p (lfb (bneg b))) c2 q ->
    WFCommn (S (max n1 n2)) I p (cond b c1 c2) q
| Rulen_loop:
    forall n I p b c,
    aimp p (lor (lfb b) (lfb (bneg b))) ->
    WFCommn n I (land p (lfb b)) c p ->
    WFCommn (S n) I p (loop b c) (land p (lfb (bneg b)))
| Rulen_para:
    forall n1 n2 I p1 p2 c1 c2 q1 q2,
    WFCommn n1 I p1 c1 q1 ->
    WFCommn n2 I p2 c2 q2 ->
    WFCommn (S (max n1 n2)) I (dand p1 p2) (para c1 c2) (dand q1 q2)
| Rulen_atom:
    forall n I p c q,
    WFCommn n EMP (dand I p) c (dand I q) ->
    WFCommn (S n) I p (atom c) q
| Rulen_cons:
    forall n I p c q,
    (exists p', exists q',
     aimp p p' /\ WFCommn n I p' c q' /\ aimp q' q) ->
    WFCommn (S n) I p c q
| Rulen_exst:
    forall n A I P c Q,
    (forall x : A, WFCommn n I (P x) c (Q x)) ->
    WFCommn (S n) I (lex P) c (lex Q)
| Rulen_fram:
    forall n I' I p c q,
    WFCommn n I p c q ->
    WFCommn (S n) (dand I' I) p c q
| Rulen_resr:
    forall n I' I p c q,
    WFCommn n (dand I' I) p c q ->
    WFCommn (S n) I (dand I' p) c (dand I' q)
| Rulen_conj:
    forall n1 n2 I p1 p2 c q1 q2,
    Prec I ->
    WFCommn n1 I p1 c q1 ->
    WFCommn n2 I p2 c q2 ->
    WFCommn (S (max n1 n2)) I (land p1 p2) c (land q1 q2)
```

```
  | Rulen_conj2:
      forall n1 n2 I1 I2 p1 p2 c q1 q2,
      Coin I1 I2 ->
      WFCommn n1 I1 p1 c q1 ->
      WFCommn n2 I2 p2 c q2 ->
      WFCommn (S (max n1 n2)) (land I1 I2) (land p1 p2) c (land q1 q2)
  | Rulen_disj:
      forall n1 n2 I p1 p2 c q1 q2,
      WFCommn n1 I p1 c q1 ->
      WFCommn n2 I p2 c q2 ->
      WFCommn (S (max n1 n2)) I (lor p1 p2) c (lor q1 q2).
Definition WFComm (I p: Assert) (c: Comm) (q: Assert): Prop :=
  exists n, WFCommn n I p c q.
```

csl_soundness.v

In this file, we implement the soundness proof for CSL, as described in Sec. 6.3. Below, we define the semantic triple (Def. 6.3), semantic judgement (Def. 6.15), and parameterized semantic judgement (Def. 6.32) for CSL. Note that in this implementation, when defining triples, we removed the constraint that a thread tree must be either 0- or 1-atomic, given by Def. 6.3 (item 1), as it is an invariant of the semantics as given by Remark 3.17.

```
Inductive SSConfn: nat -> Assert -> Conf -> Assert -> Prop :=
  | SSConfn_O:
      forall I t s q,
      SSConfn O I (conf t s) q
  | SSConfn_S:
      forall n I t s q,
      ~Smlstep (conf t s) abrt ->
      ~Smlstep (conf t s) race ->
      (forall t' s', Smlstep (conf t s) (conf t' s') ->
        (ATOM 0 t -> ((ATOM 0 t' -> state.dom s = state.dom s'
                                 /\ SSConfn n I (conf t' s') q) /\
                      (ATOM 1 t' -> forall sx' ss',
                                    spl sx' ss' s' -> I ss' ->
                                    SSConfn n I (conf t' sx') q))) /\
        (ATOM 1 t -> ((ATOM 1 t' -> SSConfn n I (conf t' s') q) /\
                      (ATOM 0 t' -> exists s1', exists s2',
                                    spl s' s1' s2' /\ I s1' /\
                                    SSConfn n I (conf t' s2') q)))) ->
      (t = tcomm skip -> q s) ->
      SSConfn (S n) I (conf t s) q.
Definition SSConf (I: Assert) (k: Conf) (q: Assert): Prop :=
  forall n, SSConfn n I k q.

Definition SSComm (I p: Assert) (c: Comm) (q: Assert): Prop :=
  forall s, p s -> SSConf I (conf (tcomm c) s) q.
```

```
Definition SSComm_prm (L: Trfrm) (I p: Assert)
                      (c: Comm) (q: Assert) : Prop :=
  forall s, dand I p s ->
    ~Multi (Prmstep L) (conf (tcomm c) s) abrt /\
    forall s', Multi (Prmstep L) (conf (tcomm c) s)
                             (conf (tcomm skip) s') -> dand I q s'.
```

We establish the auxiliary properties of semantic triples (Lemma 6.4 through Lemma 6.14, and Lemma 6.39).

