Taming Release-Acquire Consistency

Ori Lahav Nick Giannarakis Viktor Vafeiadis
Max Planck Institute for Software Systems (MPI-SWS), Germany
{orilahav,nickgian,viktor}@mpi-sws.org

Abstract

We introduce a strengthening of the release-acquire fragment of the C11 memory model that (i) forbids dubious behaviors that are not observed in any implementation; (ii) supports fence instructions that restore sequential consistency; and (iii) admits an equivalent intuitive operational semantics based on point-to-point communication. This strengthening has no additional implementation cost: it allows the same local optimizations as C11 release and acquire accesses, and has exactly the same compilation schemes to the x86-TSO and Power architectures. In fact, the compilation to Power is complete with respect to a recent axiomatic model of Power; that is, the compiled program exhibits exactly the same behaviors as the source one. Moreover, we provide criteria for placing enough fence instructions to ensure sequential consistency, and apply them to an efficient RCU implementation.

Categories and Subject Descriptors D.1.3 [*Concurrent Programming*]: Parallel programming; D.3.1 [*Programming Languages*]: Formal Definitions and Theory; D.3.3 [*Programming Languages*]: Language Constructs and Features; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs

Keywords Weak memory model; release-acquire; C11; operational semantics

1. Introduction

Weak memory models for programming languages formalize the set of behaviors that multithreaded programs may exhibit taking into account the effect of both the hardware architectures and compiler optimizations. An important such model is the C11 model introduced in the 2011 revisions of the C and C++ standards [14, 15].

C11 provides several kinds of memory accesses, each having a different implementation cost and providing different synchronization guarantees. At one end of the spectrum, non-atomic accesses provide absolutely no guarantees in case of racy access (they are considered programming errors). At the other end, sequentially consistent accesses are globally synchronized following the simple and well-known model of *sequential consistency* (SC) [19]. Between these two extremes, the most useful access kinds are *release* stores and *acquire* loads that strike a good balance between performance and programmability.

While the full C11 model suffers from some problems (such as "out-of-thin-air executions" [8, 38]), its *release-acquire* fragment, obtained by restricting all writes and reads to be release and acquire accesses respectively, constitutes a particularly useful and (relatively) well-behaved model. This fragment, hereinafter referred to as RA, provides much weaker guarantees than SC, that allow high performance implementations, and still suffice for fundamental concurrent algorithms (e.g., a variant of the read-copy-update synchronization mechanism [13] presented in the sequel). To understand RA, consider the following two programs:

Store buffering (SB)	**Message passing (MP)**
Initially, $x = y = 0$.	Initially, $x = y = 0$.
$x := 1 \;\Big\|\; y := 1$ $\texttt{wait}\,(y = 0) \;\Big\|\; \texttt{wait}\,(x = 0)$	$x := 1 \;\Big\|\; \texttt{wait}\,(y = 1)$ $y := 1 \;\Big\|\; \texttt{wait}\,(x = 0)$
May terminate under RA.	Never terminates under RA.

RA is designed to support "message passing", a common idiom for relatively cheap synchronization between threads. On the other hand, to allow efficient implementations, a non-SC behavior is allowed in the case of the "store buffering" program. An intuitive (but incomplete) explanation of these examples can be given in terms of *reorderings* (performed by the hardware and/or the compiler): consecutive *write and read* accesses may be arbitrarily reordered, but reordering of *two reads* or *two writes* is disallowed. This easily accounts for the non-SC result of the SB program, and explains why MP does not expose undesirable behaviors.

The precise formulation of RA is *declarative* (also known as *axiomatic*): the model associates a set of graphs (called: *executions*) to every program, and filters out disallowed executions by imposing certain formal restrictions. Roughly speaking, it requires that every read is justified by a corresponding write, that cannot appear later according to the *program order*. This induces a *happens-before* relation on the memory accesses, that, assuming only release/acquire accesses, is taken to be the union of the program order and the *reads-from* justification edges. An additional condition is needed to ensure that reads cannot observe overwritten values. For that, the model asserts the existence of a per-location *modification order* that orders write accesses, and requires that reads are justified only by writes that happened before and are maximal according to the modification order.

The RA model is appropriate as a rigorous foundation that is not tied to a particular compiler and architecture. Further, C11 in general, and RA in particular, has verified compilation schemes to the x86-TSO and Power/ARM architectures [6, 7, 29]. However, RA suffers from three problems.

Problem 1: Unobservable relaxed behaviors. First, there is a mismatch between the declarative model and the intuitive reordering account. The following program (again, assuming all accesses are release/acquire) may terminate under the formal model if the

$x := 2$ and $y := 2$ writes are placed before the $x := 1$ and $y := 1$ writes in the respective modification orders.

$$\begin{array}{c} x := 1 \;\|\; y := 1 \\ y := 2 \;\|\; x := 2 \\ \texttt{wait}\ (x = 1 \wedge y = 1) \end{array} \qquad \text{(2+2W)}$$

Disallowing reordering of consecutive writes should, however, forbid this behavior. Furthermore, this behavior is not observable on any known hardware under any known sound compilation scheme of the C11 release writes.

Problem 2: Overly weak SC fences. C11 provides a memory fence construct, called *SC fences*, that enforces stronger ordering guarantees among memory accesses. Despite their name, however, C11's semantics for these fences is overly weak and fails to ensure sequential consistency when used in conjunction with release-acquire accesses. For example, consider the following program:

$$x := 1 \;\Big\|\; y := 1 \;\Big\|\; \begin{array}{l}\texttt{wait}\ (x = 1) \\ \texttt{wait}\ (y = 0)\end{array} \;\Big\|\; \begin{array}{l}\texttt{wait}\ (y = 1) \\ \texttt{wait}\ (x = 0)\end{array} \qquad \text{(IRIW)}$$

Assuming that initially $x = y = 0$, this program may terminate under RA. Furthermore, according to the C11 semantics, inserting SC fence instructions between the two waits on the reader threads does not rule out this weak behavior. Again, this is a result of too liberal specification: the compilation of SC fences generates hardware fence instructions (`mfence` on x86-TSO, `sync` on Power, `dmb` on ARM, `mf` on Itanium) that forbid such weak behaviors.[1]

Problem 3: No intuitive operational semantics. The axiomatic nature of RA is also a drawback: global restrictions on executions do not allow the traditional understanding of programs that simulates their step-by-step progress. Thus, both programming and developing techniques for reasoning about programs are substantially more difficult with a purely declarative memory model. For these tasks, a more *operational* approach can be very helpful. The most well-known example is, of course, SC, that is naturally modeled by a machine with one global shared memory and interleaved accesses.

For weak models, *total store ordering* (TSO) [25, 33] provides a good case in point. This weak memory model emerged from a concrete operational model provided by the SPARC and x86 architectures. In addition to a main memory, TSO-machines have per-processor store buffers, where write operations are enqueued, and non-deterministically propagate to the main memory. Based on this operational model, several reasoning methods and analysis techniques have been developed for TSO. This includes some of the most useful and well-studied verification techniques: model checking and abstract interpretation (see, e.g., [1, 5, 12, 17, 20]), program logics (see, e.g., [27, 32]), simulations for verifying program transformations (see, e.g., [16, 22, 30, 37]), and reductions to SC (see, e.g., [2, 9, 10, 24]).

In this paper, we attempt to overcome these drawbacks. We propose a simple strengthening of RA that solves problems 1 and 3, and an alternative modeling of SC fences to solve problem 2. Next, we briefly overview each of these contributions.

Strong release-acquire. We introduce a model, called SRA (for "Strong RA"), that strengthens RA by requiring the union of the per-location modification orders and the happens-before relation to be acyclic. As a result, SRA forbids behaviors that require reordering of two writes as in the (2+2W) example, but coincides with RA for programs containing no write-write races.

[1] Specifically for the IRIW program, we note that on x86-TSO and Itanium, its weak behavior is forbidden even without a fence. For Itanium, see `ftp://download.intel.com/design/Itanium/Downloads/25142901.pdf`, §3.3.5.

More importantly, alongside the declarative definition, SRA has an intuitive operational presentation, that, like the operational semantics of TSO, does not refer at all to formal restrictions on graphs and partial orders. Unlike TSO, SRA-machines are based on point-to-point communication: processors have local memories and they communicate with one another via ordered message buffers using global timestamps to order different writes to the same location. This operational model is not meant to mimic any real hardware implementation, but to serve as a formal foundation for understanding and reasoning about concurrent programs under SRA.

Moreover, we show that SRA allows the same program transformations as RA and has the same implementation cost over x86-TSO and Power. For the latter, we prove that the existing compilation schemes for the C11 release-acquire accesses to TSO and Power guarantee the stronger specification of SRA. In fact, under the declarative Power/ARM memory model of Alglave et al. [4] the compilation to Power corresponds exactly to SRA, which means that SRA cannot further be strengthened without performance implications. In contrast, TSO is strictly stronger than SRA (the difference can be observed even with two threads). Nevertheless, for a restricted class of programs following a certain client-server communication discipline, we prove that TSO and SRA coincide.

Stronger semantics for SC fences. To address problem 2 above, we suggest a new semantics for SC fences. Our key idea is to model SC fence commands using an existing mechanism: as if they were atomic (acquire-release) update commands to a special, otherwise unused, location with an arbitrary value. We show that inserting fences between every two potentially racy RA accesses in each thread restores SC. For a particular large and common class of programs, it suffices to have a fence between every potentially racy *write* and subsequent potentially racy *read*. To demonstrate the usefulness of this criterion, we show how it can be utilized to reduce reasoning about the correctness of an RA-based RCU implementation down to SC.

Moreover, we show that this modeling of SC fences bears no additional implementation cost, and again one can follow the existing compilation scheme used for SC fences to TSO and Power. In fact, we prove that the semantics of Power's `sync` instruction is *equivalent* to our modeling of SC fences as acquire-release updates.

The rest of this paper is organized as follows: §2 reviews the RA memory model, §3 presents our declarative SRA model, §4 provides an equivalent operational model for SRA, §5 discusses fences and reduction theorems to SC, §6 shows that TSO is stronger than SRA and presents conditions under which they are equivalent, §7 proves that Power's `sync` fences are equivalent to release-acquire updates to a distinguished location and that SRA and its compilation to Power are equivalent, §8 discusses related work, and §9 concludes.

Supplementary material including full proofs as well as Coq proof scripts is available at: `http://plv.mpi-sws.org/sra/`. Except for the results in §4, all propositions and theorems of this paper have been proved in Coq with minor presentational differences.

2. RA Memory Model

In this section, we present RA, the declarative model behind the release-acquire fragment of C11's memory model [6]. The basic approach in the C11 formalization is to define the semantics of a program P to be the set of *consistent executions* of P. For the simplicity of the presentation, we employ a simplified programming language. While our notations are slightly different, the declarative semantics presented in this section corresponds exactly to the semantics of C11 programs from [6] in which all reads are acquire

$$\frac{}{\texttt{skip}; c \to c} \qquad \frac{c_1 \xrightarrow{l} c_1'}{c_1; c_2 \xrightarrow{l} c_1'; c_2} \qquad \frac{c_1 \to c_1'}{c_1; c_2 \to c_1'; c_2} \qquad \frac{y \in \mathit{fv}[e] \qquad l = \langle \texttt{R}, y, v\rangle}{x := e \xrightarrow{l} x := e\{v/y\}}$$

$$\frac{\mathit{fv}[e] = \emptyset \qquad l = \langle \texttt{W}, x, [\![e]\!]\rangle}{x := e \xrightarrow{l} \texttt{skip}} \qquad \frac{l = \langle \texttt{U}, x, v, [\![e]\!](v)\rangle}{\langle x := e(x)\rangle \xrightarrow{l} \texttt{skip}}$$

$$\frac{y \in \mathit{fv}[e] \qquad l = \langle \texttt{R}, y, v\rangle}{\texttt{if } e \texttt{ then } c \texttt{ else } c' \xrightarrow{l} \texttt{if } e\{v/y\} \texttt{ then } c \texttt{ else } c'} \qquad \frac{\mathit{fv}[e] = \emptyset \qquad [\![e]\!] \neq 0}{\texttt{if } e \texttt{ then } c \texttt{ else } c' \to c} \qquad \frac{\mathit{fv}[e] = \emptyset \qquad [\![e]\!] = 0}{\texttt{if } e \texttt{ then } c \texttt{ else } c' \to c'}$$

$$\frac{}{\texttt{repeat } c \texttt{ until } e \to c; \texttt{if } e \texttt{ then } (\texttt{repeat } c \texttt{ until } e) \texttt{ else skip}}$$

$$\frac{l = \langle \texttt{R}, x, v\rangle \qquad [\![e]\!](v) = 0}{\texttt{when } e(x) \texttt{ do } x := e'(x) \xrightarrow{l} \texttt{when } e(x) \texttt{ do } x := e'(x)} \qquad \frac{l = \langle \texttt{U}, x, v, [\![e']\!](v)\rangle \qquad [\![e]\!](v) \neq 0}{\texttt{when } e(x) \texttt{ do } x := e'(x) \xrightarrow{l} \texttt{skip}}$$

Figure 1. Command steps.

$$\frac{P(i) \xrightarrow{l} c}{P \xrightarrow{l,i} P[i \mapsto c]} \qquad \frac{P(i) \to c}{P \to P[i \mapsto c]}$$

Figure 2. Program steps

$$\frac{\mathit{lab}(a) = l \qquad \mathit{tid}(a) = i}{G \xrightarrow{l,i} G; a}$$

Figure 3. Execution steps

$$\frac{P \xrightarrow{l,i} P' \qquad G \xrightarrow{l,i} G'}{\langle P, G\rangle \to \langle P', G'\rangle} \qquad \frac{P \to P'}{\langle P, G\rangle \to \langle P', G\rangle}$$

Figure 4. Program and execution combined steps

reads, writes are release writes, and atomic updates are acquire-release read-modify-writes (RMWs).

Basic notations. Given a relation R on a set A, $R^?$, R^+, and R^* respectively denote its reflexive, transitive, and reflexive-transitive closures. The inverse relation of R is denoted by R^{-1}. When R is a strict partial order, a pair $\langle a, b\rangle \in R$ is called an *immediate R-edge* if b immediately follows a in R, that is: no $c \in A$ satisfies both $\langle a, c\rangle \in R$ and $\langle c, b\rangle \in R$. We denote by $R_1; R_2$ the left composition of two relations R_1, R_2. Finally, Id_A denotes the identity relation on the set A.

A simplified programming language. We assume a finite set Loc of locations, a finite set Val of values with a distinguished value $0 \in \mathsf{Val}$, and any standard interpreted language for expressions containing at least all locations and values. We use x, y, z as metavariables for locations, v for values, e for expressions, and denote by $\mathit{fv}[e]$ the set of locations that appear free in e. The sequential fragment of the language is given by the following grammar:

$$c ::= \texttt{skip} \mid \texttt{if } e \texttt{ then } c \texttt{ else } c \mid \texttt{repeat } c \texttt{ until } e \mid \texttt{wait } e \mid c\,;c \mid x := e \mid \langle x := e(x)\rangle \mid \texttt{when } e(x) \texttt{ do } x := e'(x)$$

All commands are standard. The command $\texttt{wait } e$ is a syntactic sugar for $\texttt{repeat skip until } e$. The command $\langle x := e(x)\rangle$ is an atomic assignment corresponding to a primitive RMW instruction and, as such, mentions only one location. The command $\texttt{when } e(x) \texttt{ do } x := e'(x)$ corresponds to a compare-and-swap loop, that loops until the value v of x satisfies $[\![e]\!](v) \neq 0$, and then atomically assigns $[\![e']\!](v)$ to x.

To define multithreaded programs, we assume a constant number N of threads with thread identifiers being $1, \ldots, N$, and take a program P to be a function that assigns a command c to every thread identifier. We use i, j as metavariables for thread identifiers.

Executions. We employ the following terminology:

- A *type* is either $\texttt{R}$ ("Read"), $\texttt{W}$ ("Write"), or $\texttt{U}$ ("Update"). We use $\texttt{T}$ as a metavariable for types.
- A *label* is either a triple of the form $\langle \texttt{R}, x, v_r\rangle$, a triple of the form $\langle \texttt{W}, x, v_w\rangle$, or a quadruple of the form $\langle \texttt{U}, x, v_r, v_w\rangle$. We denote by Lab the set of all labels.
- An *event* is a tuple of the form $\langle k, i, l\rangle$, where k is an event identifier (natural number), i is a thread identifier (natural number), and l is a label. The functions id, tid, lab, typ, loc, val_r, and val_w respectively return (when applicable) the k, i, l, $\texttt{T}$, x, v_r and v_w components of an event (or its label). We denote by $\mathcal{A}$ the set of all events, while $\mathcal{R}, \mathcal{W}$, and $\mathcal{U}$ respectively denote the set of events $a \in \mathcal{A}$ with $\mathit{typ}(a)$ being $\texttt{R}, \texttt{W}$, or $\texttt{U}$.
- A *reads-from edge* is a pair $\langle a, b\rangle \in (\mathcal{W} \cup \mathcal{U}) \times (\mathcal{R} \cup \mathcal{U})$ satisfying $\mathit{loc}(a) = \mathit{loc}(b)$ and $\mathit{val}_w(a) = \mathit{val}_r(b)$.

An *execution* G is a triple $\langle A, \mathit{po}, \mathit{rf}\rangle$ where:

- $A \subseteq \mathcal{A}$ is a finite set of events. We denote by $G.\texttt{T}_x$ the set of events $a \in A$ for which $\mathit{typ}(a) = \texttt{T}$ and $\mathit{loc}(a) = x$, while $G.\texttt{T}$ denotes the set $\bigcup_x G.\texttt{T}_x$.
- po, called *program order*, is a strict partial order on A.
- rf is a set of reads-from edges in $A \times A$.

Note that the program order of an execution may not respect the thread identifiers assigned to its events. This cannot happen in executions that are generated by programs (see Prop. 1 below).

Relating programs and executions. Figure 1 presents the semantics of sequential programs (commands in our language) as a labeled transition system. The system associates (some) *command steps* with labels in Lab. Note that in this stage the values of reads are completely arbitrary. Given the command steps, *program steps* are also labeled transitions, as presented in Fig. 2. These are associated with pairs of the form $\langle l, i\rangle \in \mathsf{Lab} \times \{1, \ldots, N\}$.

To relate programs with executions, we combine the program steps with *execution steps*. The definition of execution steps, given in Fig. 3, employs the following notation:

Notation 1. Given two executions $G_1 = \langle A_1, \mathit{po}_1, \mathit{rf}_1\rangle$ and $G_2 = \langle A_2, \mathit{po}_2, \mathit{rf}_2\rangle$, with $A_1 \cap A_2 = \emptyset$, we denote by $G_1; G_2$ the execution $\langle A_1 \cup A_2, \mathit{po}_1 \cup \mathit{po}_2 \cup \mathit{po}, \mathit{rf}_1 \cup \mathit{rf}_2\rangle$, where $\mathit{po} = \{\langle a_1, a_2\rangle \in A_1 \times A_2 \mid \mathit{tid}(a_1) = \mathit{tid}(a_2)\}$. We identify an event a with the execution $\langle\{a\}, \emptyset, \emptyset\rangle$ when writing expressions like $G; a$.

Thus, each execution step labeled with l, i augments an execution G with an event a, whose label is l and thread identifier is i. Fig. 4 presents the combined semantics, where steps are either a labeled program step combined with an execution step with the same label, or an internal program step that does not affect the execution.

Definition 1. We say that G is an *execution of a program P* if $\langle P, G_\emptyset\rangle \to^* \langle P_{\text{final}}, G\rangle$, where $G_\emptyset$ denotes the empty execution (i.e., $G_\emptyset = \langle\emptyset, \emptyset, \emptyset\rangle$), and P_{final} is the program given by $\lambda i.\ \texttt{skip}$.

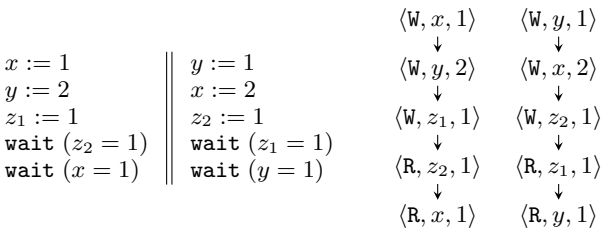

Figure 5. A program together with one of its executions. Arrows denote immediate program order edges.

Figure 5 provides an example of a program (a thread-partitioned version of the (2+2W) example from §1) and one of its executions. Note that if G is an execution of some program P then it does not include any reads-from edges, and it is well-structured with respect to tid. We refer to such executions as *plain*:

Definition 2. An execution $G = \langle A, po, rf\rangle$ is called *plain* if $rf = \emptyset$, $1 \leq tid(a) \leq N$ for every $a \in A$, and $tid(a) = tid(b)$ iff $\langle a, b\rangle \in po \cup po^{-1}$ for every two different events $a, b \in A$.

The last condition guarantees that po consists of a disjoint union of N strict total orders, one for each thread.

Proposition 1. If $\langle P, G_\emptyset\rangle \rightarrow^* \langle P', G\rangle$ then G is plain.

Consistency. Many of the executions associated with a program P are nonsensical as they can, for instance, read values never written in the program. Thus, the model restricts the attention to *consistent* executions. For the purpose of this paper, following [18], we find it technically convenient to define consistency using two properties, that we call *completeness* and *coherence*.

Definition 3. An execution $G = \langle A, po, rf\rangle$ is called *complete* if for every $b \in G.\mathtt{R} \cup G.\mathtt{U}$, we have $\langle a, b\rangle \in rf$ for some $a \in A$.

Completeness of an execution guarantees that every read/update event is justified by a corresponding write/update (recall that, by definition, reads-from edges are only from a write/update event to a read/update event with the same location and value). Obviously, not all reads-from edges are allowed. Roughly speaking, the model has to ensure that overwritten values are not read. RA does this by asserting the existence of a modification order on write accesses to the same location, as defined next.

Definition 4. Given $x \in \mathsf{Loc}$, a relation mo_x is an *x-modification order* in an execution $G = \langle A, po, rf\rangle$ if the following hold:

- mo_x is a strict total order on $G.\mathtt{W}_x \cup G.\mathtt{U}_x$.
- $mo_x; (po \cup rf)^+$ is irreflexive.
- If $\langle a, c\rangle \in rf$ and $\langle b, c\rangle \in (po \cup rf)^+ \cup mo_x$, then $\langle a, b\rangle \notin mo_x$.