```
Lemma SSConf_n:
  forall n1 n2 I t s q,
  n2 <= n1 ->
  SSConfn n1 I (conf t s) q ->
  SSConfn n2 I (conf t s) q.
Lemma SSConf_cons:
  forall n I t s q' q,
  aimp q' q ->
  SSConfn n I (conf t s) q' ->
  SSConfn n I (conf t s) q.
Lemma SSConf_resr:
  forall n I' I t s q,
  (ATOM 0 t \/ ATOM 1 t) ->
  (ATOM 0 t -> exists s1, exists s2, spl s s1 s2 /\ I' s1
                /\ SSConfn n (dand I' I) (conf t s2) q) /\
  (ATOM 1 t -> SSConfn n (dand I' I) (conf t s) q) ->
  SSConfn n I (conf t s) (dand I' q).
Lemma SSConf_fram:
  forall n I' I t s q,
  (ATOM 0 t \/ ATOM 1 t) ->
  (ATOM 0 t -> SSConfn n I (conf t s) q) /\
  (ATOM 1 t -> exists s1, exists s2, spl s s1 s2 /\ I' s1
                /\ SSConfn n I (conf t s2) q) ->
  SSConfn n (dand I' I) (conf t s) q.
Lemma SSConf_conj:
  forall n I t s q1 q2,
  Prec I ->
  SSConfn n I (conf t s) q1 ->
  SSConfn n I (conf t s) q2 ->
  SSConfn n I (conf t s) (land q1 q2).
Lemma SSConf_conj2:
  forall n I1 I2 t s q1 q2,
  Coin I1 I2 ->
  SSConfn n I1 (conf t s) q1 ->
  SSConfn n I2 (conf t s) q2 ->
  SSConfn n (land I1 I2) (conf t s) (land q1 q2).
Lemma SSConf_skip:
  forall I s (q : Assert),
  q s ->
  SSConf I (conf (tcomm skip) s) q.
Lemma SSConf_asgn:
  forall lv e s q,
  qoa q (asgn lv e) s ->
  SSConf EMP (conf (tcomm (actn (asgn lv e))) s) q.
Lemma SSConf_actn:
  forall a s q,
  qoa q a s ->
  SSConf EMP (conf (tcomm (atom (actn a))) s) q.
Lemma SSConf_atom:
  forall n (I : Assert) c s q,
  (forall ss sx, spl sx ss s -> I ss ->
                SSConfn n EMP (conf (tcomm c) sx) (dand I q)) ->
  SSConfn (S n) I (conf (tcomm (atom c)) s) q.
Lemma SSConf_sequ:
  forall n I t c' t' s (p : Assert) q,
  ((exists c, t = tcomm c /\ t' = tcomm (sequ c c')) \/
   (exists t1, exists t2, exists c, t = tpara t1 t2 c
                                  /\ t' = tpara t1 t2 (sequ c c')) \/
   (exists t'', exists c, t = tatom t'' c
                       /\ t' = tatom t'' (sequ c c'))) ->
  SSConfn n I (conf t s) p ->
  (forall s, p s -> SSConfn n I (conf (tcomm c') s) q) ->
  SSConfn n I (conf t' s) q.
Lemma SSConf_cond:
  forall n I b c c' s q,
  SSConfn n I (conf (tcomm c) s) q ->
  (lfb b s -> SSConfn n I (conf (tcomm (cond b c c')) s) q) /\
  (lfb (bneg b) s -> SSConfn n I (conf (tcomm (cond b c' c)) s) q).
Lemma SSConf_loop:
  forall n I b c s p,
  aimp p (lor (lfb b) (lfb (bneg b))) ->
  p s ->
  (forall s', land p (lfb b) s' ->
              SSConfn n I (conf (tcomm c) s') p) ->
  SSConfn n I (conf (tcomm (loop b c)) s) (land p (lfb (bneg b))).
Lemma SSConf_para:
  forall n I  t1 t2 s s1 s2 q1 q2,
  (ATOM 0 (tpara t1 t2 skip) \/ ATOM 1 (tpara t1 t2 skip)) ->
  spl s s1 s2 ->
  SSConfn n I (conf t1 s1) q1 ->
  SSConfn n I (conf t2 s2) q2 ->
  SSConfn n I (conf (tpara t1 t2 skip) s) (dand q1 q2).
```

We establish the semantic rules (Lemma 6.17 through Lemma 6.30, and Lemma 6.40).

```
Lemma SemRule_asgn:
  forall lv e q,
  SSComm EMP (qoa q (asgn lv e)) (actn (asgn lv e)) q.
Lemma SemRule_actn:
  forall a q,
  SSComm EMP (qoa q a) (atom (actn a)) q.
Lemma SemRule_sequ:
  forall I p c1 c2 q,
  (exists p', SSComm I p c1 p' /\ SSComm I p' c2 q) ->
  SSComm I p (sequ c1 c2) q.
Lemma SemRule_skip:
  forall I q,
  SSComm I q skip q.
Lemma SemRule_cond:
  forall I p b c1 c2 q,
  aimp p (lor (lfb b) (lfb (bneg b))) ->
  SSComm I (land p (lfb b)) c1 q ->
  SSComm I (land p (lfb (bneg b))) c2 q ->
  SSComm I p (cond b c1 c2) q.
Lemma SemRule_loop:
  forall I p b c,
  aimp p (lor (lfb b) (lfb (bneg b))) ->
  SSComm I (land p (lfb b)) c p ->
  SSComm I p (loop b c) (land p (lfb (bneg b))).
Lemma SemRule_para:
  forall I p1 p2 c1 c2 q1 q2,
  SSComm I p1 c1 q1 ->
  SSComm I p2 c2 q2 ->
  SSComm I (dand p1 p2) (para c1 c2) (dand q1 q2).
Lemma SemRule_atom:
  forall I p c q,
  SSComm EMP (dand I p) c (dand I q) ->
  SSComm I p (atom c) q.
Lemma SemRule_cons:
  forall I p c q,
  (exists p', exists q', aimp p p' /\ SSComm I p' c q' /\ aimp q' q) ->
  SSComm I p c q.
Lemma SemRule_exst:
  forall A I P c Q,
  (forall x: A, SSComm I (P x) c (Q x)) ->
  SSComm I (lex P) c (lex Q).
Lemma SemRule_fram:
  forall I' I p c q,
  SSComm I p c q ->
  SSComm (dand I' I) p c q.
Lemma SemRule_resr:
  forall I' I p c q,
  SSComm (dand I' I) p c q ->
  SSComm I (dand I' p) c (dand I' q).
Lemma SemRule_conj:
  forall I p1 p2 c q1 q2,
  Prec I ->
  SSComm I p1 c q1 ->
  SSComm I p2 c q2 ->
  SSComm I (land p1 p2) c (land q1 q2).
Lemma SemRule_conj2:
  forall I1 I2 p1 p2 c q1 q2,
  Coin I1 I2 ->
  SSComm I1 p1 c q1 ->
  SSComm I2 p2 c q2 ->
  SSComm (land I1 I2) (land p1 p2) c (land q1 q2).
```

```
Lemma SemRule_disj:
  forall I p1 p2 c q1 q2,
  SSComm I p1 c q1 ->
  SSComm I p2 c q2 ->
  SSComm I (lor p1 p2) c (lor q1 q2).
```

And, finally, we establish the soundness with regard to the interleaved (Theorem 6.31 and Lemma 6.16), parameterized (Theorem 6.33), and relaxed semantics (Theorem 6.34).