Figure 6 illustrates the conditions on mo_x imposed by this definition. First, $(po \cup rf)^+$ (that corresponds to C11's "happens-before" relation) between two write accesses enforces mo_x in the same direction. The second forbidden case corresponds to C11's coherence write-read axiom, that ensures that read/update events cannot read from an overwritten write/update event. The third case is needed to assert that update events read from the mo_x-latest event. Note that program order always appears together with the reads-from relation. This follows the basic principle of release/acquire consistency according to which reads-from edges induce synchronizations between events.

The following alternative more compact definition of an x-modification order is useful in some of the proofs below.

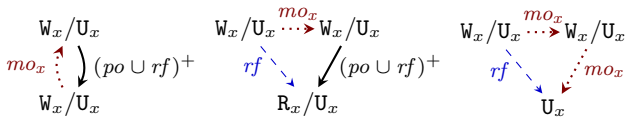

Figure 6. Illustration of forbidden cases according to Def. 4.

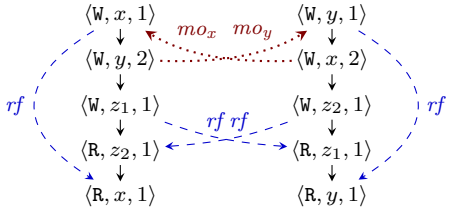

Figure 7. Evidence for RA-consistency of the execution in Fig. 5.

Proposition 2. A relation mo_x is an x-modification order in an execution $G = \langle A, po, rf\rangle$ iff it is a strict total order on $G.\mathtt{W}_x \cup G.\mathtt{U}_x$, and $po \cup rf \cup mo_x \cup fr_x$ is acyclic, where $fr_x = (rf^{-1}; mo_x) \setminus Id_A$.

Next, RA-coherent executions are those that never read "overwritten" values, and RA-consistent executions are defined by combining completeness and RA-coherence.

Definition 5. An execution $G = \langle A, po, rf\rangle$ is called *RA-coherent* if $po \cup rf$ is acyclic (i.e., $(po \cup rf)^+$ is irreflexive) and for every $x \in \mathsf{Loc}$, there exists an x-modification order in G.

Definition 6. A plain execution $G = \langle A, po, \emptyset\rangle$ is called *RA-consistent* if $G' = \langle A, po, rf\rangle$ is complete and RA-coherent for some set $rf \subseteq A \times A$ of reads-from edges.

Figure 7 presents four reads-from edges whose addition to the execution in Fig. 5 results in an RA-coherent execution. In the same figure, we also depict modification orders for each location, witnessing RA-coherence.

Finally, we define the semantics of programs under RA. The idea is to filter out the non-consistent executions among all executions of a given program. If we are left with at least one execution, then the program may terminate under RA. In the formal definition, we also support initialized locations. Thus, we assume some *initial state*, taken to be a function σ from Loc to $\mathsf{Val} \cup \{\bot\}$, where $\sigma(x) = \bot$ means that x is uninitialized. To attach an initialization part to program executions we employ the following notation.

Notation 2. Given an initial state σ, A_σ denotes the set of events $\{a_x \mid \sigma(x) \neq \bot\}$, where each a_x is the event defined by $id(a_x) = tid(a_x) = 0$ and $lab(a_x) = \langle \mathtt{W}, x, \sigma(x)\rangle$. Given an initial state σ and a plain execution $G = \langle A, po, \emptyset\rangle$, $\sigma; G$ denotes the execution $\langle A \cup A_\sigma, po \cup (A_\sigma \times A), \emptyset\rangle$.

Definition 7. A program P *may terminate under RA from an initial state* σ if $\sigma; G$ is RA-consistent for some execution G of P.

In particular, the program in Fig. 5 may terminate under RA from the initial state $\lambda x.\bot$ (no location is initialized), since its execution presented in the figure is RA-consistent.

3. SRA Memory Model

In this section we present the SRA model, a strengthening of RA. The main idea is simple: under RA, no condition relates modification orders of different locations. However, there are cases in which the behavior for each individual location follows the release/acquire paradigm, whereas the combination of the behaviors over all locations should be disallowed. More specifically, executions as the one

in Fig. 7 include a cycle in $po \cup rf \cup \bigcup_x mo_x$. Forbidding such cycles is the only additional condition in SRA.

Definition 8. An execution $G = \langle A, po, rf \rangle$ is called *SRA-coherent* if there exist relations $\{mo_x\}_{x \in \mathsf{Loc}}$, such that:

- For every $x \in \mathsf{Loc}$, mo_x is an x-modification order in G.
- $po \cup rf \cup \bigcup_x mo_x$ is acyclic.

SRA-consistency is defined like RA-consistency (see Def. 6), but refers to SRA-coherence instead of RA-coherence. Similarly, termination under SRA is defined as under RA (see Def. 7), referring to SRA-consistency. (The same applies for the other model definitions that appear in this paper: we define X-coherence, and X-consistency and termination under X are defined as for RA.)

Clearly, we have that RA $\leq$ SRA, i.e., if G is SRA-coherent then it is also RA-coherent. The execution in Fig. 5 is RA-consistent but not SRA-consistent, and hence, SRA is strictly stronger than RA. Next, we show that the two models coincide for executions that do not have any write-write races.

Definition 9. Let $G = \langle A, po, rf \rangle$ be an execution. We say that two events $a, b \in A$ *race in* G if $loc(a) = loc(b)$, $a \neq b$, neither $\langle a, b \rangle \in (po \cup rf)^+$ nor $\langle b, a \rangle \in (po \cup rf)^+$, and either $a \in \mathcal{W} \cup \mathcal{U}$ or $b \in \mathcal{W} \cup \mathcal{U}$. The execution G is called *WW-race free* if no two write/update events race in G.

Proposition 3. WW-race free RA-coherent executions are also SRA-coherent.

Proof. Let $G = \langle A, po, rf \rangle$ be an RA-coherent WW-race free execution, and for every $x \in \mathsf{Loc}$, let mo_x be an x-modification order in G. Since $(po \cup rf)^+$ is total on $G.\mathtt{W}_x \cup G.\mathtt{U}_x$ and $mo_x; (po \cup rf)^+$ is irreflexive, we have that $mo_x \subseteq (po \cup rf)^+$ for every $x \in \mathsf{Loc}$. Hence, $po \cup rf \cup \bigcup_x mo_x$ is acyclic. □

The simplest syntactic way to enforce WW-race freedom in all possible executions of a given program is to disallow commands modifying the same location in different threads. All executions of programs obeying this discipline are WW-race free, and so are their extensions with reads-from edges. Except for the (2+2W) program, the examples above meet this criterion, and hence, for these programs RA and SRA exhibit exactly the same behaviors.

Furthermore, a common approach to ensure WW-race freedom when such a split does not exist is to use *critical sections*. More specifically, given a distinguished lock location l that is accessed by separate `lock()` and `unlock()` pairs of commands (implemented respectively by `when` $(l = 0)$ `do` $l := 1$ and $l := 0$), we refer to commands between consecutive lock and unlock commands as *protected*. Assuming that every command in a program P that modifies a location also modified in another thread, is protected, we have that any complete execution $G' = \langle A, po, rf \rangle$ that extends a plain execution $G = \langle A, po, \emptyset \rangle$ of P is WW-race free. Consequently, SRA and RA coincide for such programs.

3.1 Validity of Local Program Transformations

We now show that SRA does not harm compiler optimizations. We follow the development of Vafeiadis et al. [38], who studied the validity of common source-to-source program transformations under C11, and identified the set of valid reorderings of adjacent commands and eliminations of redundant commands. Here we show that SRA allows the same transformations that were proven to be sound for release/acquire accesses. The basic soundness claim is that a local transformation does not introduce new behaviors. Thus, showing correctness of such transformations amounts to providing a corresponding SRA-consistent execution of the source program for every SRA-consistent execution of the target. First, we consider the elimination of redundant adjacent accesses.

Notation 3. Given a plain execution $G = \langle A, po, \emptyset \rangle$, and an event $a \in A$, we denote by $G \setminus \{a\}$ the plain execution given by $\langle A', po \cap (A' \times A'), \emptyset \rangle$, where $A' = A \setminus \{a\}$.

Proposition 4. Let $\langle a, b \rangle$ be an immediate po-edge of a plain execution $G = \langle A, po, \emptyset \rangle$. Suppose that $loc(a) = loc(b)$, and one of the following holds:

- $a, b \in \mathcal{W}$ and $G \setminus \{a\}$ is SRA-consistent.
- $a, b \in \mathcal{R}$, $val_r(a) = val_r(b)$, and $G \setminus \{a\}$ is SRA-consistent.
- $a \in \mathcal{W}$, $b \in \mathcal{R}$, $val_w(a) = val_r(b)$, and $G \setminus \{b\}$ is SRA-consistent.

Then, G is SRA-coherent.

On the program level, Prop. 4 ensures the soundness of simplifications like:

$$\begin{array}{ccc} x := 1; x := 2 & \rightsquigarrow & x := 2 \\ \texttt{if } x = 1 \texttt{ then } y := x \texttt{ else } c & \rightsquigarrow & \texttt{if } x = 1 \texttt{ then } y := 1 \texttt{ else } c \\ x := 1; y := x & \rightsquigarrow & x := 1; y := 1 \end{array}$$

The second kind of sound transformations under RA are reordering of adjacent accesses. First, reordering of two accesses of different locations, such that at least one of them is local, is safe. Second, a write access and a consecutive read access to a different location can be safely reordered. The next proposition ensures that these transformations are sound also for SRA.

Proposition 5. Let $\langle a, b \rangle$ be an immediate po-edge of a plain execution $G = \langle A, po, \emptyset \rangle$. Suppose that $loc(a) \neq loc(b)$, the execution $\langle A, (po \setminus \{\langle a, b \rangle\}) \cup \{\langle b, a \rangle\}, \emptyset \rangle$ is SRA-consistent, and one of the following holds:

- Either $loc(a)$ or $loc(b)$ is local in G (a location x is *local* in G if po is total on $\{a \in A \mid loc(a) = x\}$).
- $a \in \mathcal{W}$ and $b \in \mathcal{R}$.

Then, G is SRA-consistent.

This allows basic reorderings, and they can be combined with the previous eliminations. For example, $x := 1; y := 1; z := x$ can be simplified to $x := 1; y := 1; z := 1$ under any context. Indeed, the resulting program executions will have three consecutive events a, b, c with labels $\langle \mathtt{W}, x, 1 \rangle, \langle \mathtt{W}, y, 1 \rangle, \langle \mathtt{W}, z, 1 \rangle$ (respectively). By the third part of Prop. 4, its consistency implies the consistency of the same execution with a new node a' between a and b whose label is $\langle \mathtt{R}, x, 1 \rangle$. Then, by the second part of Prop. 5, we have that the same execution with a reversed immediate po-edge between a' and b is also consistent. The last execution is an execution of the original program, and hence the transformation is sound.

Finally, note that from the definition of SRA (similarly to RA) it is clear that removing po-edges preserves SRA-consistency. Therefore, the basic sequentialization transformation "$c_1 \parallel c_2 \rightsquigarrow c_1; c_2$" is sound. Surprisingly, this transformation is unsound under the TSO model (see §6). Indeed, the (IRIW) program does not terminate under TSO, while its transformation that puts the two writes before the corresponding reads may terminate.

3.2 An Alternative Formulation

An equivalent definition of SRA can be obtained by requiring that there is a single modification order over *all* locations satisfying similar conditions to those of an x-modification order. In particular, this definition will be used to relate SRA and TSO in §6.

Definition 10. A relation mo is called a *modification order* in an execution $G = \langle A, po, rf \rangle$ if the following hold:

- mo is a strict total order on $G.\mathtt{W} \cup G.\mathtt{U}$.
- $mo; (po \cup rf)^+$ is irreflexive.
- If $\langle a, c \rangle \in rf$, $\langle b, c \rangle \in (po \cup rf)^+ \cup mo$, and $loc(a) = loc(b)$, then $\langle a, b \rangle \notin mo$.

Proposition 6. If mo is a modification order in G, then for every $x \in \mathsf{Loc}$, $mo_x = mo \cap ((G.\mathtt{W}_x \cup G.\mathtt{U}_x) \times (G.\mathtt{W}_x \cup G.\mathtt{U}_x))$ is an x-modification order.

Proposition 7. An execution $G = \langle A, po, rf \rangle$ is SRA-coherent iff $po \cup rf$ is acyclic and there exists a modification order in G.

4. An Equivalent Operational Model

In this section we develop a simple and intuitive operational semantics for our programming language that precisely corresponds to the SRA model. In other words, we propose a particular machine, define how it executes programs, and show that a program P may terminate under SRA iff it can be fully executed in the machine. The machine consists of N processors, each of which is running a particular program thread. Unlike program executions, machine executions are ordinary sequentially consistent executions, where steps are taken non-deterministically by different processors. In addition, the machine transitions do not refer at all to formal consistency requirements on graphs. We believe that these properties make this semantics easy to grasp by practitioners, as well as a solid foundation for adapting existing formal verification and testing methods for the SRA weak memory model.

There are two main ideas in the proposed machine structure. First, it is based on point-to-point communication. Each processor has a *local memory* and an outgoing *message buffer*, that consists of write instructions performed or observed by that processor. The processors non-deterministically choose between performing their own instructions or processing messages from other processors' buffers. When performing a read, while following some program instruction, the processor obtains the read value by inspecting its local memory. When performing a write, either when following a program instruction or when processing a message, the processor writes to its local memory and reports this write to the other processors by putting a corresponding message in its buffer. Every message can be processed at most once by every other processor, and messages should be processed in the same order in which they were issued. Thus, message buffers are taken to be lists, and each processor maintains a set of indices storing its current location in every other buffer, and constantly progressing while processing messages. Naturally, a message can be removed from a buffer when all processors have processed it.

This already gives intuitive account for the possible non-SC behavior in the store buffering example, that cannot occur in the message passing example (see §1). In a terminating run of the (SB) program, both processors follow their own instructions, ignoring the other message buffer. On the other hand, in every run of the (MP) program, for the first wait command to terminate, the second processor must process the message from the first processor that sets y to 1. Since outgoing messages are ordered, it will first process the message that sets x to 1, assigning 1 to x in its local memory. Thus, the second wait will never terminate.

Nevertheless, so far, the proposed machine fails to explain the following example (litmus test CoRR2 in [23]):

$$x := 1 \;\Big\|\; x := 2 \;\Big\|\; \begin{array}{l}\texttt{wait}\ (x = 1)\\ \texttt{wait}\ (x = 2)\end{array} \;\Big\|\; \begin{array}{l}\texttt{wait}\ (x = 2)\\ \texttt{wait}\ (x = 1)\end{array} \qquad \text{(CoRR2)}$$

Assuming that initially all variables are 0, under SRA (or, similarly, under RA or SC, as they coincide on programs with a single variable) this program cannot terminate. Indeed, in its executions, the x-modification order must order the two writes, forcing one particular order in which the two values can be observed. However, according to the description above, nothing forbids the two readers to process the messages from the two buffers in opposite orders.

To address this mismatch, the second main idea in the machine is the use of global timestamps counting the number of writes to every location. Whenever some local write is performed to a location x, the global timestamp for x is increased, and attached both to the local stored value and to the message reporting this write. When processing a write message to location x from another buffer, the processor compares the timestamp stored in its local memory for x with the timestamp of the message, and performs the write only if the message has a greater timestamp. Otherwise, the processor skips the message. For the example above, assuming that the first reader terminated, it must be the case that the $x := 2$ was performed after the $x := 1$ write (otherwise, this reader would skip the message for $x := 2$). Hence, the second reader can either process the $x := 1$ message first, and then the $x := 2$ one (exactly as the first reader), or process first the $x := 2$ message, but then it is forced to skip the older message, and cannot observe the value 1.

As shown below, a machine based on these two simple ideas provides a precise account for SRA. Note that this operational model is too strong for RA. Indeed, getting back to the (2+2W) example (see Fig. 5), assuming that the first processor terminated and the second arrived just before the last wait, note that before processing the $z_2 := 1$ message, the first processor must have processed the $x := 2$ message before its own $x := 1$ instruction, or skipped the $x := 2$ message. Thus, $x := 2$ had a smaller timestamp than $x := 1$, and so, $y := 1$ had a smaller timestamp than $y := 2$. This implies, however, that between performing the local $y := 1$ write instruction and processing the $z_1 := 1$ message, the second processor must have processed the $y := 2$ message, and replace the value locally stored for y from 1 to 2. Thus, the last wait of the second processor cannot terminate.

Next, we turn to the formal development of the operational semantics and its soundness and completeness proof. We define the machine as a labeled transition system: a set of states and a labeled transition relation. A *machine state* takes the form of a pair $[S, T]$, where S assigns a *processor state* to every processor, and T is the global *timestamps* table. In turn, a processor state consists of a local memory M, an outgoing message buffer L, and a function I that records the current location in every other message buffer. For every $1 \leq i \leq N$, $I(i)$ is the index of the first message in the buffer of processor i which was not yet observed by the current processor.

Definition 11. A *processor state* is a tuple $\langle M, L, I \rangle$, where:

- $M : \mathsf{Loc} \to ((\mathsf{Val} \times \mathbb{N}) \cup \{\langle \bot, 0 \rangle\})$ is the *local memory*. We write $M(x) = v@t$ for $M(x) = \langle v, t \rangle$, and $M(x) = \bot@0$ means that x is uninitialized.
- L is an *outgoing message buffer* that takes the form of a list of messages, where a *message* m has the form $\lfloor x := v@t \rfloor$, where $x \in \mathsf{Loc}$, $v \in \mathsf{Val}$, and $t \in \mathbb{N}$. We use ";" for concatenation of such lists (and write, e.g., $m; L$ or $L; m$), and often refer to L as an array. $L[k]$ denotes the kth message in L (starting from 1), and $|L|$ is the number of messages in L.
- $I : \{1, \ldots, N\} \to \mathbb{N}^+$ is a function assigning an index in every other message list.

Definition 12. A *machine state* is a pair of the form $[S, T]$, where S is a function assigning a processor state to every processor $1 \leq i \leq N$, and $T : \mathsf{Loc} \to \mathbb{N}$ assigns a timestamp to every location. Given an initial state σ, the *initial machine state* $[S_\sigma, T_\emptyset]$ is given by $S_\sigma = \lambda i.\ \langle M_\sigma, L_\emptyset, I_\emptyset \rangle$ and $T_\emptyset = \lambda x.\ 0$, where $M_\sigma = \lambda x.\ \sigma(x)@0$, $L_\emptyset = \epsilon$, and $I_\emptyset = \lambda i.\ 1$.

The machine semantics is given in Fig. 8. Its steps follow the informal description above: (READ) steps read from the local memory; (WRITE) steps write to the local memory, add a corresponding message, and increment the global stored timestamp for the specific location; (PROCESS) steps pull a message from some buffer with a timestamp that is greater than the local stored one, perform the local write, and increment the current location in the buffer; (SKIP) steps

$$\text{(READ)}\ \frac{S(i) = \langle M, -, - \rangle \quad M(x) = v@-}{[S,T] \xrightarrow{\langle \mathtt{R},x,v\rangle,i} [S,T]}$$

$$\text{(WRITE)}\ \frac{S(i) = \langle M, L, I\rangle \quad T(x) = t \quad M' = M[x \mapsto v@t+1] \quad L' = L; \lfloor x{:=}v@t+1\rfloor \quad T' = T[x \mapsto t+1]}{[S,T] \xrightarrow{\langle \mathtt{W},x,v\rangle,i} [S[i \mapsto \langle M', L', I\rangle], T']}$$

$$\text{(UPDATE)}\ \frac{S(i) = \langle M, L, I\rangle \quad T(x) = t \quad M(x) = v_r@t \quad M' = M[x \mapsto v_w@t+1] \quad L' = L; \lfloor x{:=}v_w@t+1\rfloor \quad T' = T[x \mapsto t+1]}{[S,T] \xrightarrow{\langle \mathtt{U},x,v_r,v_w\rangle,i} [S[i \mapsto \langle M', L', I\rangle], T']}$$

$$\text{(PROCESS)}\ \frac{i \neq j \quad S(i) = \langle M, L, I\rangle \quad S(j) = \langle -, L_j, -\rangle \quad I(j) = k \quad k \leq |L_j| \quad L_j[k] = \lfloor x{:=}v@t_j\rfloor \quad M(x) = -@t_i \quad t_i < t_j \quad M' = M[x \mapsto v@t_j] \quad L' = L; \lfloor x{:=}v@t_j\rfloor \quad I' = I[j \mapsto k+1]}{[S,T] \to [S[i \mapsto \langle M', L', I'\rangle], T]}$$

$$\text{(SKIP)}\ \frac{i \neq j \quad S(i) = \langle M, L, I\rangle \quad S(j) = \langle -, L_j, -\rangle \quad I(j) = k \quad k \leq |L_j| \quad L_j[k] = \lfloor x{:=}-@t_j\rfloor \quad M(x) = -@t_i \quad t_j \leq t_i \quad I' = I[j \mapsto k+1]}{[S,T] \to [S[i \mapsto \langle M, L, I'\rangle], T]}$$

$$\text{(CLEAN)}\ \frac{S(i) = \langle M, m; L, I\rangle \quad \forall j \neq i.\ S(j) = \langle M_j, L_j, I_j\rangle \quad \forall j \neq i.\ I_j(i) > 1 \quad \forall j \neq i.\ S'(j) = \langle M_j, L_j, I_j[i \mapsto I_j(i)-1]\rangle \quad S'(i) = \langle M, L, I\rangle}{[S,T] \to [S',T]}$$

Figure 8. Machine steps

$$\frac{P \xrightarrow{l,i} P' \quad [S,T] \xrightarrow{l,i} [S',T']}{\langle P, [S,T]\rangle \to \langle P', [S',T']\rangle}$$

$$\frac{P \to P'}{\langle P, [S,T]\rangle \to \langle P', [S,T]\rangle} \qquad \frac{[S,T] \to [S',T']}{\langle P, [S,T]\rangle \to \langle P, [S',T']\rangle}$$

Figure 9. Program and machine combined steps

pull a message from some buffer with a timestamp that is less than or equal to the local stored one, and skips the message by just incrementing the current location in the buffer; and (CLEAN) steps remove the first message in a buffer, provided that all processors have already processed or skipped it. Finally, (UPDATE) steps (which were not explained above) naturally combine reads and writes. In addition, for performing (UPDATE) the processor should have the most recent value of the updated location x. Thus, it verifies that the timestamp t stored in its memory for x ($M(x) = v_r@t$) is equal to the global timestamp of x ($T(x) = t$). Otherwise, the processor cannot continue and is forced to pull messages from other processors. This corresponds to the fact that SRA update events (exactly as RA ones, and unlike plain read events) should read from their immediate predecessor in the relevant modification order.

The combined machine and program semantics is given in Fig. 9. Note that it is defined exactly as the combined execution and program semantics (see Fig. 4), using machine steps instead of execution steps, and including internal machine steps that do not affect the program ((PROCESS), (SKIP), and (CLEAN) steps).

Theorem 1 (Soundness and Completeness)**.** A program P may terminate under SRA from an initial state σ iff $\langle P, [S_\sigma, T_\emptyset]\rangle \to^* \langle P_{\text{final}}, [S,T]\rangle$ for some machine state $[S,T]$.

A straightforward application of this theorem is to provide alternative accounts for the soundness of the source-to-source transformations mentioned in §3.1. Using the operational model, proving validity of such transformations becomes a more ordinary task: one should show that the behavior of the machine on the resulting program is also possible when running the original one. For example, we give an informal argument for the soundness of the transformation that eliminates the first assignment command in a pair of the form $x := v; x := v'$. To simulate an execution of the resulting program, the machine running the original program can perform the two writes in two consecutive steps when the resulting program performs the second write, and pulls the two corresponding messages in consecutive steps whenever the machine running the resulting program pulls the $x := v'$ message.

Note that two possible variants of the machine are equivalent to our definition. First, if we omit the (CLEAN) rule, then the local memory is uniquely defined by the message buffer, and instead of inspecting the value (and timestamp) of a location x in the memory, each processor can check for the last message modifying x that appears in its buffer. The second alternative is to have one message queue between every (ordered) pair of processors. In this case, there is no need for pointers in message lists. Instead, every write message is enqueued to all outgoing queues, and the processors asynchronously dequeue messages from their incoming queues.

Soundness and completeness proof. As the main tool in the proof of Thm. 1, we introduce the notion of a *history*, that intuitively correspond to a log of a machine execution. Formally, a history is a totally ordered set of *actions* – expressions of the form $a@i$ where a is an event and i is a thread identifier, such that $a \in \mathcal{W} \cup \mathcal{U}$ whenever $tid(a) \neq i$. An action $a@i$ is standing for "processor i observes event a". The event a can either be an event originated by thread i ($tid(a) = i$), or a write/update event of another thread (as reads are not reported in message buffers). Obviously, not every order of actions is possible. Next, we define *legal* histories.

Definition 13. A history $H = \langle B, \leq\rangle$ induces the following functions:

1. $H.before : \mathcal{R} \cup \mathcal{U} \to \mathcal{P}(\mathcal{W} \cup \mathcal{U})$ is given by $H.before(a) = \{b \in \mathcal{W} \cup \mathcal{U} \mid b@tid(a) < a@tid(a), loc(b) = loc(a)\}$.
2. $H.last : \mathcal{R} \cup \mathcal{U} \rightharpoonup \mathcal{W} \cup \mathcal{U}$ is a partial function assigning a write/update event b to every read/update event a such that $b@tid(b) = \max_{\leq}\{c@i \in B \mid c \in H.before(a), i = tid(c)\}$. If $H.before(a)$ is empty, then $H.last(a)$ is undefined.

Definition 14. A history $H = \langle B, \leq\rangle$ is called *legal* if the following hold:

1. $val_r(a) = val_w(H.last(a))$ for every $a \in dom(H.last)$.
2. For every $a \in \mathcal{W} \cup \mathcal{U}$ and $b@i \in B$, if $a@tid(b) < b@tid(b)$, then $a@i \in B$ and $a@i < b@i$.
3. For every $a \in \mathcal{W} \cup \mathcal{U}$ and $b \in \mathcal{U}$, if $loc(a) = loc(b)$ and $a@tid(a) < b@tid(b)$, then $b \in dom(H.last)$ and $a@tid(a) \leq H.last(b)@tid(H.last(b))$.
4. $a@tid(a) \leq a@i$ for every $a@i \in B$.

The first condition ensures that the performed reads/updates obtain the values of the latest write/update (to the relevant location) that was observed by the current thread. The second condition

corresponds to the fact that message buffers are ordered—before thread i observes an event b originated by thread j, it must observe every event a that was observed by j before it originated b. The third requirement guarantees that updates are performed only while having the latest value. The last condition ensures that events are observed by other threads only after their origination. Next, we relate histories to a given plain execution and initial state.

Definition 15. Let $G = \langle A, po, \emptyset \rangle$ be a plain execution, and σ be an initial state. A history $H = \langle B, \leq \rangle$ is called:

- G-*suitable* if the following hold:
  1. For every $a@i \in B$, we have $a \in A$.
  2. For every $a \in A$, we have $a@tid(a) \in B$.
  3. For every $\langle a, b \rangle \in po$, we have $a@tid(a) < b@tid(b)$.
- σ-*suitable* if $val_r(a) = \sigma(loc(a))$ whenever $a@tid(a) \in B$ and $a \in (\mathcal{R} \cup \mathcal{U}) \setminus dom(H.last)$.
- $\langle G, \sigma \rangle$-*legal* if it is legal, G-suitable and σ-suitable.

The next key theorem is the formal correspondence between SRA-consistency and legal histories.

Theorem 2. Let G be a plain execution, and σ be an initial state. Then, $\sigma; G$ is SRA-consistent iff there exists a $\langle G, \sigma \rangle$-legal history.

Proof sketch. We consider here the case that $\sigma = \lambda x.\ \bot$. Assuming $G = \langle A, po, \emptyset \rangle$ is SRA-consistent, choose rf such that $G' = \langle A, po, rf \rangle$ is a complete and SRA-coherent execution, and let mo be a modification order in G' (see Prop. 7). For every $1 \leq i \leq N$, let $A_i = \{a \in A \mid tid(a) = i\}$, $A'_i = A_i \cup G.\mathtt{W} \cup G.\mathtt{U}$, and $\prec_i = ((hb \cap (A'_i \times A'_i)) \cup ((A_i \times (A'_i \setminus A_i)) \setminus hb^{-1}))^+$, where $hb = (po \cup rf)^+$. In addition, let $B = \{a@i \mid 1 \leq i \leq N, a \in A'_i\}$, $B' = \{a@i \in B \mid tid(a) = i, a \in \mathcal{W} \cup \mathcal{U}\}$, and define the following relations:

1. $T_1 = \{\langle a@i, b@j \rangle \in B \times B \mid i = j, a \prec_i b\}$.
2. $T_2 = \{\langle a@i, b@j \rangle \in B' \times (B \setminus B') \mid a = b\}$.
3. $T_3 = \{\langle a@i, b@j \rangle \in B' \times B' \mid \langle a, b \rangle \in mo\}$.

We show that $T_1 \cup T_2 \cup T_3$ is acyclic, take $\leq$ to be a total order on B extending $(T_1 \cup T_2 \cup T_3)^+$, and prove that H is $\langle G, \sigma \rangle$-legal.

For the converse, given a $\langle G, \sigma \rangle$-legal history $H = \langle B, \leq \rangle$, we show that $G' = \langle A, po, H.rf \rangle$ is a complete execution, where $H.rf = \{\langle a, b \rangle \in \mathcal{A} \times \mathcal{A} \mid H.last(b) = a\}$. To prove that it is also SRA-coherent, we choose $mo = \{\langle a, b \rangle \in (G.\mathtt{W} \cup G.\mathtt{U}) \times (G.\mathtt{W} \cup G.\mathtt{U}) \mid a@tid(a) < b@tid(b)\}$, and use Prop. 7. □

Roughly speaking, using Theorem 2, the soundness proof proceeds by showing that a log of a terminating execution of the machine from an initial state $[S_\sigma, T_\emptyset]$ of a program P forms a $\langle G, \sigma \rangle$-legal history, where G is an execution of P. The completeness proof makes use of the other direction of Theorem 2, and shows that histories are "executable", i.e., every $\langle G, \sigma \rangle$-legal history, where G is an execution of a program P, can be followed by the machine executing P starting from $[S_\sigma, T_\emptyset]$.

5. Fence Commands

In this section, we extend our programming language with a memory fence command, denoted by `fence()`, and show that these commands can be used to enforce sequential consistency under RA (or SRA). Our semantics for fence commands is as if they were atomic assignments (RMWs) to a particular otherwise unused location. In other words, `fence()` is simply syntactic sugar for the atomic assignment $\langle f := 0 \rangle$, where f is a distinguished location that is not mentioned in any non-fence command. As a result, `fence()` instructions induce *fence events*, that is events with labels $\langle \mathtt{U}, f, 0, 0 \rangle$. We further require that any initial state σ has $\sigma(f) = 0$. Accordingly, executions of programs (together with the initialization part) are all well-formed, as defined next:

Definition 16. An execution $G = \langle A, po, rf \rangle$ is *well-formed* if the set $A_f = \{a \in A \mid loc(a) = f\}$ consists only of fence events, except for one event a with $lab(a) = \langle \mathtt{W}, f, 0 \rangle$ that initializes f (i.e., $\langle a, b \rangle \in po$ for every $b \in A_f \setminus \{a\}$).

The fundamental property, that allows the reduction to SC, is that fence events are totally ordered by rf^+ as shown by the following proposition.

Proposition 8. Given a well-formed complete RA-coherent execution $G = \langle A, po, rf \rangle$, the relation rf^+ is total on the set of fence events in G.

Based on this property, we show that our fences are sufficient for restoring SC. More specifically, we call an event G-*racy* if it races with some other event in an execution G (see Def. 9), and show that having a fence event between every two $(po \cup rf)^+$-related racy events in a well-formed complete RA-coherent execution means that the execution is also SC-coherent. Following Shasha and Snir [31], we employ the following definition of SC-coherence:

Definition 17. An execution $G = \langle A, po, rf \rangle$ is called *SC-coherent* if there exist relations $\{mo_x\}_{x \in \mathsf{Loc}}$, such that:

- For every $x \in \mathsf{Loc}$, mo_x is an x-modification order in G.
- The relation $po \cup rf \cup \bigcup_x mo_x \cup \bigcup_x fr_x$ is acyclic, where $fr_x = (rf^{-1}; mo_x) \setminus Id_A$.

Theorem 3. Let $G = \langle A, po, rf \rangle$ be a well-formed complete RA-coherent execution. Suppose that for every two G-racy events a, b, if $loc(a) \neq loc(b)$, and $\langle a, b \rangle \in po \cup (po; (po \cup rf)^*; po)$, then $\langle a, c \rangle, \langle c, b \rangle \in (po \cup rf)^+$ for some fence event c. Then, G is SC-coherent.

A simple corollary of this theorem is a per-execution data race freedom (DRF) property:

Corollary 1. Well-formed complete RA-coherent executions that have no racy events are also SC-coherent.

The simplest way to enforce executions that satisfy the condition of Thm. 3 in a given program is for each thread to include a fence command between every two accesses of different shared variables. In general, all these fences are necessary: the SB program shows fences may be needed between writes and subsequent reads, IRIW shows that fences are needed between two reads, and the following two programs show that fences are required between two writes, and between a read and a subsequent write. Assuming all variables are initialized to 0, these programs (without the crossed out fences) may terminate under RA (or SRA) but not under SC.

$$\begin{array}{c|c|c} x := 1 & y := 2 & \mathtt{wait}\ (y = 1) \\ \cancel{\mathtt{fence()}} & \mathtt{fence()} & \mathtt{fence()} \\ y := 1 & \mathtt{wait}\ (x = 0) & \mathtt{wait}\ (y = 2) \end{array}$$

$$x := 1 \left\| \begin{array}{c|c|c} \mathtt{wait}\ (x = 1) & y := 2 & \mathtt{wait}\ (y = 1) \\ \cancel{\mathtt{fence()}} & \mathtt{fence()} & \mathtt{fence()} \\ y := 1 & \mathtt{wait}\ (x = 0) & \mathtt{wait}\ (y = 2) \end{array} \right.$$

It is worth contrasting Thm. 3 with the corresponding one for TSO. Under TSO, fences between every shared variable write and every subsequent shared variable read suffice to restore SC [24]. For a particular common class of programs, however, we can get a reduction to SC that requires fewer fences similar to those required for TSO. This is based on the following theorem:

Theorem 4. Let $G = \langle A, po, rf \rangle$ be a well-formed complete RA-coherent execution. Assume that G is WW-race free and there exists a set $B \subseteq A$ of *protected events* such that the following hold:

1. $(po \cup rf)^+$ is total on B.
2. If a races with b in G, then either $a \in B$ or $b \in B$.
3. For every G-racy write/update event $a \in B$ and G-racy read event $b \in B$, if $loc(a) \neq loc(b)$ and $\langle a, b\rangle \in (po \cup rf)^+$, then $\langle a, c\rangle, \langle c, b\rangle \in (po \cup rf)^+$ for some fence event c.
4. For every G-racy write/update event $a \notin B$ and G-racy read event $b \notin B$, if $loc(a) \neq loc(b)$ and $\langle a, b\rangle \in (po \cup rf)^+$, then $\langle a, c\rangle, \langle c, b\rangle \in (po \cup rf)^+$ for some fence *or* protected event c.

Then, G is SC-coherent.

Theorem 4 can be applied in various cases, e.g., two-threaded programs (by taking B to consist of the events of one of the threads), and concurrent data structures with a single writer and multiple readers (by taking B to consist of the writer events).

A direct use of Thm. 4 is for programs that include critical sections (see the discussion after Prop. 3 in §3). First, we lift the definitions of racy events to the program level by over-approximating the set of racy events: Say that two commands *race on* x if they both mention x, appear on different threads, at least one of them modifies x, and at least one of them is unprotected (appears outside a critical section). In addition, we say that a command is an *x-racy read command* if it reads x and races on x with some other command; and similarly, a command is an *x-racy write command* if it modifies x and races on x with some other command. Then, obeying the following conditions suffice to guarantee SC behavior:

(i) Every command modifying a location also modified in another thread, is protected.

(ii) No two unprotected commands race (on some location).

(iii) There exist fence commands:

- Outside critical sections, between every x-racy write command and subsequent y-racy read command for $x \neq y$.
- Inside critical sections, between every x-racy write command and subsequent y-racy read command for $x \neq y$.
- Either after the last racy write command or before the first racy read command inside all critical sections.

Indeed, suppose that a program P obeying this discipline may terminate under RA. Let $G' = \langle A, po, rf\rangle$ be an RA-coherent complete execution that extends a plain execution $G = \langle A, po, \emptyset\rangle$ of P. Condition (i) ensures that G' is WW-race free. We take the set B of protected events to be the set of all events induced by the commands in the critical sections. Then, $(po \cup rf)^+$ is total on B. Further, we can assume that all `lock()` commands immediately succeed and induce a single update event in G' (otherwise, remove the read events that correspond to failed attempts to acquire the lock, and G' remains an RA-coherent complete execution that extends an execution of P). Condition (ii) ensures no races between two events outside B. Additionally, conditions (ii)-(iii) guarantee that conditions 3 and 4 of Thm. 4 are met. The theorem implies that G' is SC-coherent, and hence P may terminate under SC.

5.1 Verifying a Read-Copy-Update Implementation

We demonstrate the application of Thm. 4 to a practical weak memory algorithm, the user-mode read-copy-update (RCU) implementation of Desnoyers et al. [13].

RCU is a mechanism, deployed heavily in the Linux kernel, that allows a single writer to manipulate a data structure, such as a linked list or a binary search tree, while multiple readers are concurrently accessing it. To ensure that a single writer updates the data structure at any given time, a global lock is used to protect the writes. To ensure correctness of the concurrent readers, the writer instead of directly modifying a piece of the structure, first copies that piece, modifies the copy, and then makes the new copy accessible and the old one inaccessible. To deallocate the inaccessible piece, it invokes the `synchronize_rcu()` procedure, which waits for all the readers to stop accessing the old copy.

```
   rcu_quiescent_state():
L1:  ~~fence();~~     // fence removed
L2:  rc[get_my_tid()] := gc;
L3:  fence();
   rcu_thread_offline():
L4:  ~~fence();~~     // fence removed
L5:  rc[get_my_tid()] := 0;
L6:  fence();     // fence inserted
   rcu_thread_online():
L7:  rc[get_my_tid()] := gc;
L8:  fence();
   synchronize_rcu():
L10:  local was_online := (rc[get_my_tid()] ≠ 0);
L11:  ~~fence();~~             // fence removed
L12:  if was_online then rc[get_my_tid()] := 0;
L13:  lock();
L14:  gc := gc + 1;
L15:  ~~barrier();~~ fence();  // barrier replaced by fence
L16:  for i := 1 to N do
L17:    wait (rc[i] ∈ {0,gc});
L18:  unlock();
L19:  if was_online then rc[get_my_tid()] := gc;
L20:  fence();
```

Figure 10. A user-mode read-copy-update implementation based on [13], where all variable accesses are release/acquire accesses. The fences are shown as in the original program: `fence()` is a memory fence, while `barrier()` is a compiler fence. In our variant, we replace the compiler fence with a (stronger) memory fence, insert a fence on line 6, and remove the unnecessary fences.

The readers, on the other hand, do not acquire any locks. They only need to periodically call `rcu_quiescent_state()` when they are not accessing the data structure and not keep any internal pointers to the data structure across calls to `rcu_quiescent_state()`. Alternatively, the readers may also call `rcu_thread_offline()` when they have stopped accessing the data structure and call `rcu_thread_online()` before accessing it again. The second alternative is slightly more expensive but avoids the need of having to call `rcu_quiescent_state()` every once in a while.

Figure 10 shows the implementations of the four aforementioned synchronization methods. The threads synchronize via two variables: the global counter, `gc`, and an array of read counters, `rc`, with one entry for each thread, and use a global lock to serialize accesses to the `synchronize_rcu` method.

RCU programs, based on the implementation in Fig. 10, follow the syntactic discipline that allows us to apply Thm. 4.

(i) The only commands that modify a location also modified in another thread, are the assignment to `gc` on line 14, and the writer's updates of the RCU-protected data structure, which are all protected by the lock.

(ii) Restricting attention to unprotected commands (lines 2, 5, 7, 10, 12, 19), the only locations that are modified are the `rc[i]`'s and `was_online`, but each of them is accessed only by one particular thread. The RCU-protected data structure itself is modified only by protected commands.

(iii) Outside the critical sections, there is a fence immediately after every racy write command (namely, the assignments to `rc[get_my_tid()]` on lines 2,5,7,19).

(iv) Within the critical section of `synchronize_rcu()` (lines 14-17), there is a fence immediately after every assignment

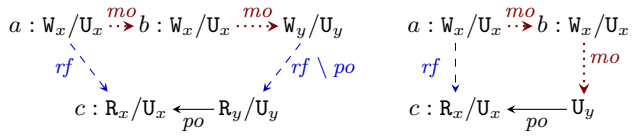

Figure 11. Illustration of the additional forbidden cases under TSO according to Thm. 5.

(namely, the assignment to gc on line 14), and before the first racy read (line 17).

(v) The critical section modifying the RCU-protected data structure has no racy reads.

Therefore, by Thm. 4, RCU programs have the same behaviors under SC as under RA (and SRA).

6. Relation to TSO

In this section we study the relationship between SRA and TSO, the *total store ordering* memory model provided by the x86 and SPARC architectures. TSO is known to be stronger than RA. We show that it is also (strictly) stronger than SRA. This entails that the ordinary compilation of C11 release/acquire accesses to TSO actually also guarantees SRA consistency.

To relate SRA and TSO, we use the declarative model of TSO of Owens et al. [25] that was used to prove correctness of the compilation of RA to x86-TSO.

Definition 18. A relation tso is called a *total store order* for an execution $G = \langle A, po, rf\rangle$ if it satisfies the following:

- tso is a strict partial order on A, that is total on $G.\mathtt{W} \cup G.\mathtt{U}$.
- $po \subseteq tso \cup (G.\mathtt{W} \times G.\mathtt{R})$ and $rf \subseteq tso \cup po$.
- If $\langle a, c\rangle \in rf$, $\langle b, c\rangle \in po \cup tso$, $b \in \mathcal{W} \cup \mathcal{U}$, and $loc(a) = loc(b)$, then $\langle a, b\rangle \notin tso$.

G is called *TSO-coherent* if there exists a total store order for it.

Note that unlike modification orders, the total store order relates also read events. Next, we provide an equivalent characterization that is based on a modification order and has a similar nature to the formulation of SRA given in Prop. 7 (see also Fig. 11).

Theorem 5. An execution $G = \langle A, po, rf\rangle$ is TSO-coherent iff $po \cup rf$ is acyclic, and there exists a modification order mo in G (see Def. 10) that satisfies the following additional condition:

- If $\langle a, c\rangle \in rf$, $\langle b, c\rangle \in ((mo; (rf \setminus po)) \cup (mo \cap (A \times \mathcal{U})))$; po, and $loc(a) = loc(b)$, then $\langle a, b\rangle \notin mo$.

Proof sketch. For one direction, we take $mo = tso \cap ((\mathcal{W} \cup \mathcal{U}) \times (\mathcal{W} \cup \mathcal{U}))$, and show that mo satisfies the required conditions. For the converse, we show that $(mo \cup (po \setminus (\mathcal{W} \times A)) \cup (rf \setminus po))^+$ is a total store order for G. □

Intuitively speaking, the modification order of Thm. 5 is the order in which the writes propagate to the main memory of the TSO-machine. When a processor of a TSO-machine reads from a location x, it obtains the value of the last write instruction to x that appears in its local store buffer, and if no such instruction exists, it obtains the value of the write to x that was the *last* to propagate to the memory. The condition on the pair $\langle b, c\rangle$ in the theorem ensures that the reads-from edge $\langle a, c\rangle$ corresponds to reading from the main memory, rather than from the local buffer. Note that taking just $\langle b, c\rangle \in ((mo; rf) \cup mo); po$ instead of $\langle b, c\rangle \in ((mo; (rf \setminus po)) \cup (mo \cap (A \times \mathcal{U})))$; po in the condition given in Thm. 5 results in another formulation of SC.