```
Theorem Soundness:
  forall I p c q,
  WFComm I p c q ->
  SSComm I p c q.
Lemma Explicit_Soundness:
  forall I p c s q,
  SSComm I p c q ->
  dand I p s ->
  ~Multi Smlstep (conf (tcomm c) s) abrt /\
  ~Multi Smlstep (conf (tcomm c) s) race /\
  (forall s', Multi Smlstep (conf (tcomm c) s) (conf (tcomm skip) s') ->
              dand I q s').
Theorem Soundness_prm:
  forall L I p c q,
  WFComm I p c q ->
  DRF L ->
  SSComm_prm L I p c q.
Theorem Soundness_rlx:
  forall I p c q,
  WFComm I p c q ->
  SSComm_prm Subsump I p c q.
```

---

sagl.v

In this file, we define the set of inference rules for SAGL, as presented in Fig 8.3. The definition carries the maximum depth of the derivation tree, which is useful when performing induction in the proofs.

```
Inductive WFCommn: nat -> (Action * Action) -> (Assert * Assert) ->
                         Comm -> (Assert * Assert) -> Prop :=
  | Rulen_asgn:
      forall lv e q,
      WFCommn O (EMP2,EMP2)
                (EMP,qoa q (asgn lv e)) (actn (asgn lv e)) (EMP,q)
  | Rulen_actn:
      forall a q,
      WFCommn O (EMP2,EMP2) (EMP,qoa q a) (atom (actn a)) (EMP,q)
  | Rulen_sequ:
      forall n1 n2 E p_s p c1 c2 q_s q,
      (exists p_s', exists p', WFCommn n1 E (p_s,p) c1 (p_s',p')
        /\ WFCommn n2 E (p_s',p') c2 (q_s,q)) ->
      WFCommn (S (max n1 n2)) E (p_s,p) (sequ c1 c2) (q_s,q)
  | Rulen_skip:
      forall E p_s p,
      WFCommn O E (p_s,p) skip (p_s,p)
  | Rulen_cond:
      forall n1 n2 E p_s p b c1 c2 q_s q,
      aimp p (lor (lfb b) (lfb (bneg b))) ->
      WFCommn n1 E (p_s,land p (lfb b)) c1 (q_s,q) ->
      WFCommn n2 E (p_s,land p (lfb (bneg b))) c2 (q_s,q) ->
      WFCommn (S (max n1 n2)) E (p_s,p) (cond b c1 c2) (q_s,q)
  | Rulen_loop:
      forall n E p_s p b c,
      aimp p (lor (lfb b) (lfb (bneg b))) ->
      WFCommn n E (p_s,land p (lfb b)) c (p_s,p) ->
      WFCommn (S n) E (p_s,p) (loop b c) (p_s,land p (lfb (bneg b)))
  | Rulen_para:
      forall n1 n2 A1 A2 G p_s1 p1 p_s2 p2 c1 c2 q_s1 q1 q_s2 q2,
      aimp p_s1 (pog p_s1 A1) ->
      aimp p_s2 (pog p_s2 A2) ->
      WFCommn n1 (A1,land2 G A2) (p_s1,p1) c1 (q_s1,q1) ->
      WFCommn n2 (A2,land2 G A1) (p_s2,p2) c2 (q_s2,q2) ->
      WFCommn (S (max n1 n2)) (land2 A1 A2,G)
              (land p_s1 p_s2,dand p1 p2) (para c1 c2)
              (land q_s1 q_s2,dand q1 q2)
  | Rulen_atom:
      forall n A G p_s p c q_s q,
      WFCommn n (EMP2,EMP2) (EMP,dand p_s p) c (EMP,dand q_s q) ->
      aimp2 (crs p_s q_s) G ->
      aimp q_s (pog q_s A) ->
      WFCommn (S n) (A,G) (p_s,p) (atom c) (q_s,q)
  | Rulen_cons:
      forall n A G p_s p c q_s q,
      (exists A', exists G', exists p_s', exists p',
       exists q_s', exists q', aimp2 A A' /\ aimp p_s p_s'
        /\ aimp p p' /\ WFCommn n (A',G') (p_s',p') c (q_s',q')
        /\ aimp2 G' G /\ aimp q_s' q_s /\ aimp q' q) ->
      WFCommn (S n) (A,G) (p_s,p) c (q_s,q)
  | Rulen_exst:
      forall n X A G p_s P c q_s Q,
      (forall x: X, WFCommn n (A,G) (p_s,P x) c (q_s,Q x)) ->
      WFCommn (S n) (A,G) (p_s,lex P) c (q_s,lex Q)
  | Rulen_fram:
      forall n I A G p_s p c q_s q,
      Prec I \/ Coin (img G) (dom A) ->
      WFCommn n (A,G) (p_s,p) c (q_s,q) ->
      WFCommn (S n) (dand2 (crs I I) A,dand2 (crs I I) G)
                    (dand I p_s,p) c (dand I q_s,q)
  | Rulen_resr:
      forall n A G p_s p c q_s q,
      WFCommn n (A,G) (p_s,p) c (q_s,q) ->
      WFCommn (S n) (EMP2,EMP2) (EMP,dand p_s p) c (EMP,dand q_s q)
```

```
  | Rulen_conj:
      forall n1 n2 A1 A2 G1 G2 p_s1 p1 p_s2 p2 c q_s1 q1 q_s2 q2,
      Coin (img G1) (img G2) ->
      WFCommn n1 (A1,G1) (p_s1,p1) c (q_s1,q1) ->
      WFCommn n2 (A2,G2) (p_s2,p2) c (q_s2,q2) ->
      WFCommn (S (max n1 n2)) (land2 A1 A2,land2 G1 G2)
        (land p_s1 p_s2,land p1 p2) c (land q_s1 q_s2,land q1 q2)
  | Rulen_disj:
      forall n1 n2 A G p_s p1 p2 c q_s q1 q2,
      WFCommn n1 (A,G) (p_s,p1) c (q_s,q1) ->
      WFCommn n2 (A,G) (p_s,p2) c (q_s,q2) ->
      WFCommn (S (max n1 n2)) (A,G) (p_s,lor p1 p2) c (q_s,lor q1 q2).
Definition WFComm (E: Action * Action) (P: Assert * Assert)
                  (c: Comm) (Q: Assert * Assert): Prop :=
  exists n, WFCommn n E P c Q.
```

---

sagl_soundness.v

In this file, we implement the soundness proof for SAGL, as described in Sec. 8.3. Below, we define the semantic sixtuple (Def. 8.1), semantic judgement (Def. 8.13), and parameterized semantic judgement (Def. 8.30) for SAGL. Note that in this implementation, when defining sixtuples, we removed the constraint that a thread tree must be either 0- or 1-atomic, given by Def. 8.1 (item 1), as it is an invariant of the semantics as given by Remark 3.17.

```
Inductive SSConfn: nat -> (Action * Action) -> Assert -> Assert ->
                          Conf -> Assert -> Prop :=
  | SSConfn_O:
      forall A G p_s q_s t s q,
      SSConfn O (A,G) p_s q_s (conf t s) q
  | SSConfn_S:
      forall n A G p_s q_s t s q,
      ~Smlstep (conf t s) abrt ->
      ~Smlstep (conf t s) race ->
      (forall t' s', Smlstep (conf t s) (conf t' s') ->
        (ATOM 0 t ->
          ((ATOM 0 t' -> state.dom s = state.dom s'
                      /\ SSConfn n (A,G) p_s q_s (conf t' s') q) /\
           (ATOM 1 t' -> forall sx' ss', spl sx' ss' s' -> p_s ss' ->
                          SSConfn n (A,G) p_s q_s (conf t' sx') q))) /\
        (ATOM 1 t ->
          ((ATOM 1 t' -> SSConfn n (A,G) p_s q_s (conf t' s') q) /\
           (ATOM 0 t' -> forall s1, p_s s1 ->
                          exists p_s', exists s1', exists s2',
                          aimp2 (crs p_s p_s') G /\
                          aimp p_s' (pog p_s' A) /\
                          spl s' s1' s2' /\ p_s' s1' /\
                          SSConfn n (A,G) p_s' q_s (conf t' s2') q)))) ->
      (t = tcomm skip -> aimp p_s q_s /\ q s) ->
      SSConfn (S n) (A,G) p_s q_s (conf t s) q.
Definition SSConf (E: Action * Action) (p_s q_s: Assert)
                  (k: Conf) (q: Assert): Prop :=
  forall n, SSConfn n E p_s q_s k q.

Definition SSComm (E: Action * Action) (p_s p: Assert)
                  (c: Comm) (q_s q: Assert): Prop :=
  forall s, p s -> SSConf E p_s q_s (conf (tcomm c) s) q.

Definition SSComm_prm (L: Trfrm) (A G: Action) (p_s p: Assert)
                      (c: Comm) (q_s q: Assert): Prop :=
  forall s, dand p_s p s ->
    ~Multi (Prmstep L) (conf (tcomm c) s) abrt /\
    forall s',
      Multi (Prmstep L) (conf (tcomm c) s) (conf (tcomm skip) s') ->
      dand q_s q s'.
```

We establish the auxiliary properties of semantic sixtuples (Lemma 8.2 through Lemma 8.12).

```
Lemma SSConf_n:
  forall n1 n2 E p_s q_s t s q,
  n2 <= n1 ->
  SSConfn n1 E p_s q_s (conf t s) q ->
  SSConfn n2 E p_s q_s (conf t s) q.
Lemma SSConf_cons:
  forall n A A' G' G p_s p_s' q_s' q_s t s q' q,
  aimp2 A A' ->
  aimp2 G' G ->
  aimp p_s p_s' ->
  aimp q_s' q_s ->
  aimp q' q ->
  SSConfn n (A',G') p_s' q_s' (conf t s) q' ->
  SSConfn n (A,G) p_s q_s (conf t s) q.
Lemma SSConf_resr:
  forall n A G p_s q_s t s q,
  (ATOM 0 t \/ ATOM 1 t) ->
  (ATOM 0 t -> exists s1, exists s2, spl s s1 s2 /\ p_s s1 /\
               SSConfn n (A,G) p_s q_s (conf t s2) q) /\
  (ATOM 1 t -> (exists s, p_s s) /\
               SSConfn n (A,G) p_s q_s (conf t s) q) ->
  SSConfn n (crs EMP EMP,crs EMP EMP) EMP EMP (conf t s) (dand q_s q).
Lemma SSConf_fram:
  forall n I A G p_s q_s t s q,
  (ATOM 0 t \/ ATOM 1 t) ->
  Prec I \/ Coin (img G) (dom A) ->
  (ATOM 0 t -> SSConfn n (A,G) p_s q_s (conf t s) q) /\
  (ATOM 1 t -> exists s1, exists s2, spl s s1 s2 /\ I s1 /\
                 SSConfn n (A,G) p_s q_s (conf t s2) q) ->
  SSConfn n (dand2 (crs I I) A,dand2 (crs I I) G)
            (dand I p_s) (dand I q_s) (conf t s) q.
```