Using this theorem and Prop. 7, the relation to SRA is an immediate corollary: We have that SRA $\leq$ TSO, i.e., if G is TSO-coherent then it is also SRA-coherent. Note that SRA is strictly weaker than TSO (even for WW-race free executions): the IRIW program (presented in §1) may terminate under SRA but not under TSO. In addition, as the following example shows, two threads suffice for observing the difference between SRA and TSO.

$$\begin{array}{l} x := 1 \\ \langle f_1 := 1\rangle \\ \mathtt{wait}\ (y = 0) \end{array} \Bigg\| \begin{array}{l} y := 1 \\ \langle f_2 := 1\rangle \\ \mathtt{wait}\ (x = 0) \end{array} \qquad \text{(SBU)}$$

This program may terminate under SRA but not under TSO (where initially all variables are 0). However, if we restrict our attention to executions without update events, then SRA and TSO coincide for two threaded executions. To see this, it suffices to note that $mo; (rf \setminus po) \subseteq (po \cup rf)^+$ for such executions.

An important consequence of the relation to SRA is that the same compilation scheme for RA (that maps writes, reads, and updates to plain TSO write, read, and RMW instructions [6]) guarantees SRA. Therefore, SRA has no additional implementation cost over RA on TSO-machines. Additionally, note that our fence instructions (updates to an unused location, see §5) can be compiled directly to an `mfence` instruction on x86-TSO. The correctness is obvious: Compiling the atomic assignment $\langle f := 0\rangle$ to a suitable x86-TSO RMW instruction (such as `lock xchg`) is sound. But then, according to the operational x86-TSO model, if we ignore the value of location f, the effect of an `mfence` and a RMW on f is identical: they wait for the store buffer to be flushed.

Next, we identify a condition that ensures that RA and TSO coincide.

Theorem 6. Let $G = \langle A, po, rf\rangle$ be a WW-race free RA-coherent execution. Suppose that $G.\mathtt{U} = \emptyset$, and there exists a set $B \subseteq A$ of protected events such that the following hold:

- $(po \cup rf)^+$ is total on B.
- If a races with b in G, then either $a \in B$ or $b \in B$.
- If $\langle a, b\rangle \in rf \setminus po$, then either $a \in B$ or $b \in B$.

Then, G is TSO-coherent.

Remark 1. As the following example shows, the claim of Thm. 6 does not hold if we allow even a single update to an otherwise unused location:

$$\begin{array}{l} x := 1 \\ z_1 := 1 \\ \mathtt{wait}\ (z_2 = 1) \\ \mathtt{wait}\ (y = 0) \end{array} \Bigg\| \begin{array}{l} \mathtt{wait}\ (z_1 = 1) \\ z_2 := 1 \end{array} \Bigg\| \begin{array}{l} y := 1 \\ \langle f := 1\rangle \\ \mathtt{wait}\ (x = 0) \end{array}$$

This program may terminate under RA but not under TSO (where initially all variables are 0). In fact, under RA (or SRA), adding one update event to an otherwise unused location has no effect (whereas in the TSO-machine it forces the buffer to be flushed). This may indicate that the current semantics of updates in TSO is too strong, and weaker update instructions, as studied in [26], can be useful.

A consequence of Thm. 6 is that TSO and RA coincide for update-free "client-server programs". We say that a program follows the *client-server* discipline iff its locations can be partitioned into disjoint sets $X_1, \ldots, X_N$ such that thread S ("the server") writes only to variables in X_S, and each ("client") thread $i \neq S$ reads only from variables in $X_S \cup X_i$ and writes only to variables in X_i. In other words, client threads cannot communicate directly with other client threads, but only via the distinguished server thread. By taking the set B to be the set of events generated by the server, we can apply Thm. 6 for executions of such programs. As an example, Thm. 6 can be applied to a fence-free implementation of a simple version of the RCU mechanism (presented in Fig. 12). This

```
rcu_quiescent_state():
 rc[get_my_tid()] := gc;

synchronize_rcu():
 gc := gc + 1;
 for i := 1 to N do
   wait (rc[i] = gc);
```

Figure 12. Simple RCU implementation

$$
\begin{aligned}
G.isync &= \{\langle a,b\rangle \mid \exists c \in G.\texttt{isync}.\ \langle a,c\rangle \in deps \wedge \langle c,b\rangle \in po\}\\
G.lwsync &= \{\langle a,b\rangle \mid \exists c \in G.\texttt{lwsync}.\ \langle a,c\rangle \in po \wedge \langle c,b\rangle \in po\}\\
G.sync &= \{\langle a,b\rangle \mid \exists c \in G.\texttt{sync}.\ \langle a,c\rangle \in po \wedge \langle c,b\rangle \in po\}\\
G.ppo &= (deps \cup G.isync) \cap ((\mathcal{R}\times(\mathcal{R}\cup\mathcal{W}))\\
G.fence &= G.sync \cup (G.lwsync \cap ((\mathcal{R}\times(\mathcal{R}\cup\mathcal{W})) \cup (\mathcal{W}\times\mathcal{W}))\\
G.rfe &= rf \setminus po\\
G.hb &= G.ppo \cup G.fence \cup G.rfe\\
G.base &= G.rfe^{?}; G.fence; G.hb^{*}\\
G.po\text{-}aa &= po \cap ((At\cap\mathcal{W})\times(At\cap\mathcal{W}))\\
G.rmw &= ipo \cap ((At\cap\mathcal{R})\times(At\cap\mathcal{W}))
\end{aligned}
$$

Figure 13. Auxiliary notations for a Power execution $G = \langle A, po, deps, rf, At\rangle$. *ipo* denotes the set of immediate *po*-edges.

$$
\begin{aligned}
(\!|x|\!) &= \texttt{lwz}\ r_0, x \cdot \texttt{cmpw}\ r_0, r_0 \cdot \texttt{beq}\ L \cdot L\colon \cdot \texttt{isync}\\
(\!|x := v|\!) &= \texttt{li}\ r_0, v \cdot \texttt{lwsync} \cdot \texttt{stw}\ r_0, x\\
(\!|\langle x := x+1\rangle|\!) &= \begin{array}{l}\texttt{lwsync} \cdot Loop\colon \cdot\\ \texttt{lwarx}\ r_0, 0, x \cdot \texttt{addi}\ r_0, r_0, 1 \cdot \texttt{stwcx.}\ r_0, 0, x \cdot\\ \texttt{bne}\ Loop \cdot \texttt{cmpw}\ r_0, r_0 \cdot \texttt{beq}\ L \cdot L\colon \cdot \texttt{isync}\end{array}\\
(\!|\texttt{fence()}|\!) &= \texttt{sync}
\end{aligned}
$$

Figure 14. Compilation of reads, writes, atomic increments and fences to Power

version does not support readers going offline (and thus, requires them to constantly periodically call `rcu_quiescent_state()`), and assumes that one particular thread serves as the writer (that calls `synchronize_rcu()`). This allows reasoning about correctness under RA using existing techniques for TSO.

7. Relation to Power

In this section, we show that the SRA memory model is *equivalent* to the Power memory model of Alglave et al. [4] when following the standard compilation scheme of C11 release/acquire atomics to Power [7, 29], i.e., inserting an `lwsync` (lightweight synchronization) fence before every write and a conditional branch followed by an `isync` fence after every read. Atomic updates are compiled into two consecutive 'atomic' accesses: a load reserve (`lwarx`) instruction followed by a store conditional (`stwcx`) instruction, wrapped inside an "update loop" because the store-conditional may fail to perform the update. Fence commands are compiled to a single Power `sync` fence instruction. Figure 14 depicts some examples.

The Power model. To make our presentation self-contained, we briefly recall the definition of Alglave et al. [4]. The reader is referred to this paper for further explanations and details. A *Power execution* is taken to be a tuple $G = \langle A, po, deps, rf, At\rangle$ where:

- $A \subseteq \mathcal{A}_p$, where $\mathcal{A}_p$ is the set of *Power events*. $\mathcal{A}_p$ includes read and write events ($\mathcal{R}$ and $\mathcal{W}$, see §2), but no updates ($G.\mathtt{U} = \emptyset$). In addition, it includes events for the different Power fence instructions (whose types are `sync`, `lwsync`, or `isync`). We denote by $\mathcal{S}$ the set of all events with type `sync`.
- po and rf are program order and reads from edges (as before).
- $deps \subseteq po$ denotes the set of data, address and control *dependency edges* between instructions that Power implementations are guaranteed to preserve. In particular, the compilation of the acquire read induces a (control) dependency edge between the read and the `isync` events by introducing a "fake" compare-and-branch sequence.
- $At \subseteq A$ records the set of *atomic events* in G (i.e., the loads and stores belonging to an update).

A Power execution G induces a number of relations, defined in Fig. 13: instruction fence order ($G.isync$), lightweight fence order ($G.lwsync$), strong fence order ($G.sync$), preserved program order ($G.ppo$), fence order ($G.fence$), external reads-from ($G.rfe$), happens-before ($G.hb$), basic propagation ($G.base$), program order between atomic events ($G.po\text{-}aa$), and read-modify-write ($G.rmw$). For all these notations, we omit the "$G.$" prefix when it is clear from the context.

Definition 19. A Power execution $G = \langle A, po, deps, rf, At\rangle$ is called *Power-coherent* if hb is acyclic (*No-thin-air*) and there exist relations $\{co_x\}_{x\in\mathsf{Loc}}$, such that each co_x is a total strict order on $G.\mathtt{W}_x$ and the following hold:

- *SC-per-loc*: $po|_x \cup rf \cup fr \cup co$ is acyclic for every $x \in \mathsf{Loc}$, where $po|_x = \{\langle a,b\rangle \in po \mid loc(a) = loc(b) = x\}$.
- *Atomicity*: $rmw \cap (fre; coe) = \emptyset$.
- *Observation*: $fre; prop; hb^*$ is irreflexive.
- *Propagation*: $co \cup po\text{-}aa \cup prop$ is acyclic.

where $co = \bigcup_x co_x$, $coe = co \setminus po$, $fr = rf^{-1}; co$, $fre = fr \setminus po$, $prop = (base \cap (\mathcal{W}\times\mathcal{W})) \cup (chapo^{?}; base^{*}; sync; hb^{*})$, and $chapo = coe \cup fre \cup rfe \cup (coe; rfe) \cup (fre; rfe)$.

The relation co, called *coherence order*, is used instead of the modification order in the SRA model (see Prop. 7). The *SC-per-loc* property is similar to the x-modification order definition (see Prop. 2), but weaker as it rules out cycles with accesses of only one location. Nevertheless, it suffices to rule out the weak behavior of the (CoRR2) example. The *Atomicity* property basically corresponds to the third case of Fig. 6. The *Observation* property rules out the weak behavior of the MP example, making use of the relation $prop$, called *propagation order*. Finally, *Propagation* rules out the weak behavior of the (2+2W) example.

Alternative semantics for `sync` fences. We first prove a general property of the Power model: its `sync` fences are equivalent to release-acquire RMWs to a distinguished location. This entails that `sync` instructions (and events) are redundant in the Power model, and justifies our treatment of fences as syntactic sugar for the atomic assignment $\langle f := 0\rangle$. To prove this property, we define a correspondence between Power executions with `sync` events and those obtained from them by replacing these events with release-acquire updates. Given plain Power executions $G_s = \langle A_s, po_s, deps_s, \emptyset, At_s\rangle$ and $G_r = \langle A_r, po_r, deps_r, \emptyset, At_r\rangle$, we write $G_s \approx G_r$ if no event in G_s accesses location f and G_r is obtained from G_s by (1) adding an initialization event with label $\langle \mathtt{W}, f, 0\rangle$ and (2) splitting each `sync` event of G_s into a sequence of four po_r-consecutive events with labels $\langle\texttt{lwsync}\rangle$, $\langle \mathtt{R}, f, 0\rangle$, $\langle \mathtt{W}, f, 0\rangle$ and $\langle\texttt{isync}\rangle$, (3) adding a $deps_r$ edge between each $\langle \mathtt{R}, f, 0\rangle$ event and its corresponding $\langle \mathtt{W}, f, 0\rangle$ and `isync` events, and (4) including the introduced read and write events in At_r.

Theorem 7. Let G_s, G_r be plain Power executions, such that $G_s \approx G_r$. Then, G_s is Power-consistent iff G_r is Power-consistent.

Proof sketch. We write $a_s \approx a_r$ for corresponding events a_s in G_s and a_r in G_r. Let rf_s, such that $G'_s = \langle A_s, po_s, deps_s, rf_s, At_s\rangle$ is complete and Power-coherent, and let $\{co_x\}_{x\in \mathtt{Loc}\setminus\{f\}}$ be the coherence orders for G'_s that satisfy the conditions of Def. 19. We construct rf_r and co_f, such that $G'_r = \langle A_r, po_r, deps_r, rf_r, At_r\rangle$ is complete and $\{co_x\}_{x\in \mathtt{Loc}}$ satisfy the conditions of Def. 19 for G'_r. We note that *Propagation* entails that the relation

$$\begin{pmatrix} po_s; hb^*; (co \cup po\text{-}aa \cup (base \cap (\mathcal{W}\times\mathcal{W})))^*; \\ chapo^?; base^*; po_s \end{pmatrix} \cap (\mathcal{S}\times\mathcal{S})$$

is acyclic (using the notations of Fig. 13 and Def. 19 for G_s). Take T to be a strict total order on $G_s.\mathtt{sync} \times G_s.\mathtt{sync}$ extending this relation. We set $co_f = \{\langle a_r, b_r\rangle \in G_r.\mathtt{W}_f \times G_r.\mathtt{W}_f \mid \exists \langle a_s, b_s\rangle \in T.\ a_s \approx a_r \wedge b_s \approx b_r\}$ and $rf_r = rf_s \cup (ico_f; rmw^{-1})$, where ico_f consists of the immediate co_f-edges, and show that the conditions of Def. 19 are satisfied.

For the converse, we pick $rf_s = \{\langle a, b\rangle \in rf_r \mid loc(a) \neq f\}$ and show that the conditions of Def. 19 are satisfied for the relations $\{co_x\}_{x\in \mathtt{Loc}\setminus\{f\}}$. □

An interesting consequence of this theorem is that a Power program with just one `sync` fence is equivalent to the program where that `sync` fence is replaced with an `lwsync` fence. Indeed, the effect of a release-acquire RMW to an otherwise unused location is equivalent to the effect of a single `lwsync` fence.

Compilation soundness and completeness. To relate the SRA and Power models, we first define a correspondence between executions defined in §2 (that we refer to as *SRA executions*) and Power executions. Here, we consider only Power executions that result from the standard compilation scheme of C11 release-acquire accesses to Power (see Fig. 14). Further, observing that a program may terminate under Power iff it has a Power-consistent execution in which all update loops succeed immediately, we assume that update commands generate just one read event.

Definition 20. Given an SRA execution $G = \langle A, po, rf\rangle$ and a Power execution $G_p = \langle A_p, po_p, deps, rf_p, At\rangle$ with $G_p.\mathtt{sync} = \emptyset$, we write $G \sim G_p$ if there exist bijections:

$$w : G.\mathtt{W} \cup G.\mathtt{U} \to G_p.\mathtt{W} \quad f_w : G.\mathtt{W} \cup G.\mathtt{U} \to G_p.\mathtt{lwsync}$$
$$r : G.\mathtt{R} \cup G.\mathtt{U} \to G_p.\mathtt{R} \quad f_r : G.\mathtt{R} \cup G.\mathtt{U} \to G_p.\mathtt{isync}$$

such that the following hold, where $g : A_p \to A$ denotes the function $w^{-1} \cup f_w^{-1} \cup r^{-1} \cup f_r^{-1}$:

- $loc \circ w = loc, val_w \circ w = val_w, loc \circ r = loc, val_r \circ r = val_r$
- $po_p = \begin{pmatrix} \{\langle a, b\rangle \in A_p \times A_p \mid \langle g(a), g(b)\rangle \in po\} \\ \cup\{\langle f_w(a), w(a)\rangle \mid a \in G.\mathtt{W}\} \\ \cup\{\langle r(a), f_r(a)\rangle \mid a \in G.\mathtt{R}\} \\ \cup\{\langle f_w(a), r(a)\rangle \mid a \in G.\mathtt{U}\} \\ \cup\{\langle r(a), w(a)\rangle \mid a \in G.\mathtt{U}\} \\ \cup\{\langle w(a), f_r(a)\rangle \mid a \in G.\mathtt{U}\} \end{pmatrix}^+$
- $deps \supseteq \{\langle r(a), w(a)\rangle \mid a \in G.\mathtt{U}\}$
- $deps \supseteq \{\langle r(a), f_r(a)\rangle \mid a \in G.\mathtt{U} \cup G.\mathtt{R}\}$
- $rf_p = \{\langle w(a), r(b)\rangle \mid \langle a, b\rangle \in rf\}$
- $At = \{r(a) \mid a \in G.\mathtt{U}\} \cup \{w(a) \mid a \in G.\mathtt{U}\}$

Informally, one can easily see that for every SRA execution G of a program P, there exists a Power execution G_p of the compilation of P such that $G \sim G_p$, and vice versa. For Power executions that relate to SRA ones, we can simplify the Power model as follows:

Proposition 9. Let $G_p = \langle A, po, deps, rf, At\rangle$ be a Power execution that is $\sim$-related to some SRA execution. Then:

- $ppo = po \cap (\mathcal{R} \times (\mathcal{R} \cup \mathcal{W}))$.
- $fence = \begin{pmatrix} ((po \setminus rmw) \cap ((\mathcal{R} \cup \mathcal{W}) \times \mathcal{W})); \\ (po \cap (\mathcal{W} \times (\mathcal{R} \cup \mathcal{W})))^? \end{pmatrix}$.

Next, we prove that for related executions, SRA-coherence coincides with Power-coherence.

Theorem 8. Let $G \sim G_p$ be related SRA and Power executions. Then, G is SRA-coherent iff G_p is Power-coherent.

Proof sketch. For one direction, we show that any total order extending $\{\langle w^{-1}(a_p), w^{-1}(b_p)\rangle \mid \langle a_p, b_p\rangle \in (co \cup prop)^+\}$ is a modification order in G. For the converse, we satisfy the conditions of Def. 19 by taking $co_x = \{\langle w(a), w(b)\rangle \mid \langle a, b\rangle \in mo_x\}$. □

The exact correspondence between SRA and Power entails that SRA cannot be further strengthened without modifying the existing compilation scheme. Note that for RA only the right-to-left implication holds because SRA is strictly stronger than RA.

8. Related Work

Steinke and Nutt [34] developed a hierarchy of memory models expressed in terms of per-thread *view order*, that uniformly accounts for several classical consistency models (such as causal consistency [3] and PRAM consistency [21]). Except for SC, all these classical models are strictly weaker than RA. Roughly speaking, the reason is that a per-thread view induces a per-thread modification order, while RA assumes a global modification order for each location.

There are some other recent operational memory models. Many of these (e.g., [4, 8, 36]) are essentially wrappers around a declarative model: they build an execution graph in a stepwise fashion and check for consistency as new events are added to the execution. In contrast, our operational semantics for SRA does not refer to graphs and consistency axioms. An operational memory model for Power was defined by Sarkar et al. [28], and it is stronger than the model of [4]. Their operational model attempts to cover the entire Power architecture and is thus substantially more complex than ours. In addition, Zhang and Feng [40] present an operational weak memory model for Java that models weak behaviors by allowing event replay. The resulting model is much weaker than RA.

Wickerson et al. [39] proposed a new simpler semantics for SC accesses, including SC fences. While their semantics of SC fences is stronger than the C11's semantics, it is weaker than ours, and still allows weak behaviors for RA programs even when fences are placed between every two commands.

Weak consistency models appear also in the context of distributed systems, and more specifically, for replicated databases. These models include an additional construct of a *transaction*, which can be seen as an atomic block of instructions. Recently, Cerone et al. [11] proposed a framework for specifying such models, and studied several important ones in this framework. Among those, the closest to SRA is PSI ("Parallel Snapshot Isolation"). Stripping compound transactions out, and adapting to our terminology, PSI-coherence can be seen as a strengthening of SRA with the condition asserting that $po \cup rf \cup \bigcup_y mo_y \cup fr_x$ is acyclic for every $x \in \mathsf{Loc}$ (where, as above, $fr_x = (rf^{-1}; mo_x) \setminus Id_A$). The next example shows that PSI is strictly stronger than SRA:

$$\begin{array}{l|l} x := 1 & y := 2 \\ x := 2 & \mathtt{wait}\ (x = 1) \\ y := 1 & \mathtt{wait}\ (z = 1) \\ z := 1 & \mathtt{wait}\ (y = 2) \end{array}$$

This program may terminate under SRA (and, hence, also under Power), but not under PSI. In addition, it may terminate under TSO. Hence, we believe that PSI is too strong as a high-performance memory model for a programming language. Note, however, that PSI coincides with SRA for WW-race free executions.

Regarding the verification of RCU, Tassarotti et al. [35] and Lahav and Vafeiadis [18] use program logics (called GPS and

OGRA, respectively) to verify an RCU implementation under RA. However, their RCU implementation is based on the simple one shown in Fig. 12, while the more advanced implementation of Fig. 10 was not considered.

9. Conclusion

We have presented a new memory model, SRA, a slight strengthening of RA with many nice properties: (1) it is equivalent to RA in the absence of write-write races, (2) it has the same implementation cost on TSO and Power, (3) it allows the same local program transformations, and (4) it has an intuitive operational characterization. Moreover, we provided a simple notion of a strong memory fence that suffices for restoring sequential consistency assuming all (racy) accesses are RA, and presented reduction theorems to SC and TSO for particular classes of programs.

Practically speaking, the results of this paper suggest two independent strengthenings of the C11 memory model that do not incur any performance penalty: (1) adopting SRA instead of RA as the semantics of release-acquire accesses, and (2) modeling SC fences as release-acquire RMWs to a special location. Of course, a prerequisite for employing (1) in C11 is the investigation of our results in the broader context of the full C11 memory model, which includes more access kinds: non-atomic, SC atomic, relaxed atomic and consume atomic accesses. The most important of these are the non-atomic accesses because they avoid the synchronization overhead on Power, ARM, and Itanium. For them, the extension of our results is mostly straightforward (as sketched in the supplementary materiel). In fact, we believe that a programming language that has only release/acquire accesses (with SRA semantics) alongside non-atomic accesses (that should not be racy) should provide a reasonable balance between performance and programmability.