```
Lemma SSConf_pfram:
  forall n I A G p_s q_s t s q,
  (exists s1, exists s2, spl s s1 s2 /\ I s1
    /\ SSConfn n (A,G) p_s q_s (conf t s2) q) ->
  SSConfn n (A,G) p_s q_s (conf t s) (dand I q).
Lemma SSConf_conj:
  forall n A1 A2 G1 G2 p_s1 p_s2 t s q_s1 q_s2 q1 q2,
  Coin (img G1) (img G2) ->
  SSConfn n (A1,G1) p_s1 q_s1 (conf t s) q1 ->
  SSConfn n (A2,G2) p_s2 q_s2 (conf t s) q2 ->
  SSConfn n (land2 A1 A2,land2 G1 G2)
            (land p_s1 p_s2) (land q_s1 q_s2) (conf t s) (land q1 q2).
Lemma SSConf_skip:
  forall E q_s s (q : Assert),
  q s ->
  SSConf E q_s q_s (conf (tcomm skip) s) q.
Lemma SSConf_asgn:
  forall lv e s q,
  qoa q (asgn lv e) s ->
  SSConf (EMP2,EMP2) EMP EMP (conf (tcomm (actn (asgn lv e))) s) q.
Lemma SSConf_actn:
  forall a s q,
  qoa q a s ->
  SSConf (EMP2,EMP2) EMP EMP (conf (tcomm (atom (actn a))) s) q.
Lemma SSConf_atom:
  forall n A G (p_s q_s : Assert) c s q,
  aimp2 (crs p_s q_s) G ->
  aimp q_s (pog q_s A) ->
  (forall ss sx, spl sx ss s -> p_s ss ->
    SSConfn n (EMP2,EMP2) EMP EMP (conf (tcomm c) sx) (dand q_s q)) ->
  SSConfn (S n) (A,G) p_s q_s (conf (tcomm (atom c)) s) q.
Lemma SSConf_sequ:
  forall n E t c' t' s p_s p_s' q_s (p' : Assert) q,
  ((exists c, t = tcomm c /\ t' = tcomm (sequ c c')) \/
   (exists t1, exists t2, exists c, t = tpara t1 t2 c
                              /\ t' = tpara t1 t2 (sequ c c')) \/
   (exists t'', exists c, t = tatom t'' c
                    /\ t' = tatom t'' (sequ c c'))) ->
  SSConfn n E p_s p_s' (conf t s) p' ->
  (forall s, p' s -> SSConfn n E p_s' q_s (conf (tcomm c') s) q) ->
  SSConfn n E p_s q_s (conf t' s) q.
Lemma SSConf_cond:
  forall n E p_s q_s b c c' s q,
  SSConfn n E p_s q_s (conf (tcomm c) s) q ->
  (lfb b s ->
     SSConfn n E p_s q_s (conf (tcomm (cond b c c')) s) q) /\
  (lfb (bneg b) s ->
     SSConfn n E p_s q_s (conf (tcomm (cond b c' c)) s) q).
Lemma SSConf_loop:
  forall n A G b c s p_s p,
  aimp p (lor (lfb b) (lfb (bneg b))) ->
  p s ->
  (forall s', land p (lfb b) s' ->
              SSConfn n (A, G) p_s p_s (conf (tcomm c) s') p) ->
  SSConfn n (A, G)
            p_s p_s (conf (tcomm (loop b c)) s) (land p (lfb (bneg b))).
Lemma SSConf_para:
  forall n A1 A2 G p_s1 p_s2 q_s1 q_s2 t1 t2 s s1 s2 q1 q2,
  (ATOM 0 (tpara t1 t2 skip) \/ ATOM 1 (tpara t1 t2 skip)) ->
  spl s s1 s2 ->
  aimp p_s1 (pog p_s1 A1) ->
  aimp p_s2 (pog p_s2 A2) ->
  SSConfn n (A1,land2 G A2) p_s1 q_s1 (conf t1 s1) q1 ->
  SSConfn n (A2,land2 G A1) p_s2 q_s2 (conf t2 s2) q2 ->
  SSConfn n (land2 A1 A2,G) (land p_s1 p_s2) (land q_s1 q_s2)
            (conf (tpara t1 t2 skip) s) (dand q1 q2).
```

We establish the semantic rules (Lemma 8.15 through Lemma 8.28).

```
Lemma SemRule_asgn:
  forall lv e q,
  SSComm (crs EMP EMP,crs EMP EMP)
         EMP (qoa q (asgn lv e)) (actn (asgn lv e)) EMP q.
Lemma SemRule_actn:
  forall a q,
  SSComm (crs EMP EMP,crs EMP EMP) EMP (qoa q a) (atom (actn a)) EMP q.
Lemma SemRule_sequ:
  forall E p_s p c1 c2 q_s q,
  (exists p_s', exists p',
    SSComm E p_s p c1 p_s' p' /\ SSComm E p_s' p' c2 q_s q) ->
  SSComm E p_s p (sequ c1 c2) q_s q.
Lemma SemRule_skip:
  forall E q_s q,
  SSComm E q_s q skip q_s q.
Lemma SemRule_cond:
  forall E p_s p b c1 c2 q_s q,
  aimp p (lor (lfb b) (lfb (bneg b))) ->
  SSComm E p_s (land p (lfb b)) c1 q_s q ->
  SSComm E p_s (land p (lfb (bneg b))) c2 q_s q ->
  SSComm E p_s p (cond b c1 c2) q_s q.
Lemma SemRule_loop:
  forall E p_s p b c,
  aimp p (lor (lfb b) (lfb (bneg b))) ->
  SSComm E p_s (land p (lfb b)) c p_s p ->
  SSComm E p_s p (loop b c) p_s (land p (lfb (bneg b))).
Lemma SemRule_para:
  forall A1 A2 G p_s1 p1 p_s2 p2 c1 c2 q_s1 q1 q_s2 q2,
  aimp p_s1 (pog p_s1 A1) ->
  aimp p_s2 (pog p_s2 A2) ->
  SSComm (A1,land2 G A2) p_s1 p1 c1 q_s1 q1 ->
  SSComm (A2,land2 G A1) p_s2 p2 c2 q_s2 q2 ->
  SSComm (land2 A1 A2,G) (land p_s1 p_s2) (dand p1 p2)
         (para c1 c2) (land q_s1 q_s2) (dand q1 q2).
Lemma SemRule_atom:
  forall A G p_s p c q_s q,
  SSComm (crs EMP EMP,crs EMP EMP) EMP (dand p_s p) c EMP (dand q_s q) ->
  aimp2 (crs p_s q_s) G ->
  aimp q_s(pog q_s A) ->
  SSComm (A,G) p_s p (atom c) q_s q.
Lemma SemRule_cons:
  forall A G p_s p c q_s q,
  (exists A', exists G', exists p_s', exists p', exists q_s', exists q',
    aimp2 A A' /\ aimp p_s p_s' /\ aimp p p' /\
    SSComm (A',G') p_s' p' c q_s' q' /\ aimp2 G' G /\ aimp q_s' q_s /\
    aimp q' q) ->
  SSComm (A,G) p_s p c q_s q.
Lemma SemRule_exst:
  forall X A G p_s P c q_s Q,
  (forall x: X, SSComm (A,G) p_s (P x) c q_s (Q x)) ->
  SSComm (A,G) p_s (lex P) c q_s (lex Q).
Lemma SemRule_fram:
  forall I A G p_s p c q_s q,
  Prec I \/ Coin (img G) (dom A) ->
  SSComm (A,G) p_s p c q_s q ->
  SSComm (dand2 (crs I I) A,dand2 (crs I I) G)
         (dand I p_s) p c (dand I q_s) q.
Lemma SemRule_pfram:
  forall I A G p_s p c q_s q,
  SSComm (A,G) p_s p c q_s q ->
  SSComm (A,G) p_s (dand I p) c q_s (dand I q).
Lemma SemRule_resr:
  forall A G p_s p c q_s q,
  SSComm (A,G) p_s p c q_s q ->
  SSComm (crs EMP EMP,crs EMP EMP) EMP (dand p_s p) c EMP (dand q_s q).
Lemma SemRule_conj:
  forall A1 A2 G1 G2 p_s1 p_s2 p1 p2 c q_s1 q_s2 q1 q2,
  Coin (img G1) (img G2) ->
  SSComm (A1,G1) p_s1 p1 c q_s1 q1 ->
  SSComm (A2,G2) p_s2 p2 c q_s2 q2 ->
  SSComm (land2 A1 A2,land2 G1 G2)
         (land p_s1 p_s2) (land p1 p2) c (land q_s1 q_s2) (land q1 q2).
Lemma SemRule_disj:
  forall A G p_s q_s p1 p2 c q1 q2,
  SSComm (A,G) p_s p1 c q_s q1 ->
  SSComm (A,G) p_s p2 c q_s q2 ->
  SSComm (A,G) p_s (lor p1 p2) c q_s (lor q1 q2).
```

And, finally, we establish the soundness with regard to the interleaved (Theorem 8.29 and Lemma 8.14), parameterized (Theorem 8.31), and relaxed semantics (Theorem 8.32).