In addition, the simple operational model for SRA may pave the way for better understanding of programs running under weak memory, and development of essential verification methods and tools for these programs (e.g., model checking and program logics). Another theoretically interesting question is whether basic RA admits a similar machine semantics.

Acknowledgments

We would like to thank Marko Doko, Jeehoon Kang, Peter Sewell, Francesco Zappa Nardelli, and the POPL'16 reviewers for their helpful feedback. This work was supported by EC FET project ADVENT (308830).

References

[1] P. A. Abdulla, S. Aronis, M. F. Atig, B. Jonsson, C. Leonardsson, and K. Sagonas. Stateless model checking for TSO and PSO. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2015*, volume 9035 of *LNCS*, pages 353–367. Springer, 2015.

[2] P. A. Abdulla, M. F. Atig, and N. T. Phong. The best of both worlds: Trading efficiency and optimality in fence insertion for TSO. In *ESOP 2015: 24th European Symposium on Programming*, volume 9032 of *LNCS*, pages 308–332. Springer, 2015.

[3] M. Ahamad, G. Neiger, J. Burns, P. Kohli, and P. Hutto. Causal memory: definitions, implementation, and programming. *Distributed Computing*, 9(1):37–49, 1995.

[4] J. Alglave, L. Maranget, and M. Tautschnig. Herding cats: Modelling, simulation, testing, and data mining for weak memory. *ACM Trans. Program. Lang. Syst.*, 36(2):7:1–7:74, July 2014.

[5] J. Barnat, L. Brim, and V. Havel. LTL model checking of parallel programs with under-approximated TSO memory model. In *13th International Conference on Application of Concurrency to System Design, ACSD'13*, pages 51–59, July 2013.

[6] M. Batty, S. Owens, S. Sarkar, P. Sewell, and T. Weber. Mathematizing C++ concurrency. In *38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 55–66. ACM, 2011.

[7] M. Batty, K. Memarian, S. Owens, S. Sarkar, and P. Sewell. Clarifying and compiling C/C++ concurrency: From C++11 to POWER. In *39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, pages 509–520. ACM, 2012.

[8] M. Batty, K. Memarian, K. Nienhuis, J. Pichon-Pharabod, and P. Sewell. The problem of programming language concurrency semantics. In *24th European Symposium on Programming (ESOP 2015)*, volume 9032 of *LNCS*, pages 283–307. Springer, 2015.

[9] A. Bouajjani, R. Meyer, and E. Möhlmann. Deciding robustness against total store ordering. In *Automata, Languages and Programming*, volume 6756 of *LNCS*, pages 428–440. Springer, 2011.

[10] A. Bouajjani, E. Derevenetc, and R. Meyer. Checking and enforcing robustness against TSO. In *Programming Languages and Systems*, volume 7792 of *LNCS*, pages 533–553. Springer, 2013.

[11] A. Cerone, G. Bernardi, and A. Gotsman. A Framework for Transactional Consistency Models with Atomic Visibility. In *26th International Conference on Concurrency Theory (CONCUR 2015)*, volume 42 of *LIPIcs*, pages 58–71. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

[12] A. Dan, Y. Meshman, M. Vechev, and E. Yahav. Effective abstractions for verification under relaxed memory models. In *Verification, Model Checking, and Abstract Interpretation*, volume 8931 of *LNCS*, pages 449–466. Springer, 2015.

[13] M. Desnoyers, P. E. McKenney, A. S. Stern, M. R. Dagenais, and J. Walpole. User-level implementations of read-copy update. *IEEE Trans. Parallel Distrib. Syst.*, 23(2):375–382, 2012. .

[14] ISO/IEC 14882:2011. Programming language C++, 2011.

[15] ISO/IEC 9899:2011. Programming language C, 2011.

[16] S. Jagannathan, V. Laporte, G. Petri, D. Pichardie, and J. Vitek. Atomicity refinement for verified compilation. *ACM Trans. Program. Lang. Syst.*, 36(2):6:1–6:30, 2014.

[17] M. Kuperstein, M. Vechev, and E. Yahav. Partial-coherence abstractions for relaxed memory models. In *32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 187–198. ACM, 2011.

[18] O. Lahav and V. Vafeiadis. Owicki-gries reasoning for weak memory models. In *Automata, Languages, and Programming, ICALP'15*, volume 9135 of *LNCS*, pages 311–323. Springer, 2015.

[19] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Computers*, 28(9):690–691, 1979.

[20] A. Linden and P. Wolper. An automata-based symbolic approach for verifying programs on relaxed memory models. In *Model Checking Software*, volume 6349 of *LNCS*, pages 212–226. Springer, 2010.

[21] R. J. Lipton and J. S. Sandberg. PRAM: A scalable shared memory. Technical report, Technical Report CS-TR-180-88, Princeton University, 1988.

[22] W. Mansky and E. L. Gunter. Verifying optimizations for concurrent programs. In *First International Workshop on Rewriting Techniques for Program Transformations and Evaluation, WPTE 2014*, volume 40 of *OASICS*, pages 15–26. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[23] L. Maranget, S. Sarkar, and P. Sewell. A tutorial introduction to the ARM and POWER relaxed memory models. http://www.cl.cam.ac.uk/˜pes20/ppc-supplemental/test7.pdf, 2012.

[24] S. Owens. Reasoning about the implementation of concurrency abstractions on x86-TSO. In *ECOOP 2010: 24th European Conference on Object-Oriented Programming*, volume 6183 of *LNCS*, pages 478–503. Springer, 2010.

[25] S. Owens, S. Sarkar, and P. Sewell. A better x86 memory model: x86-TSO. In *TPHOLs 2009*, volume 5674 of *LNCS*, pages 391–407. Springer, 2009.

[26] B. Rajaram, V. Nagarajan, S. Sarkar, and M. Elver. Fast RMWs for TSO: Semantics and implementation. In *34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 61–72. ACM, 2013.

[27] T. Ridge. A rely-guarantee proof system for x86-TSO. In *VSTTE 2010*, volume 6217 of *LNCS*, pages 55–70. Springer, 2010.

[28] S. Sarkar, P. Sewell, J. Alglave, L. Maranget, and D. Williams. Understanding POWER multiprocessors. In *32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 175–186. ACM, 2011.

[29] S. Sarkar, K. Memarian, S. Owens, M. Batty, P. Sewell, L. Maranget, J. Alglave, and D. Williams. Synchronising C/C++ and POWER. In *33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '12, pages 311–322. ACM, 2012.

[30] J. Sevcik, V. Vafeiadis, F. Zappa Nardelli, S. Jagannathan, and P. Sewell. CompCertTSO: A verified compiler for relaxed-memory concurrency. *J. ACM*, 60(3):22, 2013.

[31] D. Shasha and M. Snir. Efficient and correct execution of parallel programs that share memory. *ACM Trans. Program. Lang. Syst.*, 10 (2):282–312, 1988.

[32] F. Sieczkowski, K. Svendsen, L. Birkedal, and J. Pichon-Pharabod. A separation logic for fictional sequential consistency. In *ESOP 2015*, volume 9032 of *LNCS*, pages 736–761. Springer, 2015.

[33] SPARC International Inc. *The SPARC Architecture Manual: Version 8*. Prentice-Hall, Inc., 1992. ISBN 0-13-825001-4.

[34] R. C. Steinke and G. J. Nutt. A unified theory of shared memory consistency. *J. ACM*, 51(5):800–849, Sept. 2004.

[35] J. Tassarotti, D. Dreyer, and V. Vafeiadis. Verifying read-copy-update in a logic for weak memory. In *36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2015, pages 110–120. ACM, 2015.

[36] A. Turon, V. Vafeiadis, and D. Dreyer. GPS: Navigating weak memory with ghosts, protocols, and separation. In *2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '14, pages 691–707. ACM, 2014. .

[37] V. Vafeiadis and F. Zappa Nardelli. Verifying fence elimination optimisations. In *18th International Conference on Static Analysis, SAS'11*, volume 6887 of *LNCS*, pages 146–162. Springer, 2011.

[38] V. Vafeiadis, T. Balabonski, S. Chakraborty, R. Morisset, and F. Zappa Nardelli. Common compiler optimisations are invalid in the C11 memory model and what we can do about it. In *42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 209–220. ACM, 2015.

[39] J. Wickerson, M. Batty, and A. F. Donaldson. Overhauling SC atomics in C11 and OpenCL. In *43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, 2016.

[40] Y. Zhang and X. Feng. An operational approach to happens-before memory model. In *7th International Symposium on Theoretical Aspects of Software Engineering, TASE 2013*, pages 121–128. IEEE Computer Society, 2013.

A. Extension with Non-Atomics

In this appendix, we explain how our results can be incorporated in a programming language that allows also non-atomic accesses in the style of C11. Such accesses provide no guarantees in case of data races, and these are considered programming errors.

Programming language. Programs annotate every memory access as atomic or non-atomic. For the simple programming language presented above, this can be done by requiring that every variable x mentioned in a command is annotated either by $x^{\texttt{at}}$ or $x^{\texttt{na}}$. In update commands $\langle x := e(x)\rangle$ and `when` $e(x)$ `do` $x := e'(x)$, all accesses should be annotated as atomic.

Executions. Events should include an additional boolean entry that can be either `at` or `na`. The set of plain executions of a given program is defined just as in §2, where atomic and non-atomic accesses naturally generate corresponding atomic and non-atomic events. For every model X that we consider (RA, SRA, SC, TSO), all of the properties of executions (completeness, X-coherence, X-consistency) are defined without any change, and they do not depend at all on the new `at`/`na` entry. These only affect the *X-safety* of a program, as defined next:

Definition 21. A program P is called *X-safe* if every complete and X-coherent execution $G' = \langle A, po, rf\rangle$, that extends a plain execution $G = \langle A, po, \emptyset\rangle$ of P with a set rf of reads-from edges, has no G'-racy non-atomic events (recall that an event is called G'-racy if it races with some other event in G' according to Def. 9).

Note that if X-coherence of a complete execution implies its Y-coherence for two models X and Y (i.e., X is not weaker than Y), then Y-safety guarantees X-safety. In particular, RA-safety implies SRA-safety, and SRA-safety implies SC-safety.

Now, following the C11 approach according to which unsafe programs may behave arbitrarily, we adapt Def. 7 as follows (for simplicity, we do not consider initialized locations here):

Definition 22. A program *may terminate under X* if either it has some X-consistent execution or it is not X-safe.

SRA. Note that all definitions and results that speak only about executions remain exactly the same.

To see that SRA and RA coincide for WW-protected programs, it remains to show that such programs are safe under RA if they are safe under SRA. This follows from the fact that extensions of executions of WW-protected programs are WW-race free, and for them SRA-coherence and RA-coherence coincide. Of course, the accesses to l used for the lock variable should be all atomic.

To see that the transformations described in §3.1 are valid also in this context, it remains to show that they preserve SRA-safety of programs, i.e., these transformations cannot introduce races on non-atomic accesses. Since we use exactly the same construction of [38], this follows from their results. In addition, the "strengthening" transformation, that changes a non-atomic access to an atomic one, is trivially valid.

Fences. It is easy to see that addition of fence commands in a program P preserves X-safety (for every model X that we consider). Thus, if we start with an RA-safe program, and add fences as described in §5, we obtain an RA-safe program. Hence, as in §5, we obtain that if a RA-safe program P may terminate under RA, then P', obtained from P by adding fences as described, may terminate under SC. Similarly, our reduction to SC given in 4 works out just the same, assuming the original program is RA-safe (or SRA-safe, as these coincide for WW-protected programs). A simple syntactic condition ensuring RA-safety (in the class of programs described after Prop. 3) is that each thread may non-atomically access locations that are not mentioned in commands of other threads and locations mentioned only in protected commands, non-atomically read from locations that are not modified by commands of other threads, and non-atomically read within critical sections from locations that are modified only by protected commands. The other accesses it can perform should be atomic. In the RCU implementation given in Fig. 10, we can make the reads of `rc[get my tid()]` at line 10 and the reads of `gc` at lines 14 and 17 non-atomic. The remaining accesses to shared variables must remain atomic.

Relation to TSO. Since TSO primitive instructions already guarantee SRA, the correctness of compilation of non-atomic access to TSO is trivial. For client-server programs, as shown in §6, TSO and RA coincide, but now we should also require that the program is RA-safe (otherwise, it might be TSO-safe and have undefined behavior under RA). This is obvious if non-atomic accesses are used only for local variables and for reads of X_{S} by the server. For example, in Fig. 12, the two reads of `gc` in `rcu_synchronize()` can be non-atomic; the remaining accesses have to be atomic.

Further work. To complete the extension with non-atomics, the following are left for a future work:

- The SRA-machine described in 4 should be used to detect races on non-atomic accesses. For example, before performing an atomic instruction the processor could check that there are no pending messages for the same location that are marked as non-atomic, and before performing a non-atomic one the processor could check that there are no pending messages for the same location of any kind. We conjecture that this would ensure that a program is SRA-safe iff all its executions in the SRA-machine do not detect a race.
- The correctness of the compilation to Power should be verified. We believe that this can be done along the lines of our current proof, although some of our technical developments should be adjusted. In particular, assuming that non-atomic accesses are compiled without Power fence instructions (otherwise, such accesses are useless), the simplifications in Prop. 9 do not hold.

B. General Useful Lemmas

Lemma 1. Let R be an acyclic relation on a set A. Suppose that R^+ contains some strict total order R_B on some subset B of A. Let $C = A \setminus B$, $R_C = R \cap (C \times C)$, $R_{BC} = R \cap (B \times C)$, and $R_{CB} = R \cap (C \times B)$. Then, the following hold:

- $R^+ \subseteq R_C^*; R_{CB}^?, R_B^?; R_{BC}^?; R_C^*$.
- $R^+ \cap (A \times B) \subseteq R_C^*; R_{CB}^?, R_B^?$.
- $R^+ \cap (B \times A) \subseteq R_B^?; R_{BC}^?; R_C^*$.

Lemma 2. Let R be a relation on a set A. Suppose that R^+ contains some strict total order R_B on some subset B of A. Let $C = A \setminus B$, $R_C = R \cap (C \times C)$, $R_{BC} = R \cap (B \times C)$, and $R_{CB} = R \cap (C \times B)$. Then, R is acyclic iff the following hold:

1. R is irreflexive.
2. R_C is acyclic.
3. $R_{CB}; R_{BC}; R_C^*$ is irreflexive.
4. $R_{CB}; R_B; R_{BC}; R_C^*$ is irreflexive.
5. $R_B; R$ is irreflexive.

Lemma 3. Let R be an acyclic relation on a set A, and T be a strict total order on some subset B of A. If $T; R^+$ is irreflexive, then $R \cup T$ is acyclic.

Lemma 4. Let R be a strict partial order on a set A, that is total on some subset B of A. Let $R' \subseteq B \times (A \setminus B)$, and let $\pi = R \cup (R' \setminus R^{-1})$. Then, π is acyclic, and $\langle a, b\rangle \in R$ whenever $\langle a, b\rangle \in \pi^+$ and $b \in B$.

Proof. We first show that π is acyclic. Let $\pi_B = \pi \cap (B \times B)$, $C = A \setminus B$, and $\pi_C = \pi \cap (C \times C)$. Note that $\pi_C \subseteq R$, and since R is transitive, $\pi_C^+ \subseteq R$ as well. We also clearly have that π and $\pi_b; \pi$ are irreflexive. Since π_B is a strict total order on B, by Lemma 2, to show that π is acyclic it remains to prove that the following hold:

1. π_C is cyclic. Indeed, $\pi_C \subseteq R$, and R is acyclic.
2. There do not exist $b \in B$ and $c_1, c_2 \in C$, such that $\langle c_1, b\rangle \in \pi$, $\langle b, c_2\rangle \in \pi$ and $\langle c_2, c_1\rangle \in \pi_C^*$. Suppose otherwise. Then, we must have $\langle c_1, b\rangle \in R$, and so $\langle c_2, b\rangle \in R$. It follows that $\langle b, c_2\rangle \notin R$, and $\langle b, c_2\rangle \notin R' \setminus R^{-1}$. This contradicts the fact that $\langle b, c_2\rangle \in \pi$.
3. There do not exist $b_1, b_2 \in B$ and $c_1, c_2 \in C$, such that $\langle c_1, b_1\rangle, \langle b_2, c_2\rangle \in \pi$, $\langle b_1, b_2\rangle \in \pi_B$, and $\langle c_2, c_1\rangle \in \pi_C^*$. Suppose otherwise. Then, we have $\langle c_2, b_2\rangle \in R$. It follows that $\langle b_2, c_2\rangle \notin R$ and $\langle b_2, c_2\rangle \notin R' \setminus R^{-1}$. This contradicts the fact that $\langle b_2, c_2\rangle \in \pi$.

Next, suppose that $\langle a, b\rangle \in \pi^+$ and $b \in B$. By Lemma 1, we have $\langle a, b\rangle \in (\pi \cap (C \times C))^*; (\pi \cap (C \times B))^?; (\pi \cap (B \times B))^?$. Since $\pi \cap (B \times B), \pi \cap (C \times C), \pi \cap (C \times B) \subseteq R$, and R is transitive, it follows that $\langle a, b\rangle \in R$. □

Lemma 5. Let R_1 and R_2 be two strict partial orders on a set A. Let $B \subseteq A$, such that $R_2 \subseteq B \times A$. Then, $R_1 \cup R_2$ is acyclic iff $(R_2; R_1) \cap (B \times B)$ is acyclic.

C. Proofs for §2 (RA Memory Model)

Proof (of Prop. 2). Assuming that $po \cup rf \cup mo_x \cup fr_x$ is acyclic, we clearly have that $mo_x; (po \cup rf)^+$ is irreflexive. In addition, a violation of the third requirement in Def. 4 immediately entails a cycle in $po \cup rf \cup mo_x \cup fr_x$. For the converse, suppose that mo_x is an x-modification order in G. By definition, it is a strict total order on $G.\mathtt{W}_x \cup G.\mathtt{U}_x$. It remains to show that $po \cup rf \cup mo_x \cup fr_x$ is acyclic.

Claim 1: $po \cup rf \cup mo_x$ is acyclic.
Proof: By Lemma 3, since $po \cup rf$ is acyclic and $mo_x; (po \cup rf)^+$ is irreflexive.

Claim 2: $fr_x; (po \cup rf \cup mo_x)^*$ is irreflexive.
Proof: Let $a, b \in A$ such that $\langle a, b\rangle \in fr_x$, and $\langle b, a\rangle \in (po \cup rf \cup mo_x)^*$. Then, $a \neq b$, and there exists $a_1 \in A$ such that $\langle a_1, a\rangle \in rf$, and $\langle a_1, b\rangle \in mo_x$. Since mo_x is an x-modification order in G, we cannot have $\langle b, a\rangle \in (po \cup rf)^+ \cup mo_x$. By Lemma 1, it follows that $\langle b, a\rangle \in mo_x; (po \cup rf)^+$. Let $a_2 \in A$ such that $\langle b, a_2\rangle \in mo_x$, and $\langle a_2, a\rangle \in (po \cup rf)^+$. Since mo_x is transitive, we have $\langle a_1, a_2\rangle \in mo_x$. Since $\langle a_1, a\rangle \in rf$ and $\langle a_2, a\rangle \in (po \cup rf)^+$, this contradicts the fact that mo_x is an x-modification order in G.

Claim 3: If $\langle a, b\rangle \in fr_x$ and $\langle c, d\rangle \in fr_x$, then either $\langle a, d\rangle \in fr_x$ or $\langle c, b\rangle \in fr_x$.
Proof: Let $a, b, c, d \in A$ such that $\langle a, b\rangle \in fr_x$, and $\langle c, d\rangle \in fr_x$. Then, $a \neq b$, $c \neq d$, and there exist $a_1, a_2 \in A$ such that $\langle a_1, a\rangle \in rf$, $\langle a_1, b\rangle \in mo_x$, $\langle a_2, c\rangle \in rf$ and $\langle a_2, d\rangle \in mo_x$. Suppose first that $a = d$. Since mo_x is an x-modification order in G, $\langle a_1, a\rangle \in rf$, and $\langle a_1, b\rangle \in mo_x$, we cannot have $\langle b, a\rangle \in mo_x$. Hence, since mo_x is total on $G.\mathtt{W}_x \cup G.\mathtt{U}_x$, we have $\langle a, b\rangle \in mo_x$. Since mo_x is transitive, we have $\langle a_2, b\rangle \in mo_x$. Since mo_x is an x-modification order in G, we cannot have $b = c$, and so $\langle c, b\rangle \in fr_x$. Symmetrically, if $b = c$, we obtain $\langle a, d\rangle \in fr_x$. Now, suppose that $a \neq d$ and $b \neq c$. If $a_1 = a_2$, then $\langle a, d\rangle \in fr_x$, and we are done. Otherwise, since mo_x is total on $G.\mathtt{W}_x \cup G.\mathtt{U}_x$, we have either $\langle a_1, a_2\rangle \in mo_x$ or $\langle a_2, a_1\rangle \in mo_x$. In the first case, we obtain $\langle a_1, d\rangle \in mo_x$ (since mo_x is transitive), and so $\langle a, d\rangle \in fr_x$. Symmetrically, if $\langle a_2, a_1\rangle \in mo_x$, then $\langle a_2, b\rangle \in mo_x$, and so $\langle c, b\rangle \in fr_x$.

Finally, note that the fact that $po \cup rf \cup mo_x \cup fr_x$ is acyclic follows from the previous claims. Indeed, using Claim 3, a cycle with a minimal number of fr_x edges contains at most one such edge, which is not possible according to Claims 1 and 2. □

D. Proofs for §3 (SRA Memory Model)

D.1 Validity of Local Program Transformations

Proof sketch (of Prop. 4). In all cases, we use exactly the same construction used in the corresponding proof in [38], and it remains to show that the additional condition of SRA is also preserved.

- Let $A' = A \setminus \{a\}$, $po_a = po \cap (A' \times A')$, and $G_a = G \setminus \{a\} = \langle A', po_a, \emptyset\rangle$. Let rf be a set of reads-from edges such that $G'_a = \langle A', po_a, rf\rangle$ is complete and SRA-coherent. For every $x \in \mathsf{Loc}$, let mo_x be an x-modification order in G'_a, such that $po_a \cup rf \cup \bigcup_x mo_x$ is acyclic. We show that $G' = \langle A, po, rf\rangle$ is SRA-coherent. Let $y = loc(a)$. For every $x \in \mathsf{Loc} \setminus \{y\}$, we take ${mo_x}'$ to be identical to mo_x, and define ${mo_y}' = mo_y \cup \{\langle a, c\rangle \mid \langle b, c\rangle \in {mo_y}^?\} \cup \{\langle c, a\rangle \mid \langle c, b\rangle \in mo_y\}$. To see that $po \cup rf \cup \bigcup_x {mo_x}'$ is acyclic, it suffices to note that any cycle that goes through a implies a cycle through b in $po_a \cup rf \cup \bigcup_x mo_x$. Indeed, (i) $\langle c, a\rangle \in po$ implies $\langle c, b\rangle \in po_a$; (ii) $\langle c, a\rangle \in {mo_y}'$ implies $\langle c, b\rangle \in mo_y$; (iii) $\langle a, c\rangle \in po$ implies $\langle b, c\rangle \in po_a^?$; and (iv) $\langle a, c\rangle \in {mo_y}'$ implies $\langle b, c\rangle \in {mo_y}^?$.
- Let $A' = A \setminus \{a\}$, $po_a = po \cap (A' \times A')$, and $G_a = G \setminus \{a\} = \langle A', po_a, \emptyset\rangle$. Let rf_a be a set of reads-from edges such that $G'_a = \langle A', po_a, rf_a\rangle$ is complete and SRA-coherent. Since G'_a is complete, there exists some $d \in A'$, such that $\langle d, b\rangle \in rf_a$. Since $loc(a) = loc(b)$ and $val_r(a) = val_r(b)$, we have that $\langle d, a\rangle$ is also a reads-from edge. Let $rf = rf_a \cup \{\langle d, a\rangle\}$. Then, $G' = \langle A, po, rf\rangle$ is complete. We show that it is also SRA-coherent. For every $x \in \mathsf{Loc}$, let mo_x be an x-modification

order in G'_a, such that $po_a \cup rf_a \cup \bigcup_x mo_x$ is acyclic. By the same proof for RA, we have that each mo_x is also an x-modification order in G'. To see that $po \cup rf \cup \bigcup_x mo_x$ is acyclic, it suffices to note that any cycle that goes through a implies a cycle through b in $po_a \cup rf_a \cup \bigcup_x mo_x$. Indeed, (i) $\langle c,a\rangle \in po$ implies $\langle c,b\rangle \in po_a$; (ii) $\langle c,a\rangle \in rf$ implies $\langle c,b\rangle \in rf_a$; and (iii) $\langle a,c\rangle \in po$ implies $\langle b,c\rangle \in po_a^?$.

- Let $A' = A \setminus \{b\}$, $po_b = po \cap (A' \times A')$, and $G_b = G \setminus \{b\} = \langle A', po_b, \emptyset\rangle$. Let rf_b be a set of reads-from edges such that $G'_b = \langle A', po_b, rf_b\rangle$ is complete and SRA-coherent. Since $loc(a) = loc(b)$ and $val_w(a) = val_r(b)$, we have that $\langle a,b\rangle$ is a reads-from edge. Let $rf = rf_b \cup \{\langle a,b\rangle\}$. Then, $G' = \langle A, po, rf\rangle$ is complete. We show that it is also SRA-coherent. For every $x \in \mathsf{Loc}$, let mo_x be an x-modification order in G'_b, such that $po_b \cup rf_b \cup \bigcup_x mo_x$ is acyclic. By the same proof for RA, we have that each mo_x is also an x-modification order in G'. To see that $po \cup rf \cup \bigcup_x mo_x$ is acyclic, it suffices to note that any cycle that goes through b implies a cycle through a in $po_b \cup rf_b \cup \bigcup_x mo_x$. Indeed, (i) $\langle c,b\rangle \in po$ implies $\langle c,a\rangle \in po_b^?$; (ii) $\langle c,b\rangle \in rf$ implies $c = a$; and (iii) $\langle b,c\rangle \in po$ implies $\langle a,c\rangle \in po_b$. □

Proof sketch (of Prop. 5). Let $po_0 = (po \cup \{\langle b,a\rangle\}) \setminus \{\langle a,b\rangle\}$, and $G_0 = \langle A, po_0, \emptyset\rangle$. Let rf be a set of reads-from edges such that $G'_0 = \langle A, po_0, rf\rangle$ is complete and SRA-coherent. Then, $G' = \langle A, po, rf\rangle$ is also complete. For every $x \in \mathsf{Loc}$, let mo_x be an x-modification order in G'_0, such that $po_0 \cup rf \cup \bigcup_x mo_x$ is acyclic. We claim that each mo_x is also an x-modification order in G', and we also have that $po \cup rf \cup \bigcup_x mo_x$ is acyclic. Since we use exactly the same construction in the corresponding proof in [38] (for local accesses we use the construction given in [38] for non-atomics), we only show here that $po \cup rf \cup \bigcup_x mo_x$ is acyclic. Consider the two possible cases:

- Suppose that $loc(a)$ is local in G (the case that $loc(b)$ is symmetric). To see that $po \cup rf \cup \bigcup_x mo_x$ is acyclic, it suffices to note that the fact that $loc(a)$ is local implies that $\langle c,a\rangle \in po$ whenever $\langle c,a\rangle \in po \cup rf \cup \bigcup_x mo_x$. Hence, any cycle that uses the po-edge $\langle a,b\rangle$ induces a cycle in $po_0 \cup rf \cup \bigcup_x mo_x$ that skips this edge, and goes directly from the event c before a to b.
- Suppose that $a \in G.\mathtt{W}$ and $b \in G.\mathtt{R}$. In this case, to see that $po \cup rf \cup \bigcup_x mo_x$ is acyclic, it suffices to note that any cycle that uses the po-edge $\langle a,b\rangle$ must continue with a po-edge $\langle b,c\rangle$. Then, a corresponding cycle in $po_0 \cup rf \cup \bigcup_x mo_x$ can use the po-edge $\langle a,c\rangle$ instead. □

D.2 An Alternative Formulation

Proof (of Prop. 7). Suppose that there exists a relation mo that is a modification order in G. For every $x \in \mathsf{Loc}$, let $mo_x = mo \cap ((G.\mathtt{W}_x \cup G.\mathtt{U}_x) \times (G.\mathtt{W}_x \cup G.\mathtt{U}_x))$. By Prop. 6, mo_x is an x-modification order for every $x \in \mathsf{Loc}$. By Lemma 3, since $po \cup rf$ is acyclic, the fact that $mo; (po \cup rf)^+$ is irreflexive entails that $po \cup rf \cup mo$ is acyclic. Hence, $po \cup rf \cup \bigcup_x mo_x$ is acyclic.

For the converse, assume an x-modification order mo_x in G for every $x \in \mathsf{Loc}$, such that $R = po \cup rf \cup \bigcup_x mo_x$ is acyclic. Clearly, $po \cup rf$ is acyclic. Let mo be any strict total order on $G.\mathtt{W} \cup G.\mathtt{U}$ extending $R^+ \cap ((G.\mathtt{W} \cup G.\mathtt{U}) \times (G.\mathtt{W} \cup G.\mathtt{U}))$. It is easy to verify that mo is a modification order in G. □

E. Proofs for §4 (An Equivalent Operational Model)

Notation 4. Given a history $H = \langle B, \leq\rangle$, we denote the set $\{\langle a,b\rangle \in \mathcal{A} \times \mathcal{A} \mid H.last(b) = a\}$ by $H.rf$.

To prove Thm. 2, we use the following lemma:

Lemma 6. Let $H = \langle B, \leq\rangle$ be a legal and G-suitable history for some plain execution $G = \langle A, po, \emptyset\rangle$. Then, for every $a \in \mathcal{W} \cup \mathcal{U}$ and $b@i \in B$, if $\langle a,b\rangle \in (po \cup H.rf)^+$, then $a@i \in B$ and $a@i < b@i$.

Proof. We use induction on the length of the $po \cup H.rf$ path from a to b. For the base case, let $a \in \mathcal{W} \cup \mathcal{U}$ and $b@i \in B$, such that $\langle a,b\rangle \in po \cup H.rf$. Since H is legal, to prove that $a@i \in B$ and $a@i < b@i$, it suffices to show that $a@tid(b) < b@tid(b)$. First, if $\langle a,b\rangle \in po$, we have $a@tid(b) < b@tid(b)$ since H is G-suitable. Otherwise, $\langle a,b\rangle \in H.rf$, and by definition, we have $a@tid(b) < b@tid(b)$.

For the induction step, suppose the claim holds for paths of size $< n$, let $a_1 \in \mathcal{W} \cup \mathcal{U}$ and $a_n@i \in B$, such that there exist events $a_2, \ldots, a_{n-1}$ satisfying $\langle a_k, a_{k+1}\rangle \in po \cup H.rf$ for every $1 \leq k \leq n-1$. We show that $a_1@i \in B$ and $a_1@i < a_n@i$. Consider two cases:

- $a_k \in \mathcal{W} \cup \mathcal{U}$ for some $1 < k < n$. In this case, by the induction hypothesis, $a_k@i \in B$ and $a_k@i < a_n@i$. Again, by the induction hypothesis, we obtain that $a_1@i \in B$ and $a_1@i < a_k@i$. It follows that $a_1@i < a_n@i$.
- $a_k \in \mathcal{R}$ for every $1 < k < n$. In this case, we must have $\langle a_2, a_n\rangle \in po$. Now, if $\langle a_1, a_2\rangle \in po$, then we are done using the base case. Otherwise, $\langle a_1, a_2\rangle \in H.rf$. Let $j = tid(a_2)$. Using the base case, we have that $a_1@j < a_2@j$. Since H is G-suitable, we have that $a_2@j < a_n@j$. It follows that $a_1@j < a_n@j$. Since H is legal, we obtain that $a_1@i \in B$ and $a_1@i < a_n@i$. □

Proof (of Thm. 2). Let $G = \langle A, po, \emptyset\rangle$, and $G_0 = \sigma; G = \langle A_0, po_0, \emptyset\rangle$. For every $x \in \mathsf{Loc}$ with $\sigma(x) \neq \bot$, let $a_x = \langle 0, 0, \langle \mathtt{W}, x, \sigma(x)\rangle\rangle$ (that is, the event in A_0 that initializes x). Let $A_\sigma = \{a_x \mid \sigma(x) \neq \bot\}$, and $po_\sigma = A_\sigma \times A$. Then, $A_0 = A_\sigma \uplus A$ and $po_0 = po_\sigma \uplus po$.

($\Rightarrow$) Suppose that G_0 is SRA-consistent. Let $rf_0 \subseteq A_0 \times A_0$ such that $G'_0 = \langle A_0, po_0, rf_0\rangle$ is a complete and SRA-coherent execution. Let $rf = rf_0 \cap (A \times A)$ and $rf_\sigma = rf_0 \cap (A_\sigma \times A)$ (note that $rf_0 = rf_\sigma \uplus rf$). By Proposition 7, there exists a relation mo_0 that is a modification order in G'_0. Let $mo = mo_0 \cap (A \times A)$ and $mo_\sigma = mo_0 \cap (A_\sigma \times A_0)$. It is easy to verify that $mo_0 = mo_\sigma \uplus mo$ (since $mo; po$ is irreflexive); $\langle a_x, a\rangle \in mo_\sigma$ for every $x \in \mathsf{Loc}$ and $a \in G.\mathtt{W} \cup G.\mathtt{U}$; and mo is a modification order in $G' = \langle A, po, rf\rangle$. We construct a $\langle G, \sigma\rangle$-legal history $H = \langle B, \leq\rangle$.

First, for every $1 \leq i \leq N$, let $A_i = \{a \in A \mid tid(a) = i\}$, $A'_i = A_i \cup G.\mathtt{W} \cup G.\mathtt{U}$, and $\prec_i = ((hb \cap (A'_i \times A'_i)) \cup ((A_i \times (A'_i \setminus A_i)) \setminus hb^{-1}))^+$, where $hb = (po \cup rf)^+$. For every $1 \leq i \leq N$, $hb \cap (A'_i \times A'_i)$ is a strict partial order on A'_i, that is total on A_i. Hence, by Lemma 4, for every $1 \leq i \leq N$, $\prec_i$ is a strict partial order, and $\langle a,b\rangle \in hb$ whenever $a \prec_i b$ and $b \in A_i$.

Now, let $B = \{a@i \mid 1 \leq i \leq N, a \in A'_i\}$ (note that for every $a@i \in B$, if $tid(a) \neq i$, then $a \in \mathcal{W} \cup \mathcal{U}$). Let $B' = \{a@i \in B \mid tid(a) = i, a \in G.\mathtt{W} \cup G.\mathtt{U}\}$, $C = B \setminus B'$, and define the following relations:

1. $T_1 = \{\langle a@i, b@j\rangle \in B \times B \mid i = j, a \prec_i b\}$.
2. $T_2 = \{\langle a@i, b@j\rangle \in B' \times C \mid a = b\}$.
3. $T_3 = \{\langle a@i, b@j\rangle \in B' \times B' \mid \langle a,b\rangle \in mo\}$.

We take $\leq$ to be any total order on B extending $(T_1 \cup T_2 \cup T_3)^+$. To show that such an order exists, we prove that $T_1 \cup T_2 \cup T_3$ is acyclic. Note that since mo is a strict total order on $G.\mathtt{W} \cup G.\mathtt{U}$, T_3 is a strict total order on B'. By Lemma 2, to show that $T_1 \cup T_2 \cup T_3$ is acyclic it suffices to prove the following:

1. $T_1 \cup T_2 \cup T_3$ is irreflexive. Indeed, T_1, T_2, and T_3 are all irreflexive.
2. $(T_1 \cup T_2 \cup T_3) \cap (C \times C)$ is acyclic. Indeed, $(T_1 \cup T_2 \cup T_3) \cap (C \times C) \subseteq T_1$, and acyclicity of T_1 follows directly from the fact that $\prec_i$ is acyclic for every $1 \leq i \leq N$.
3. For every $a@i \in B'$ and $b_1@j_1, b_2@j_2 \in C$, it is not the case that $\langle b_1@j_1, a@i\rangle, \langle a@i, b_2@j_2\rangle \in T_1 \cup T_2 \cup T_3$ and $\langle b_2@j_2, b_1@j_1\rangle \in ((T_1 \cup T_2 \cup T_3) \cap (C \times C))^*$. Otherwise, we must have $\langle b_1@j_1, a@i\rangle \in T_1$, and $\langle b_2@j_2, b_1@j_1\rangle \in T_1^*$, and since T_1 is transitive, we obtain $\langle b_2@j_2, a@i\rangle \in T_1$ Hence, $j_2 = i$, and so $\langle a@i, b_2@j_2\rangle \notin T_2$. It follows that $\langle a@i, b_2@j_2\rangle \in T_1$, which contradicts the fact that T_1 is acyclic.
4. For every $a_1@i_1, a_2@i_2 \in B'$ and $b_1@j_1, b_2@j_2 \in C$, it is not the case that $\langle b_1@j_1, a_1@i_1\rangle, \langle a_2@i_2, b_2@j_2\rangle \in T_1 \cup T_2 \cup T_3$, $\langle a_1@i_1, a_2@i_2\rangle \in T_3$, and $\langle b_2@j_2, b_1@j_1\rangle \in ((T_1 \cup T_2 \cup T_3) \cap (C \times C))^*$. Assume otherwise. Then, $\langle b_1@j_1, a_1@i_1\rangle \in T_1$, and $\langle b_2@j_2, b_1@j_1\rangle \in T_1$, and hence, $\langle b_2@j_2, a_1@i_1\rangle \in T_1^*$. Since $\langle a_1@i_1, a_2@i_2\rangle \in T_3$, we have $\langle a_1, a_2\rangle \in mo$. We show that $\langle a_2, a_1\rangle \in hb$. This contradicts the fact that $mo; hb$ is irreflexive. Since $a_1@i_1 \in B'$, we have $tid(a_1) = i_2$, and it suffices to prove that $a_2 \prec_{i_1} a_1$. First, if $\langle a_2@i_2, b_2@j_2\rangle \in T_1$, then since T_1 is transitive, we have $\langle a_2@i_2, a_1@i_1\rangle \in T_1$. Hence, $i_1 = i_2$ and $a_2 \prec_{i_1} a_1$. Otherwise, we have $\langle a_2@i_2, b_2@j_2\rangle \in T_2$. Hence, $a_2 = b_2$, and so $\langle a_2@j_2, a_1@i_1\rangle \in T_1$. In this case we have $j_2 = i_1$ and $a_2 \prec_{i_1} a_1$.
5. For every $a_1@i_1, a_2@i_2 \in B'$, it is not the case that $\langle a_1@i_1, a_2@i_2\rangle \in T_3$ and $\langle a_2@i_2, a_1@i_1\rangle \in T_1 \cup T_2 \cup T_3$. Since T_3 is acyclic and $a_1@i_2 \notin C$, we have $\langle a_2@i_2, a_1@i_1\rangle \in T_1$. Hence, $i_1 = i_2$ and $a_2 \prec_{i_1} a_1$. Since $a_1@i_1 \in B'$, we have $tid(a_1) = i_2$, and so, $\langle a_2, a_1\rangle \in hb$. Since $\langle a_1@i_1, a_2@i_2\rangle \in T_3$, we also have $\langle a_1, a_2\rangle \in mo$. Again, this contradicts the fact that $mo; hb$ is irreflexive.

Next, note that H is G-suitable. Indeed, for every $a@i \in B$, we have that $a \in A'_i$, and so $a \in A$. In addition, for every $a \in A$, we have $a \in A'_{tid(a)}$, and so $a@tid(a) \in B$. Finally, for every $a, b \in A$, if $\langle a, b\rangle \in po$, then $a \prec_{tid(a)} b$, and so $a@tid(a) < b@tid(b)$.

To show that H is also legal and σ-suitable, we first prove the following properties:

- For every $a@i, b@i \in B$, if $\langle a, b\rangle \in po$ then $a@i < b@i$. Indeed, in this case we have $a, b \in A'_i$, and $a \prec_i b$. Hence, $\langle a@i, b@i\rangle \in T_1$, and so $a@i < b@i$.
- For every $a \in G.\mathtt{R} \cup G.\mathtt{U}$, if $a \in dom(H.last)$ then $\langle H.last(a), a\rangle \in rf$, and otherwise $\langle a_{loc(a)}, a\rangle \in rf_\sigma$. Indeed, let $a \in G.\mathtt{R} \cup G.\mathtt{U}$. Let $i = tid(a)$ and $x = loc(a)$. Since G'_0 is complete, we have $\langle b, a\rangle \in rf_0$ for some $b \in A_0$. We show that $b = H.last(a)$ if $a \in dom(H.last)$, and $b = a_x$ otherwise. Note that $\langle c, a\rangle \in hb$ for every $c \in H.before(a)$. Indeed, if $tid(c) = i$, then $c@i < a@i$ entails that $\langle c, a\rangle \in po \subseteq hb$. Otherwise, $c \notin A_i$, and if $\langle c, a\rangle \notin hb$, we have that $a \prec_i c$, and so $a@i < c@i$. Next, consider the two possible cases:
  - $\langle b, a\rangle \in rf_\sigma$. In this case, we must have $b = a_x$, and it remains to show that $H.before(a)$ is empty, and so $a \notin dom(H.last)$. Suppose for contradiction that there exists some $c \in H.before(a)$. Then, we have $\langle c, a\rangle \in hb$. Since mo_0 is a modification order in G'_0, we have $\langle c, a_x\rangle \in mo_0$. However, since $\langle a_x, a\rangle \in mo_\sigma$ for every $a \in G.\mathtt{W} \cup G.\mathtt{U}$, we have $\langle a_x, c\rangle \in mo_0$, and this contradicts the fact that mo is acyclic.
  - $\langle b, a\rangle \in rf$. In this case, $b \in H.before(a)$. Indeed, since $\langle b, a\rangle \in rf$, we have $b \in \mathcal{W} \cup \mathcal{U}$ and $loc(b) = loc(a)$. Since $\langle b, a\rangle \in rf \subseteq hb$, we have $\langle b@i, a@i\rangle \in T_1$, and so $b@i < a@i$. To see that $b@tid(b)$ is the $\leq$-maximal action in $\{c@tid(c) \mid c \in H.before(a)\}$, consider any action $c@tid(c) \neq b@tid(b)$ with $c \in H.before(a)$. As shown above, we also have $\langle c, a\rangle \in hb$. Since $\langle c, a\rangle \in hb$ and mo is a modification order in G', we have $\langle c, b\rangle \in mo$. Hence, $\langle c@tid(c), b@tid(b)\rangle \in T_3$, and so $c@tid(c) < b@tid(b)$.

Next, we show that H is σ-suitable. Let $a@tid(a) \in B$ such that $a \in \mathcal{R} \cup \mathcal{U}$, and $a \notin dom(H.last)$. By the claim above, we have $\langle a_{loc(a)}, a\rangle \in rf_\sigma$. Thus, $\langle a_{loc(a)}, a\rangle$ is a reads-from edge, and so $val_r(a) = \sigma(loc(a))$.