```
Theorem Soundness:
  forall E p_s p c q_s q,
  WFComm E (p_s,p) c (q_s,q) ->
  SSComm E p_s p c q_s q.

Lemma Explicit_Soundness:
  forall E p_s p c s q_s q,
  SSComm E p_s p c q_s q ->
  dand p_s p s ->
  ~Multi Smlstep (conf (tcomm c) s) abrt /\
  ~Multi Smlstep (conf (tcomm c) s) race /\
  (forall s', Multi Smlstep (conf (tcomm c) s) (conf (tcomm skip) s') ->
             dand q_s q s').

Theorem Soundness_prm:
  forall L A G p_s p c q_s q,
  WFComm (A,G) (p_s,p) c (q_s,q) ->
  DRF L ->
  SSComm_prm L A G p_s p c q_s q.

Theorem Soundness_rlx:
  forall A G p_s p c q_s q,
  WFComm (A,G) (p_s,p) c (q_s,q) ->
  SSComm_prm Subsump A G p_s p c q_s q.
```

---

csl_in_sagl.v

In this file, we establish that SAGL is a generalization of CSL, by giving a translation of a CSL judgment into a SAGL judgement (Fig. 8.4) and proving the CSL rules as lemmas (Lemma 8.37 through Lemma 8.50).

```
Definition WFComm (I p: Assert) (c: Comm) (q: Assert): Prop :=
  sagl.WFComm (crs I I,crs I I) (I,p) c (I,q).