Finally, we show that H is legal:

- As shown above, for every $a \in dom(H.last)$, we have $\langle H.last(a), a\rangle \in rf$. Thus, $\langle H.last(a), a\rangle$ is a reads-from edge, and so $val_r(a) = val_w(H.last(a))$.
- Let $a \in \mathcal{W} \cup \mathcal{U}$ and $b@i \in B$, such that $a@j < b@j$, where $j = tid(b)$. By definition, we have $a@i \in B$. We show that $a@i < b@i$. If $tid(a) = j$, then we must have $\langle a, b\rangle \in po$, and so, $a@i < b@i$. Assume now that $tid(a) \neq j$, and suppose for contradiction that $a@i \not< b@i$. Then, $\langle a@i, b@i\rangle \notin T_1$. Hence, $a \not\prec_i b$, and so $\langle a, b\rangle \notin hb$. Hence, $\langle b, a\rangle \in (A_j \times (A'_j \setminus A_j)) \setminus hb^{-1}$, and so $b \prec_j a$. It follows that $\langle b@j, a@j\rangle \in T_1$, and so $b@j < a@j$, which contradicts our assumption.
- Let $a \in \mathcal{W} \cup \mathcal{U}$ and $b \in \mathcal{U}$, such that $loc(a) = loc(b)$ and $a@tid(a) < b@tid(b)$. We show that $b \in dom(H.last)$, and $a@tid(a) \leq H.last(b)@tid(H.last(b))$. Since we have $a@tid(a) < b@tid(b)$, it follows that $\langle a, b\rangle \in mo$. To see that $b \in dom(H.last)$, note that otherwise we have $\langle a_{loc(a)}, b\rangle \in rf_0$, $\langle a_{loc(a)}, a\rangle \in mo_0$, and $\langle a, b\rangle \in mo_0$, and this contradicts the fact that mo_0 is a modification order in G'_0. Let $c = H.last(b)$. Then, $\langle c, b\rangle \in rf$. Therefore, since mo is a modification order in G', we have $\langle c, a\rangle \notin mo$. It follows that $a = c$ or $\langle a, c\rangle \in mo$, and so $a@tid(a) \leq c@tid(c)$.
- Let $a@i \in B$. If $i = tid(a)$, we clearly have $a@tid(a) \leq a@i$. Otherwise, we have $a \in G.\mathtt{W} \cup G.\mathtt{U}$. Hence, we have $\langle a@tid(a), a@i\rangle \in T_2$, and so $a@tid(a) < a@i$.

($\Leftarrow$) Let $H = \langle B, \leq\rangle$ be a $\langle G, \sigma\rangle$-legal history. Let $rf = H.rf = \{\langle H.last(a), a\rangle \mid a \in dom(H.last)\}$. Since H is legal and G-suitable, for every $a \in dom(H.last)$, we have that $H.last(a) \in G.\mathtt{W} \cup G.\mathtt{U}$, $loc(H.last(a)) = loc(a)$, and $val_r(a) = val_w(H.last(a))$. Hence, rf is a set of reads-from edges. Let $rf_\sigma = \{\langle a_{loc(a)}, a\rangle \mid a \in G.\mathtt{R} \cup G.\mathtt{U} \setminus dom(H.last)\}$. Since H is σ-suitable and G-suitable, for every $a \in G.\mathtt{R} \cup G.\mathtt{U} \setminus dom(H.last)$, we have that $val_r(a) = \sigma(loc(a)) = val_w(a_{loc(a)})$. Hence, rf_σ is also a set of reads-from edges. Let $rf_0 = rf \cup rf_\sigma$. Then, $G'_0 = \langle A_0, po_0, rf_0\rangle$ is a complete execution.

Next, we show that G'_0 is SRA-coherent. First, we prove that $po_0 \cup rf_0$ is acyclic. Since the events in A_σ have no incoming po_0 or rf_0 edges, a cycle in $po_0 \cup rf_0$ cannot involve any rf_σ or po_σ edges. Hence, it suffices to show that $po \cup rf$ is acyclic. Since both po and rf are strict partial orders on A, and $rf \subseteq (G.\mathtt{W} \cup G.\mathtt{U}) \times A$, by Lemma 5, it suffices to show that $(rf; po) \cap ((G.\mathtt{W} \cup G.\mathtt{U}) \times (G.\mathtt{W} \cup G.\mathtt{U}))$ is acyclic. Indeed, if there exists $a \in G.\mathtt{W} \cup G.\mathtt{U}$, such that $\langle a, a\rangle \in (rf; po)^+$, then by Lemma 6, we obtain $a@tid(a) < a@tid(a)$, which contradicts the fact that $<$ is irreflexive.

Next, we show that there exists a modification order mo_0 in G'_0. Let $mo = \{\langle a, b\rangle \in (G.\mathtt{W} \cup G.\mathtt{U}) \times (G.\mathtt{W} \cup G.\mathtt{U}) \mid a@tid(a) <$

$b@tid(b)\}$, and $mo_\sigma = mo'_\sigma \cup (A_\sigma \times (G.\mathtt{W} \cup G.\mathtt{U}))$ where mo'_σ is an arbitrary strict total order on A_σ. We take $mo_0 = mo \cup mo_\sigma$, and show that mo_0 is a modification order in G'_0 (and hence, by Proposition 7, G'_0 is SRA-coherent):

- Clearly, mo_0 is a strict total order on $G_0.\mathtt{W} \cup G_0.\mathtt{U}$.
- We prove that $mo_0; (po_0 \cup rf_0)^+$ is irreflexive. Let $\langle b, a\rangle \in (po_0 \cup rf_0)^+$. This entails that $a \in A$. Now, if $a \notin G.\mathtt{W} \cup G.\mathtt{U}$ or $b \notin G.\mathtt{W} \cup G.\mathtt{U}$, then obviously $\langle a, b\rangle \notin mo_0$. Assume otherwise. Then, $\langle b, a\rangle \in (po \cup rf)^+$. By Lemma 6, we have $b@tid(a) < a@tid(a)$. Since H is legal, we have $b@tid(b) \leq b@tid(a)$. It follows that $b@tid(b) < a@tid(a)$, and so $\langle a, b\rangle \notin mo_0$.
- Let $a, b, c \in A_0$, such that $\langle a, c\rangle \in rf_0$, $\langle b, c\rangle \in (po_0 \cup rf_0)^+ \cup mo_0$, and $loc(a) = loc(b)$. We show that $\langle a, b\rangle \notin mo_0$. First, if $b \notin G.\mathtt{W} \cup G.\mathtt{U}$, then, by construction, $\langle a, b\rangle \notin mo_0$. Otherwise, it follows that $\langle b, c\rangle \in (po \cup rf)^+ \cup mo$. Let $x = loc(a)$ and $i = tid(c)$. Consider two cases:
  - $\langle b, c\rangle \in (po \cup rf)^+$. In this case, by Lemma 6, we have $b@i < c@i$. It follows that $b \in H.before(c)$. Hence, $c \in dom(H.last)$, and since $\langle a, c\rangle \in rf_0$, we have $H.last(c) = a$. Hence, $b@tid(b) \leq a@tid(a)$, and so $\langle a, b\rangle \notin mo_0$.
  - $\langle b, c\rangle \in mo$. Then, $b@tid(b) < c@tid(c)$ and $c \in \mathcal{U}$. Since H is legal, $c \in dom(H.last)$ and $b@tid(b) \leq H.last(c)@tid(H.last(c))$. Since $\langle a, c\rangle \in rf_0$, we have $H.last(c) = a$. Hence, $b@tid(b) \leq a@tid(a)$, and so $\langle a, b\rangle \notin mo_0$. □

E.1 Soundness and Completeness Proof

Notation 5. For a message buffer L, $L[k...]$ denotes the set $\{L[k], \ldots, L[n]\}$ of messages.

E.1.1 Soundness Proof

Suppose that $\langle P, [S_\sigma, T_\emptyset]\rangle \rightarrow^* \langle P_{\text{final}}, [S, T]\rangle$ for some machine state $[S, T]$. By contracting internal program steps together with program and machine steps, it easily follows that $\langle P, [S_\sigma, T_\emptyset]\rangle \Rightarrow^* \langle P_{\text{final}}, [S, T]\rangle$, where $\Rightarrow$ is given by:

$$\frac{P_1 \rightarrow^* P_2 \qquad P_2 \xrightarrow{l,i} P_3 \qquad P_3 \rightarrow^* P_4 \qquad [S_1, T_1] \xrightarrow{l,i} [S_2, T_2]}{\langle P_1, [S_1, T_1]\rangle \Rightarrow \langle P_4, [S_2, T_2]\rangle}$$

$$\frac{[S_1, T_1] \rightarrow [S_2, T_2]}{\langle P_1, [S_1, T_1]\rangle \Rightarrow \langle P_1, [S_2, T_2]\rangle}$$

Thus, there exist $\langle P_0, [S_0, T_0]\rangle, \ldots, \langle P_n, [S_n, T_n]\rangle$, such that $\langle P_0, [S_0, T_0]\rangle = \langle P, [S_\sigma, T_\emptyset]\rangle$, $\langle P_n, [S_n, T_n]\rangle = \langle P_{\text{final}}, [S, T]\rangle$, and $\langle P_k, [S_k, T_k]\rangle \Rightarrow \langle P_{k+1}, [S_{k+1}, T_{k+1}]\rangle$ for every $0 \leq k \leq n-1$. Let $li : \{1, \ldots, n\} \rightharpoonup \mathsf{Lab} \times \{1, \ldots, N\}$, such that for every $1 \leq k \leq n$, either $li(k) = \langle l, i\rangle$ and $[S_{k-1}, T_{k-1}] \xrightarrow{l,i} [S_k, T_k]$, or $k \notin dom(li)$ and $[S_{k-1}, T_{k-1}] \rightarrow [S_k, T_k]$. For every $1 \leq k \leq n$ such that $li(k) = \langle l, i\rangle$, let $a_k = \langle k, i, l\rangle$, i.e., a_k is the event with $id(a) = k$, $tid(a) = i$, and $lab(a) = l$. Let $G_0 = G_\emptyset$, and for every $0 \leq k \leq n-1$, let $G_{k+1} = G_k; a_{k+1}$ if $k \in dom(li)$, or $G_{k+1} = G_k$ otherwise. By definition, we have $\langle P_k, G_k\rangle \rightarrow^* \langle P_{k+1}, G_{k+1}\rangle$ for every $0 \leq k \leq n-1$. We obtain that $\langle P, G_\emptyset\rangle \rightarrow^* \langle P_{\text{final}}, G_n\rangle$. It remains to show that $\sigma; G_n$ is SRA-consistent. By Thm. 2, it suffices to provide a $\langle G_n, \sigma\rangle$-legal history.

Note first that our construction ensures that: (1) $\{id(a) \mid a \in G_k\} \subseteq \{1, \ldots, k\}$; and (2) $id(a) \neq id(b)$ for every $a \neq b$ in some G_k. For every $a \in G_n.\mathtt{W} \cup G_n.\mathtt{U}$, let $\#(a)$ denote the number of events $b \in G_n.\mathtt{W} \cup G_n.\mathtt{U}$ such that $loc(b) = loc(a)$ and $id(b) \leq id(a)$. Additionally, for every $x \in \mathsf{Loc}$ and $t \in \mathbb{N}$, let a^t_x denote the event $a \in G_n.\mathtt{W} \cup G_n.\mathtt{U}$ such that $loc(a) = x$ and $\#(a) = t$ (if such an event does not exist, let a^t_x be an arbitrary event that is not in $\sigma; G_n$). We inductively construct a sequence $H_0 = \langle B_0, \leq_0\rangle, \ldots, H_n = \langle B_n, \leq_n\rangle$ of histories, such that for every $0 \leq k \leq n$, H_k is $\langle G_k, \sigma\rangle$-legal (and, in particular, H_n is $\langle G_n, \sigma\rangle$-legal), and valid for stage k, as defined next:

Definition. We say that a history $H = \langle B, \leq\rangle$ is *valid* for stage $0 \leq k \leq n$ if the following hold, where $S_k(i) = \langle M_i, L_i, I_i\rangle$ for every $1 \leq i \leq N$:

[valid order] If $a@tid(a) < b@tid(b)$, $a, b \in \mathcal{W} \cup \mathcal{U}$, and $loc(a) = loc(b)$, then $\#(a) \leq \#(b)$.

[valid count] For every $x \in \mathsf{Loc}$, $T_k(x)$ is the number of actions $a@i \in B$ satisfying $a \in \mathcal{W} \cup \mathcal{U}$, $loc(a) = x$, and $tid(a) = i$.

[valid memory a] If $M_i(x) = v@t$ then either $t = 0$ and $v = \sigma(x)$, or $t > 0$, $a^t_x@i \in B$, and $v = val_w(a^t_x)$.

[valid memory b] If $M_i(x) = v@t$ then $a^{t'}_x@i \notin B$ for every $t' > t$.

[valid messages] If $\lfloor x{:=}v@t\rfloor$ appears in L_i then $a^t_x@i \in B$ and $v = val_w(a)$.

[valid message order] If $\lfloor x_1{:=}v_1@t_1\rfloor$ appears in L_i before $\lfloor x_2{:=}v_2@t_2\rfloor$, then $a^{t_1}_{x_1}@i < a^{t_2}_{x_2}@i$.

[valid cleans] If $a@i \in B$, $a \in \mathcal{W} \cup \mathcal{U}$, and $a@j \notin B$ for some j, then $\lfloor loc(a){:=}val_w(a)@\#(a)\rfloor$ appears in L_i.

[valid indices] For every $i \neq j$, if $I_i(j) \leq |L_j|$ and $L_j[I_i(j)] = \lfloor x{:=}v@t\rfloor$, then $a@i \in B$ for every event $a \in \mathcal{W} \cup \mathcal{U}$ such that $a@j < a^t_x@j$.

[valid overflow] For every $i \neq j$, if $I_i(j) > |L_j|$ then $I_i(j) = |L_j| + 1$, and $a@i \in B$ for every event $a \in \mathcal{W} \cup \mathcal{U}$ such that $a@j \in B$.

For the base case, let H_0 to be the empty history. Clearly, H_0 is $\langle G_\emptyset, \sigma\rangle$-legal and valid for stage 0. Suppose that $H_k = \langle B_k, \leq_k\rangle$ is defined. Below, we show how to construct H_{k+1}. Using the fact that H_k is $\langle G_k, \sigma\rangle$-legal and valid for stage k, it is straightforward to show that the constructed H_{k+1} is $\langle G_{k+1}, \sigma\rangle$-legal and valid for stage $k+1$. We prove here only some of the (more interesting) properties that are needed to be shown. Let $S_k(i) = \langle M_i, L_i, I_i\rangle$ for every $1 \leq i \leq N$, and consider two cases:

- $k \in dom(li)$. Let $li(k) = \langle l, i\rangle$. We take H_{k+1} to be $H_k + a_{k+1}@i$ (i.e., the extension of H_k with the maximum element $a_{k+1}@i$). Note that in this case we have $[S_k, T_k] \xrightarrow{l,i} [S_{k+1}, T_{k+1}]$ and $G_{k+1} = G_k; a_{k+1}$. To see that H_{k+1} is $\langle G_{k+1}, \sigma\rangle$-legal and valid for stage $k+1$, consider the three possible machine steps:

  - $[S_k, T_k] \xrightarrow{l,i} [S_{k+1}, T_{k+1}]$ is obtained by a (WRITE) step. In this case, there exist $x \in \mathsf{Loc}$, $v \in \mathsf{Val}$, and $t \in \mathbb{N}$, such that $l = \langle \mathtt{W}, x, v\rangle$, $T_k(x) = t$, $S_{k+1} = S_k[i \mapsto \langle M', L', I_i\rangle]$, and $T_{k+1} = T_k[x \mapsto t+1]$, where $M' = M_i[x \mapsto v@t+1]$ and $L' = L_i; \lfloor x{:=}v@t+1\rfloor$. To see that [valid memory a] is preserved, note that $a_{k+1} = a^{t+1}_x$ (using [valid count]). To see that [valid message order] is preserved, note that [valid messages] entails that $a^{t'}_y@i \in B_k$ whenever $\lfloor y{:=}v'@t'\rfloor \in L_i$.

  - $[S_k, T_k] \xrightarrow{l,i} [S_{k+1}, T_{k+1}]$ is obtained by a (READ) step. In this case, $S_{k+1} = S_k$, $T_{k+1} = T_k$, and there exist $x \in \mathsf{Loc}$, $v \in \mathsf{Val}$, and $t \in \mathbb{N}$, such that $l = \langle \mathtt{R}, x, v\rangle$, and $M_i(x) = v@t$.

    We prove that $val_r(a) = val_w(H_{k+1}.last(a))$ for every $a \in dom(H_{k+1}.last)$. First, for $a \neq a_{k+1}$, this follows directly from the fact that H_k is $\langle G_k, \sigma\rangle$-legal. It remains

to prove that $v = val_w(H_{k+1}.last(a_{k+1}))$ if $a_{k+1} \in dom(H_{k+1}.last)$. Let $C = H_{k+1}.before(a_{k+1})$. Note the for every $c \in C$, we have $c@i \in B_k$, $loc(c) = x$, and $c \in \mathcal{W} \cup \mathcal{U}$, and by [valid memory b] (since $c = a_x^{\#(c)}$), we have $\#(c) \leq t$. Assuming that $a_{k+1} \in dom(H_{k+1}.last)$, C is not empty, and so we have $t > 0$. By [valid memory a], we have $a_x^t@i \in B_k$ and $v = val_w(a_x^t)$. We prove that $H_{k+1}.last(a_{k+1}) = a_x^t$. It then follows that $v = val_w(H_{k+1}.last(a_{k+1}))$. Let $j = tid(a_x^t)$. Since H_k is $\langle G_k, \sigma\rangle$-legal, we have $a_x^t@j \leq_k a_x^t@i$, and so $a_x^t@j \in B_{k+1}$. Since $a_{k+1}@i$ is the last action in H_{k+1}, we also have $a_x^t@i <_{k+1} a_{k+1}@i$, and so $a_x^t@j \in C$. In addition, for every $c@tid(c) \in C$, $\#(c) \leq t = \#(a_x^t)$, and by [valid order], we obtain that $c@tid(c) \leq_{k+1} a_x^t@j$. Hence, $a_x^t@j$ is $\leq_{k+1}$-maximal in C.

- $[S_k, T_k] \xrightarrow{l,i} [S_{k+1}, T_{k+1}]$ is obtained by an (UPDATE) step. In this case, there exist $x \in \mathsf{Loc}$, $v_r, v_w \in \mathsf{Val}$, and $t \in \mathbb{N}$, such that $l = \langle \mathtt{U}, x, v_r, v_w\rangle$, $T_k(x) = t$, $M_i(x) = v_r@t$, $S_{k+1} = S_k[i \mapsto \langle M', L', I_i\rangle]$, and $T_{k+1} = T_k[x \mapsto t+1]$, where $M' = M_i[x \mapsto v_w@t+1]$ and $L' = L_i; \lfloor x{:=}v_w@t+1\rfloor$.

  As in the previous case, one proves that we have $val_r(a) = val_w(H_{k+1}.last(a))$ for every $a \in dom(H_{k+1}.last)$, and, in particular, $H_{k+1}.last(a_{k+1}) = a_x^t$ if $a_{k+1} \in dom(H_{k+1}.last)$. We further show that $a \in \mathcal{W} \cup \mathcal{U}$ and $b \in \mathcal{U}$, if $loc(a) = loc(b)$ and $a@tid(a) <_{k+1} b@tid(b)$, then $b \in dom(H_{k+1}.last)$, and $a@tid(a) \leq_{k+1} H_{k+1}.last(b)@tid(H_{k+1}.last(b))$. If $b \neq a_{k+1}$, this follows from the fact that H_k is $\langle G_k, \sigma\rangle$-legal. Now, let $a \in \mathcal{W} \cup \mathcal{U}$ such that $loc(a) = x$ and $a@tid(a) <_{k+1} a_{k+1}@tid(a_{k+1})$. Since $T_k(x) = t$ and H_k is $\langle G_k, \sigma\rangle$-legal, [valid count] entails that $\#(a) \leq t$. By [valid order], it follows that $H_{k+1}.before(a_{k+1})$ is non-empty, and so $a_{k+1} \in dom(H_{k+1}.last)$. Thus, $H_{k+1}.last(a_{k+1}) = a_x^t$. By [valid order], we also obtain that $a@tid(a) \leq_k a_x^t@tid(a_x^t)$.

- $k \notin dom(li)$. In this case, we have $G_{k+1} = G_k$ and $[S_k, T_k] \to [S_{k+1}, T_{k+1}]$. Since the machine step is not (WRITE) or (UPDATE), we have $T_{k+1} = T_k$. To define H_{k+1}, consider the possible machine steps:

  - $[S_k, T_k] \to [S_{k+1}, T_{k+1}]$ is obtained by a (PROCESS) step. In this case, there exist $i \neq j$, $x \in \mathsf{Loc}$, $v \in \mathsf{Val}$, and $t_i < t_j$, such that $I_i(j) \leq |L_j|$, $L_j[I_i(j)] = \lfloor x{:=}v@t_j\rfloor$, $M_i(x) = -@t_i$, and $S_{k+1} = S_k[i \mapsto \langle M', L', I'\rangle]$, where $M' = M_i[x \mapsto \langle v, t_j\rangle]$, $L' = L_i; \lfloor x{:=}v@t_j\rfloor$, and $I' = I_i[j \mapsto I_i(j)+1]$. By [valid messages], we have $a_x^{t_j}@j \in B_k$ and $v = val_w(a_x^{t_j})$. We take H_{k+1} is to be $H_k + a_x^{t_j}@i$.

    We prove that for every $a \in \mathcal{W} \cup \mathcal{U}$ and $b@i' \in B_{k+1}$, if $a@tid(b) <_{k+1} b@tid(b)$, then $a@i' \in B_{k+1}$ and $a@i' <_{k+1} b@i'$. Note first that $a_x^{t_j}@i \notin B_k$ (by [valid memory b], since $M_i(x) = -@t_i$ and $t_i < t_j$). Since H_k is $\langle G_k, \sigma\rangle$-legal and $a@j \in B_k$, we have $a_x^{t_j}@tid(a_x^{t_j}) \in B_k$. Hence, we also have $i \neq tid(a_x^{t_j})$. Therefore, since $G_{k+1} = G_k$ and H_k is $\langle G_k, \sigma\rangle$-legal, it suffices to prove that for every $a \in \mathcal{W} \cup \mathcal{U}$, if $a@tid(a_x^{t_j}) <_k a_x^{t_j}@tid(a_x^{t_j})$, then $a@i \in B_k$. Let $a \in \mathcal{W} \cup \mathcal{U}$, such that $a@tid(a_x^{t_j}) <_k a_x^{t_j}@tid(a_x^{t_j})$. Since H_k is $\langle G_k, \sigma\rangle$-legal and $a_x^{t_j}@j \in B_k$, it follows that $a@j \in B_k$ and $a@j <_k a_x^{t_j}@j$. Since $I_i(j) \leq |L_j|$ and $L_j[I_i(j)] = \lfloor x{:=}v@t_j\rfloor$, [valid indices] entails that $a@i \in B_k$.