Lemma Rule_asgn: forall lv e q,
                 WFComm EMP (qoa q (asgn lv e)) (actn (asgn lv e)) q.
Lemma Rule_actn: forall a q,
                 WFComm EMP (qoa q a) (atom (actn a)) q.
Lemma Rule_sequ: forall I p c1 c2 q,
                 (exists p', WFComm I p c1 p' /\ WFComm I p' c2 q) ->
                 WFComm I p (sequ c1 c2) q.
Lemma Rule_skip: forall I p,
                 WFComm I p skip p.
Lemma Rule_cond: forall I p b c1 c2 q,
                 aimp p (lor (lfb b) (lfb (bneg b))) ->
                 WFComm I (land p (lfb b)) c1 q ->
                 WFComm I (land p (lfb (bneg b))) c2 q ->
                 WFComm I p (cond b c1 c2) q.
Lemma Rule_loop: forall I p b c,
                 aimp p (lor (lfb b) (lfb (bneg b))) ->
                 WFComm I (land p (lfb b)) c p ->
                 WFComm I p (loop b c) (land p (lfb (bneg b))).
Lemma Rule_para: forall I p1 p2 c1 c2 q1 q2,
                 WFComm I p1 c1 q1 ->
                 WFComm I p2 c2 q2 ->
                 WFComm I (dand p1 p2) (para c1 c2) (dand q1 q2).
Lemma Rule_atom: forall I p c q,
                 WFComm EMP (dand I p) c (dand I q) ->
                 WFComm I p (atom c) q.
Lemma Rule_cons: forall I p c q,
                 (exists p', exists q', aimp p p'
                              /\ WFComm I p' c q' /\ aimp q' q) ->
                 WFComm I p c q.
Lemma Rule_fram: forall I' I p c q,
                 Prec I' \/ Prec I ->
                 WFComm I p c q ->
                 WFComm (dand I' I) p c q.
Lemma Rule_resr: forall I p c q,
                 WFComm I p c q ->
                 WFComm EMP (dand I p) c (dand I q).
Lemma Rule_conj: forall I p1 p2 c q1 q2,
                 Prec I ->
                 WFComm I p1 c q1 ->
                 WFComm I p2 c q2 ->
                 WFComm I (land p1 p2) c (land q1 q2).
Lemma Rule_conj2: forall I1 I2 p1 p2 c q1 q2,
                  Coin I1 I2 ->
                  WFComm I1 p1 c q1 ->
                  WFComm I2 p2 c q2 ->
                  WFComm (land I1 I2) (land p1 p2) c (land q1 q2).
Lemma Rule_disj: forall I p1 p2 c q1 q2,
                 WFComm I p1 c q1 ->
                 WFComm I p2 c q2 ->
                 WFComm I (lor p1 p2) c (lor q1 q2).
```

Bibliography

[1] Advanced Micro Devices. *AMD64 Architecture Programmer's Manual, Volume 2: System Programming*, Sep. 2007.

[2] S. Adve. *Designing Memory Consistency Models for Shared-Memory Multiprocessors*. PhD thesis, University of Wisconsin-Madison, Dec. 1993.

[3] S. Adve and K. Gharachorloo. Shared memory consistency models: A tutorial. *IEEE Computer*, 29(12):66–76, Dec. 1996.

[4] S. Adve and M. Hill. Weak ordering — a new definition. In *17th ISCA*, pages 2–14, Seattle, Washington, May 1990.

[5] S. Adve and M. Hill. A unified formalization of four shared-memory models. *IEEE Transactions on Parallel and Distributed Systems*, 4(6):613–624, Jun. 1993.

[6] S. Adve, V. Pai, and P. Ranganathan. Recent advances in memory consistency models for hardware shared memory systems. *Proceedings of the IEEE, Special Issue on Distributed Shared Memory*, 87(3):445–455, Mar. 1999.

[7] M. Ahamad, R. Bazzi, R. John, P. Kohli, and G. Neiger. The power of processor consistency. In *5th ACM Symposium on Parallel Algorithms and Architectures*, pages 251–260, Velen, Germany, Jul. 1993.

[8] M. Ahamad, G. Neiger, J. Burns, P. Kohli, and P. Hutto. Causal memory: Definitions, implementation and programming. *Distributed Computing*, 9(1):37–49, Mar. 1995.

[9] Arvind and J.-W. Maessen. Memory model = instruction reordering + store atomicity. In *ISCA*, pages 29–40, Washington, DC, May 2006.

[10] D. Aspinall and J. Ševčík. Formalising java's data-race-free guarantee. In *20th TPHOLS*, pages 22–37, Kaiserslautern, Germany, Sep. 2007.

[11] D. Aspinall and J. Ševčík. Java memory model examples: Good, bad and ugly. In *VAMP*, Lisbon, Sep. 2007.

[12] H. Boehm and S. Adve. The foundations of the C++ concurrency memory model. In *PLDI*, pages 68–78, Tucson, Arizona, Jun. 2008.

[13] H.-J. Boehm. Threads cannot be implemented as a library. In *PLDI*, pages 261–268, Chicago, Jun. 2005.

[14] R. Bornat, C. Calcagno, P. O'Hearn, and M. Parkinson. Permission accounting in separation logic. In *32nd POPL*, pages 259–270, Long Beach, California, Jan. 2005.

[15] G. Boudol and G. Petri. Relaxed memory models: an operational approach. In *36th POPL*, pages 392–403, Savannah, Georgia, USA, Jan. 2009.

[16] G. Bronevetsky and B. de Supinski. Complete formal specification of the OpenMP memory model. *International Journal of Parallel Programming*, 35(4):335–392, Jul. 2007.

[17] S. Brookes. A grainless semantics for parallel programs with shared mutable data. *Theoretical Comp. Sci.*, 155:277–307, May 2006.

[18] S. Brookes. A semantics for concurrent separation logic. *Theoretical Comp. Sci.*, 375(1–3):227–270, May 2007.

[19] S. Burckhardt, R. Alur, and M. Martin. Checkfence: Checking consistency of concurrent data types on relaxed memory models. In *PLDI*, pages 12–21, San Diego, California, Jun. 2007.

[20] C. Calcagno, P. O'Hearn, and H. Yang. Local action and abstract separation logic. In *22nd LICS*, pages 366–378, Wroclaw, Poland, July 2007.

[21] P. Cenciarelli, A. Knapp, and E. Sibilio. The java memory model: Operationally, denotationally, axiomatically. In *ESOP*, pages 331–346, Braga, Mar. 2007.

[22] A. Condon, M. Hill, M. Plakal, and D. Sorin. Using lamport clocks to reason about relaxed memory models. In *5th International Symposium on High Performance Computer Architecture*, pages 270–278, Washington, DC, Jan. 1999.

[23] A. de Melo. Defining uniform and hybrid memory consistency models on a unified framework. In *32nd Annual Hawaii International Conference on System Sciences, Volume 8*, Maui, Hawaii, Jan. 1999.

[24] E. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages*, pages 43–112. Academic Press, London, 1968.

[25] D. Dill, S. Park, and A. Nowatzyk. Formal specification of abstract memory models. In *Research on Integrated Systems*, pages 38–52, Seattle, Washington, Mar. 1993.

[26] M. Dubois, C. Scheurich, and F. Briggs. Memory access buffering in multiprocessors. In *13th ISCA*, pages 434–442, Tokyo, Jun. 1986.

[27] M. Dubois, C. Scheurich, and F. Briggs. Synchronization, coherence, and event ordering. *IEEE Computer*, 21(2):9–21, Feb. 1988.

[28] X. Fang, J. Lee, and S. Midkiff. Automatic fence insertion for shared memory multiprocessing. In *17th International Conference on Supercomputing*, pages 285–294, San Francisco, California, Jun. 2003.

[29] X. Feng. Local rely-guarantee reasoning. In *36th POPL*, pages 315–327, Savannah, Georgia, USA, Jan. 2009.

[30] X. Feng, R. Ferreira, and Z. Shao. On the relationship between concurrent separation logic and assume-guarantee reasoning. In *ESOP*, pages 173–188, Braga, Mar. 2007.

[31] X. Feng and Z. Shao. Modular verification of concurrent assembly code with dynamic thread creation and termination. In *ICFP*, pages 254–267, Tallinn, Estonia, Sep. 2005.

[32] G. Gao and V. Sarkar. Location consistency – a new memory model and cache consistency protocol. *IEEE TC*, 49(8):798–813, Aug. 2000.

[33] P. Gastin and M. Mislove. A truly concurrent semantics for a simple parallel programming language. In *8th Annual Conference of the EACSL on Computer Science Logic*, pages 515–529, Madrid, Sep. 1999.

[34] K. Gharachorloo. *Memory Consistency Models for Shared-Memory Multiprocessors*. PhD thesis, Department of Electrical Engineering, Stanford University, Dec. 1995.

[35] K. Gharachorloo, S. Adve, A. Gupta, J. Hennessy, and M. Hill. Programming for different memory consistency models. *Journal of Parallel and Distributed Computing*, 15(4):399–407, Aug. 1992.

[36] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. *SIGARCH News*, 18(3):15–26, Jun. 1990.

[37] J. Goodman. Cache consistency and sequential consistency. Technical Report 61, IEEE Scalable Coherence Interface Committee, Mar. 1989.

[38] D. Grossman, J. Manson, and W. Pugh. What do high-level memory models mean for transactions? In *Workshop on Memory System Performance and Correctness*, pages 62–69, San Jose, Jun. 2006.

[39] N. Hamid, Z. Shao, V. Trifonov, S. Monnier, and Z. Ni. A syntactic approach to foundational proof carrying-code. *Journal of Automated Reasoning (Special issue on Proof-Carrying Code)*, 31(3–4):191–229, Dec. 2003.

[40] L. Higham, J. Kawash, and N. Verwaal. Defining and comparing memory consistency models. In *International Conference on Parallel and Distributed Computing Systems*, pages 349–356, New Orleans, Louisiana, Oct. 1997.

[41] M. Hill. Multiprocessors should support simple memory-consistency models. *IEEE Computer*, 31(8):28–34, Aug. 1998.

[42] A. Hobor, A. Appel, and F. Nardelli. Oracle semantics for concurrent separation logic. In *ESOP*, pages 353–367, Budapest, Hungary, Mar. 2008.

[43] M. Huisman and G. Petri. The java memory model: A formal explanation. In *VAMP*, Lisbon, Sep. 2007.

[44] Intel Corporation. *A Formal Specification of Intel Itanium Processor Family Memory Ordering*, Oct. 2002.

[45] Intel Corporation. *Intel 64 Architecture Memory Ordering White Paper*, Aug. 2007.

[46] R. Jagadeesan, C. Pitcher, and J. Riely. Generative operational semantics for relaxed memory models. In *19th ESOP*, pages 307–326, Paphos, Cyprus, Mar. 2010.

[47] Java Community Process. *JSR-133: Java Memory Model and Thread Specification*, Aug. 2004.

[48] C. Jones. Tentative steps toward a development method for interfering programs. *TOPLAS*, 5(4):596–619, Oct. 1983.

[49] A. Krishnamurthy and K. Yelick. Optimizing parallel programs with explicit synchronization. In *PLDI*, pages 196–204, La Jolla, California, Jun. 1995.

[50] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, Jul. 1978.

[51] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE TC*, 28(9):690–691, Sep. 1979.

[52] X. Leroy. Formal certification of a compiler back-end, or: Programming a compiler with a proof assistant. In *POPL*, pages 42–54, Charleston, South Carolina, Jan. 2006.

[53] R. Lipton and J. Sandberg. PRAM: A scalable shared memory. Technical Report CS-TR-180-88, Department of Computer Science, Princeton University, Sep. 1988.

[54] J. Manson, W. Pugh, and S. Adve. The java memory model. In *32nd POPL*, pages 378–391, Long Beach, California, Jan. 2005.

[55] M. Michael and M. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In *15th ACM Symposium on Principles of Distributed Computing*, pages 267–275, Philadelphia, Pennsylvania, May 1996.

[56] S. Midkiff, J. Lee, and D. Padua. A compiler for multiple memory models. *Concurrency and Computation: Practice and Experience*, 16(2–3):197–220, Mar. 2004.

[57] D. Mosberger. Memory consistency models. *Operating Systems Review*, 27(1):18–26, Jan. 1993.

[58] P. O'Hearn. Resources, concurrency, and local reasoning. *Theoretical Comp. Sci.*, 375(1–3):271–307, May 2007.

[59] S. Owens, S. Sarkar, and P. Sewell. A better x86 memory model: x86-TSO. In *22nd TPHOLS*, pages 391–407, Munich, Germany, Aug. 2009.

[60] S. Park and D. Dill. An executable specification and verifier for relaxed memory order. *IEEE TC*, 48(2):227–235, Feb. 1999.

[61] M. Parkinson, R. Bornat, and C. Calcagno. Variables as resources in hoare logics. In *LICS*, pages 137–146, Seattle, Washington, Aug. 2006.

[62] J. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74, Copenhagen, Jul. 2002.

[63] J. Reynolds. Towards a grainless semantics for shared-variable concurrency. In *FSTTCS*, pages 37–48, Chennai, India, Dec. 2004.

[64] A. Roychoudhury. Formal reasoning about hardware and software memory models. In *4th International Conference on Formal Engineering Methods*, pages 423–434, Shanghai, China, Oct. 2002.

[65] V. Saraswat, R. Jagadeesan, M. Michael, and C. von Praun. A theory of memory models. In *12th PPoPP*, pages 161–172, San Jose, Mar. 2007.

[66] S. Sarkar, P. Sewell, F. Nardelli, S. Owens, T. Ridge, T. Braibant, M. Myreen, and J. Alglave. The semantics of x86-cc multiprocessor machine code. In *36th POPL*, pages 379–391, Savannah, Georgia, USA, Jan. 2009.

[67] D. Shasha and M. Snir. Efficient and correct execution of parallel programs that share memory. *TOPLAS*, 10(2):282–312, Apr. 1988.

[68] X. Shen, Arvind, and L. Rudolph. Commit-reconcile & fences (CRF): A new memory model for architects and compiler writers. In *26th ISCA*, pages 150–161, Atlanta, Georgia, May 1999.

[69] R. Steinke and G. Nutt. A unified theory of shared memory consistency. *Journal of the ACM*, 51(5):800–849, Jun. 2004.

[70] Z. Sura, C.-L. Wong, X. Fang, J. Lee, S. Midkiff, and D. Padua. Automatic implementation of programming language consistency models. In *15th Workshop on Languages and Compilers for Parallel Computing*, pages 172–187, College Park, Maryland, Jul. 2002.

[71] V. Vafeiadis and M. Parkinson. A marriage of rely/guarantee and separation logic. In *CONCUR*, pages 256–271, Lisbon, Sep. 2007.

[72] Q. Xu, W. de Roever, and J. He. The rely-guarantee method for veryfing shared variable concurrent programs. *Formal Aspects of Computing*, 9(2):149–174, 1997.

[73] H. Yang and P. O'Hearn. A semantic basis for local reasoning. In *5th FOSSACS*, pages 402–416, Grenoble, France, Apr. 2002.

[74] Y. Yang. *Formalizing Shared-Memory Consistency Models for Program Analysis*. PhD thesis, School of Computing, University of Utah, May 2005.

[75] Y. Yang, G. Gopalakrishnan, and G. Lindstrom. Specifying java thread semantics using a uniform memory model. In *Java Grande Conference*, pages 192–201, Seattle, Washington, Nov. 2002.

[76] Y. Yang, G. Gopalakrishnan, G. Lindstrom, and K. Slind. Analyzing the Intel Itanium memory ordering rules using logic programming and SAT. In *12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 81–95, L'Aquila, Italy, Oct. 2003.

[77] Y. Yang, G. Gopalakrishnan, G. Lindstrom, and K. Slind. Nemos: A framework for axiomatic and executable specifications of memory consistency models. In *18th International Parallel and Distributed Processing Symposium*, Santa Fe, New Mexico, Apr. 2004.

[78] D. Yu, N. Hammid, and Z. Shao. Building certified libraries for pcc: Dynamic storage allocation. In *ESOP*, pages 363–379, Warsaw, Apr. 2003.

[79] D. Yu and Z. Shao. Verification of safety properties for concurrent assembly code. In *ICFP*, pages 175–188, Snowbird, Utah, Sep. 2004.