  - $[S_k, T_k] \to [S_{k+1}, T_{k+1}]$ is obtained by a (SKIP) step. In this case, there exist $i \neq j$, $x \in \mathsf{Loc}$, $v \in \mathsf{Val}$, and $t_j \leq t_i$, such that $I_i(j) \leq |L_j|$, $L_j[I_i(j)] = \lfloor x{:=}v@t_j\rfloor$, $M_i(x) = -@t_i$, and $S_{k+1} = S_k[i \mapsto \langle M_i, L_i, I'\rangle]$, where $I' = I_i[j \mapsto I_i(j)+1]$. By [valid messages], we have $a_x^{t_j}@j \in B_k$ and $v = val_w(a_x^{t_j})$. We take H_{k+1} is to be $H_k + a_x^{t_j}@i$ if $a_x^{t_j}@i \notin B_k$, or $H_{k+1} = H_k$ otherwise.

    We prove [valid indices] for H_{k+1}. Since it held for H_k, it suffices to prove that if $I'(j) \leq |L_j|$ and $L_j[I'(j)] = \lfloor y{:=}v'@t'\rfloor$, then $a@i \in B_{k+1}$ for every event $a \in \mathcal{W} \cup \mathcal{U}$ such that $a@j <_{k+1} a_y^{t'}@j$. Suppose that $I'(j) \leq |L_j|$ and $L_j[I'(j)] = \lfloor y{:=}v'@t'\rfloor$. Let $a \in \mathcal{W} \cup \mathcal{U}$ be some event such that $a@j <_{k+1} a_y^{t'}@j$. Suppose for contradiction that $a@i \notin B_{k+1}$. By [valid cleans], $m = \lfloor loc(a){:=}val_w(a)@\#(a)\rfloor$ appears in L_j. By [valid message order], m appears in L_j before $\lfloor y{:=}v'@t'\rfloor$. Since (by construction) $a_x^{t_j}@i \in B_{k+1}$, it must be the case that m appears in L_j before $\lfloor x{:=}v@t_j\rfloor$. By [valid message order], $a@j <_{k+1} a_x^{t_j}@j$. This contradicts [valid indices] (for stage k).

  - $[S_k, T_k] \to [S_{k+1}, T_{k+1}]$ is obtained by a (CLEAN) step. In this case, there exists $1 \leq i \leq N$, such that $L_i = m; L$ for some message m and message list L, $I_j(i) > 1$ for every $j \neq i$, and $S_{k+1} = \lambda j.\ \langle M_j, L'_j, I'_j\rangle$, where $L'_j = L_j$ for every $j \neq i$, $L'_i = L$, $I'_j = I_j[i \mapsto I_j(i)-1]$ for every $j \neq i$, and $I'_i = I_i$. We take $H_{k+1} = H_k$.

    To see that [valid indices] holds for H_{k+1}, note that we have $L'_{j'}[I'_{j'}(j')] = L_{j'}[I_{i'}(j')]$ for every $j' \neq i'$. We prove [valid cleans] for H_{k+1}. Let $a@i' \in B_{k+1}$ such that $a \in \mathcal{W} \cup \mathcal{U}$ and $a@j \notin B_{k+1}$ for some j. Let $m' = \lfloor loc(a){:=}val_w(a)@\#(a)\rfloor$. Since $B_{k+1} = B_k$, by [valid cleans] for H_k, m' appears in $L_{i'}$. Clearly, if $i' \neq i$, then m' appears in $L_{i'}$, and we are done. Otherwise, it suffices to show that $m' \neq m$. Suppose otherwise. Since $a@i \in B_k$ but $a@j \notin B_k$, [valid overflow] implies that $I_j(i) \leq |L_j|$. Let $L_i[I_j(i)] = \lfloor x{:=}v@t\rfloor$. By [valid messages], $a_x^t@i \in B_k$. Since $I_j(i) > 1$, m appears before $L_i[I_j(i)]$ in L_i. By [valid message order], we obtain that $a_{loc(a)}^{\#(a)}@i < a_x^t@i$. Since $a = a_{loc(a)}^{\#(a)}$, by [valid indices], we obtain that $a@j \in B_k$, which contradicts our assumption. □

E.1.2 Completeness Proof

Assume that P may terminate under SRA. Then, $\langle P, G_\emptyset\rangle \to^* \langle P_{\text{final}}, G\rangle$ for some SRA-consistent execution G. By contracting internal program steps together with consecutive program and execution steps, it easily follows that $\langle P, G_\emptyset\rangle \Rightarrow^* \langle P_{\text{final}}, G\rangle$, where $\Rightarrow$ is given by:

$$\frac{P_1 \to^* P_2 \qquad P_2 \xrightarrow{l,i} P_3 \qquad P_3 \to^* P_4 \qquad G_1 \xrightarrow{l,i} G_2}{\langle P_1, G_1\rangle \Rightarrow \langle P_4, G_2\rangle}$$

Now, Thm. 2 guarantees that there exists a $\langle G, \sigma\rangle$-legal history $H = \langle B, \leq\rangle$. Let $a_1@i_1, \ldots, a_n@i_n$ be an enumeration of B according to $\leq$, and let $l_k = lab(a_k)$ for every $1 \leq k \leq n$. Let $G_0 = G_\emptyset$, and for every $0 \leq k \leq n-1$, let $G_{k+1} = G_k; a_{k+1}$ if $tid(a_{k+1}) = i_{k+1}$, and $G_{k+1} = G_k$ otherwise.

Claim. There are programs $P = P_0, P_1, \ldots, P_n = P_{\text{final}}$, such that for every $0 \leq k \leq n-1$, we have $P_k \xrightarrow{l_{k+1}, i_{k+1}} P_{k+1}$ if $tid(a_{k+1}) = i_{k+1}$, and $P_k = P_{k+1}$ otherwise, where $\xrightarrow{l,i}$ is given

by:

$$\frac{P(i) \rightarrow^* c \qquad c \xrightarrow{l} c' \qquad c' \rightarrow^* c''}{P \xRightarrow{l,i} P[i \mapsto c'']}$$

Proof. Intuitively speaking, we reorder the steps in the derivation of $\langle P, G_\emptyset \rangle \Rightarrow \langle P_{\text{final}}, G \rangle$, so they follow the order of the events in H. Since steps of different threads do not interfere, we can freely change the order among threads, preserving the order inside each thread. Since the history follows this order for each thread, we can follow $\leq$.

Formally, we inductively construct a sequence $P_0, \ldots, P_n$ of programs, such that $P = P_0$, $P_k \xrightarrow{l_{k+1}, i_{k+1}} P_{k+1}$ if $tid(a_{k+1}) = i_{k+1}$ for every $0 \leq k \leq n-1$, $P_k = P_{k+1}$ if $tid(a_{k+1}) \neq i_{k+1}$ for every $0 \leq k \leq n-1$, and $\langle P_k, G_k \rangle \Rightarrow^* \langle P_{\text{final}}, G \rangle$ for every $0 \leq k \leq n$. Finally, we would obtain that $\langle P_n, G_n \rangle \Rightarrow^* \langle P_{\text{final}}, G \rangle$, and since $G_n = G$, it must be the case that $P_n = P_{\text{final}}$.

For the base case, we take $P_0 = P$. Now, suppose that P_k is defined. First, if $tid(a_{k+1}) \neq i_{k+1}$, we take $P_{k+1} = P_k$. Assume otherwise. Since $\langle P_k, G_k \rangle \Rightarrow^* \langle P_{\text{final}}, G \rangle$, there are commands $c_1, \ldots, c_m$ and events $a'_1, \ldots, a'_m$ such that $\langle P'_d, G'_d \rangle \Rightarrow \langle P'_{d+1}, G'_{d+1} \rangle$ for every $0 \leq d \leq m-1$, where $P'_0 = P_k$, $G'_0 = G_k$, $P'_m = P_{\text{final}}$, $G'_m = G$, and $\langle P'_{d+1}, G'_{d+1} \rangle = \langle P'_d[tid(a'_{d+1}) \mapsto c_{d+1}], G'_d; a'_{d+1} \rangle$ for every $0 \leq d \leq m-1$. Since H is $\langle G, \sigma \rangle$-legal, we have $a_{k+1} \notin G_k$. Thus, since $a_{k+1} \in G$, we have $a_{k+1} \in \{a'_1, \ldots, a'_m\}$. Let d_0 be the minimal index such that $a_{k+1} = a'_{d_0}$. We take P_{k+1} to be $P_k[i_{k+1} \mapsto c_{d_0}]$. To see that $P_k \xrightarrow{l_{k+1}, i_{k+1}} P_{k+1}$, it suffices to note that $tid(a'_d) \neq i_{k+1}$ for every $1 \leq d < d_0$ (and so $P'_{d_0}(i_{k+1}) = P_k(i_{k+1})$). To see that $\langle P_{k+1}, G_{k+1} \rangle \Rightarrow^* \langle P_{\text{final}}, G \rangle$, it suffices to show that $\langle P_{k+1}, G_{k+1} \rangle \Rightarrow^* \langle P'_{d_0+1}, G'_{d_0+1} \rangle$. Let $P''_0 = P_{k+1}$, and $P''_{d+1} = P''_d[tid(a'_{d+1}) \mapsto c_{d+1}]$ for every $0 \leq d \leq d_0 - 1$. Let $G''_0 = G_{k+1}$ and $G''_{d+1} = G''_d; a'_{d+1}$ for every $0 \leq d \leq d_0 - 1$. Then, $P''_{d_0} = P'_{d_0+1}$ and $G''_{d_0} = G'_{d_0+1}$. In addition, it is straightforward to show that $\langle P''_d, G''_d \rangle \Rightarrow \langle P''_{d+1}, G''_{d+1} \rangle$ for every $0 \leq d \leq d_0 - 1$. □

For every $0 \leq k \leq n$, let $A_k = \{a_1, \ldots, a_k\}$, $A'_k = A_k \cap (\mathcal{W} \cup \mathcal{U})$, and $B_k = \{a_1 @ i_1, \ldots, a_k @ i_k\}$. In addition, for every $a \in A'_n$, let $\#(a)$ denote the number of events $b \in A'_n$ such that $loc(b) = loc(a)$ and $b @ tid(b) \leq a @ tid(a)$. Note that for every $a, b \in A'_n$ such that $loc(a) = loc(b)$, if $\#(a) \leq \#(b)$, then $a @ tid(a) \leq b @ tid(b)$ (and hence, $\#(a) = \#(b)$ implies $a = b$).

Let $T_k = \lambda x \in \mathsf{Loc}.\ |\{a \in A'_k \mid loc(a) = x\}|$. We inductively construct a sequence $S_0, \ldots, S_n$ of functions assigning a processor state to every $i \in \{1, \ldots, N\}$, such that $S_0 = S_\sigma$, $\langle P_k, [S_k, T_k] \rangle \rightarrow^+ \langle P_{k+1}, [S_{k+1}, T_{k+1}] \rangle$ for every $0 \leq k \leq n-1$, and S_k is valid for stage k for every $0 \leq k \leq n$, as defined next:

Definition. We say that a function S assigning a processor state to every $i \in \{1, \ldots, N\}$ is *valid* for stage $0 \leq k \leq n$ if the following hold, where $S(i) = \langle M_i, L_i, I_i \rangle$ for every $1 \leq i \leq N$:

[valid memory] For every $1 \leq i \leq N$ and $x \in \mathsf{Loc}$, if $M_i(x) = v @ t$, then either $\lfloor x := v @ t \rfloor$ is the last message of the form $\lfloor x := - @ - \rfloor$ in L_i, or $v @ t = \sigma(x) @ 0$ and no such message exists in L_i.

[valid lists] For every $a \in A'_k$, $\lfloor loc(a) := val_w(a) @ \#(a) \rfloor$ appears in $L_{tid(a)}$.

[valid indices] For every $1 \leq i \leq N$ and $a \in A'_k$, if $a @ i \notin B_k$, then $\lfloor loc(a) := val_w(a) @ \#(a) \rfloor$ appears in $L_j[I_i(j)...]$ where $j = tid(a)$.

[valid overflow] For every $1 \leq i, j \leq N$, $I_i(j) \leq |L_j| + 1$.

[valid skips] For every $a @ i \in B_k$, if $\lfloor loc(a) := val_w(a) @ \#(a) \rfloor$ does not appear in L_i, then $\lfloor loc(a) := v @ t \rfloor \in L_i$ for some $v \in \mathsf{Val}$ and $t > \#(a)$.

[valid messages] For every $1 \leq i \leq N$, if $\lfloor x := v @ t \rfloor \in L_i$, then $a @ i \in B_k$ for some event $a \in A'_n$ such that $loc(a) = x$, $val_w(a) = v$, and $\#(a) = t$.

[valid message order] For every $1 \leq i \leq N$, and $a, b \in A'_n$, if $\lfloor loc(a) := - @ \#(a) \rfloor$ appears in L_i before $\lfloor loc(b) := - @ \#(b) \rfloor$, then $a @ i < b @ i$.

[valid message times] For every $1 \leq i \leq N$, $x \in \mathsf{Loc}$, and $t_1, t_2 \in \mathbb{N}$, if $\lfloor x := - @ t_1 \rfloor$ appears in L_i before $\lfloor x := - @ t_2 \rfloor$, then $t_1 < t_2$.

Finally, since $T_0 = T_\emptyset$, it would follow that $\langle P, [S_\sigma, T_\emptyset] \rangle \Rightarrow^* \langle P_{\text{final}}, [S_n, T_n] \rangle$.

For the base case, let $S_0 = S_\sigma$. It is trivial to check that S_0 is valid for stage 0. For the inductive step, we show how to construct S_{k+1}, assuming that S_k is defined. It is straightforward to show that the constructed function is valid for stage $k+1$. We prove here only some of the (more interesting) properties. Let $a = a_{k+1}$, $l = l_{k+1}$, and $i = i_{k+1}$. Consider the following two cases:

- $tid(a) = i$. In this case we have $P_k \xRightarrow{l,i} P_{k+1}$. Thus, there exist commands c and c', such that $P_k(i) \rightarrow^* c$ and $c \xrightarrow{l} c'$ and $c' \rightarrow^* P_{k+1}(i)$. We construct S_{k+1} such that $[S_k, T_k] \xrightarrow{l,i} [S_{k+1}, T_{k+1}]$. Then, $\langle P_k, [S_k, T_k] \rangle \rightarrow^+ \langle P_{k+1}, [S_{k+1}, T_{k+1}] \rangle$ follows by (possibly) internal program steps deriving $\langle P_k[i \mapsto c], [S_k, T_k] \rangle$ from $\langle P_k, [S_k, T_k] \rangle$, followed by a combined program and machine step deriving $\langle P_k[i \mapsto c'], [S_{k+1}, T_{k+1}] \rangle$ from $\langle P_k[i \mapsto c], [S_k, T_k] \rangle$, and (possibly) again followed by internal program steps deriving $\langle P_{k+1}, [S_{k+1}, T_{k+1}] \rangle$ from $\langle P_k[i \mapsto c'], [S_{k+1}, T_{k+1}] \rangle$.

  Let $S_k(i) = \langle M, L, I \rangle$, and consider the following three possible cases:

  - $typ(a) = \mathtt{W}$. Let $x = loc(a)$, $v = val_w(a)$, and $t = T_k(x)$. In this case, we take S_{k+1} to be $S_k[i \mapsto \langle M', L', I \rangle]$, where $M' = M[x \mapsto v @ t+1]$, and $L' = L; \lfloor x := v @ t+1 \rfloor$. Then, $[S_k, T_k] \xrightarrow{l,i} [S_{k+1}, T_{k+1}]$ is obtained using a (WRITE) step.

    We prove [valid messages] for S_{k+1}. Using the fact that S_k is valid for stage k, it suffices to show that there exists an event $b \in A'_n$ satisfying $loc(b) = x$, $val_w(b) = v$, $\#(b) = t+1$, and $b @ i \in B_{k+1}$. Indeed, if we take $b = a$, we have $loc(a) = x$, $val_w(a) = v$, and $a @ i \in B_{k+1}$. Since $a @ i$ is the last action in B_{k+1},$\{b \in A'_k \mid loc(b) = x\}$ is exactly the set of all events $b \in A'_n$ such that $loc(b) = x$ and $b @ tid(b) < a @ i$. It follows that $\#(a) = t+1$.

  - $typ(a) = \mathtt{R}$. Let $x = loc(a)$ and $v = val_r(a)$. In this case, we take S_{k+1} to be S_k. Since H is $\langle G, \sigma \rangle$-legal, $v = val_w(H.last(a))$ if $a \in dom(H.last)$, and otherwise $v = \sigma(x)$. We show that $M(x) = v @ t_a$, where $t_a = \#(H.last(a))$ if $a \in dom(H.last)$, and otherwise $t_a = 0$. It follows that $[S_k, T_k] \xrightarrow{l,i} [S_{k+1}, T_{k+1}]$ is obtained using a (READ) step.

    Note that no message of the form $\lfloor x := - @ t \rfloor$ with $t > t_a$ appears in L. Otherwise, by [valid messages], $c @ i \in B_k$ for some $c \in A'_n$ such that $loc(c) = x$ and $\#(c) = t$. It follows that $c @ i < a @ i$, and so $c \in H.bf(a)$. Hence, $a \in dom(H.last)$, and we have $H.last(a) @ tid(H.last(a)) < c @ tid(c)$. This is not possible according to the definition of $H.last(a) @ tid(H.last(a))$.

Now, if $a \notin dom(H.last)$, then no message of the form $\lfloor x{:=}-@-\rfloor$ appears in L, and by [valid memory], $M(x) = \sigma(x)@0$, and we are done. Otherwise, let $b = H.last(a)$. Hence, we have $b@i < a@i$, and thus $b@i \in B_k$. By [valid skips], it follows that $m = \lfloor x{:=}v@\#(b)\rfloor$ appears in L. Moreover, m is the last message of the form $\lfloor x{:=}-@-\rfloor$ in L, and so, by [valid memory], $M(x) = v@\#(b)$. Indeed, a message $\lfloor x{:=}v'@t\rfloor$ appears in L after m, then $\#(b) < t$ (by [valid message times]), which, as shown above, is impossible.

Since $A'_{k+1} = A'_k$ and $S_{k+1} = S_k$, the fact that S_{k+1} is valid for stage $k+1$ trivially follows from the fact that S_k is valid for stage k.

- $typ(a) = \mathtt{U}$. Let $x = loc(a)$, $v_r = val_r(a)$, $v_w = val_w(a)$, and $t = T_k(x)$. In this case, we take S_{k+1} to be $S_k[i \mapsto \langle M', L', I\rangle]$, where $M' = M[x \mapsto v_w@t+1]$ and $L' = L; \lfloor x{:=}v_w@t+1\rfloor$. As in the previous case, we have $M(x) = val_w(H.last(a))@\#(H.last(a)) = v_r@\#(H.last(a))$ if $a \in dom(H.last)$, and $M(x) = \sigma(x)@0 = v_r@0$ otherwise.

  First, if $a \notin dom(H.last)$, then since H is $\langle G, \sigma\rangle$-legal, there cannot exist and $c \in A'_n$ such that $loc(c) = x$ and $c@tid(c) \in B_k$. Hence, in this case, we have $t = 0$. Otherwise, if $a \in dom(H.last)$, since H is $\langle G, \sigma\rangle$-legal, we have $c@tid(c) \leq H.last(a)@tid(H.last(a))$ for every $c \in A'_n$ such that $loc(c) = x$ and $c@tid(c) \in B_k$. It easily follows that $t = \#(H.last(a))$ in this case.

  Therefore, $[S_k, T_k] \xrightarrow{l,i} [S_{k+1}, T_{k+1}]$ can be obtained using an (UPDATE) step.

- $tid(a) \neq i$. In this case we have $P_k = P_{k+1}$ and $T_k = T_{k+1}$. We construct S_{k+1} such that $[S_k, T_k] \rightarrow^+ [S_{k+1}, T_{k+1}]$. Then, $\langle P_k, [S_k, T_k]\rangle \rightarrow^+ \langle P_{k+1}, [S_{k+1}, T_{k+1}]\rangle$ follows by repeatedly applying a machine step.

  Let $S_k(i) = \langle M_i, L_i, I_i\rangle$, and let $j = tid(a)$ and $S_k(j) = \langle M_j, L_j, I_j\rangle$. Note that since H is $\langle G, \sigma\rangle$-legal, we have $a \in \mathcal{W} \cup \mathcal{U}$ and $a@j < a@i$. Let $x = loc(a)$ and $v = val_w(a)$. By [valid indices], $L_j[I_i(j)...]$ has a prefix of the form

  $$\lfloor x_1{:=}v_1@t'_1\rfloor \ldots \lfloor x_m{:=}v_m@t'_m\rfloor \lfloor x{:=}v@\#(a)\rfloor$$

  (where $m \geq 0$). For every $1 \leq d \leq m$, let $M_i(x_d) = v_{x_d}@t_{x_d}$. Then, for every $1 \leq d \leq m$, by [valid memory], $\lfloor x_d{:=}v_{x_d}@t_{x_d}\rfloor$ is the last message of the form $\lfloor x_d{:=}-@-\rfloor$ in L_i. For every $1 \leq d \leq m$, since $\lfloor x_d{:=}v_d@t'_d\rfloor$ appears in L_j, by [valid messages], $b_d@j \in B_k$ for some $b_d \in A'_n$, such that $loc(b_d) = x_d$ and $\#(b_d) = t'_d$, and since $\lfloor x_d{:=}v_d@t'_d\rfloor$ appears in L_j before $\lfloor x{:=}v@\#(a)\rfloor$, by [valid message order], we have $b_d@j < a@j$. Since H is $\langle G, \sigma\rangle$-legal, it follows that for every $1 \leq d \leq m$ we have $b_d@i < a@i$, and so $b_d@i \in B_k$.

  We claim that $t'_d \leq t_{x_d}$ for every $1 \leq d \leq m$. Let $1 \leq d \leq m$. Consider two cases:

  1. $\lfloor x_d{:=}v_d@t'_d\rfloor \in L_i$. Then, it does not appear after the message $\lfloor x_d{:=}v_{x_d}@t_{x_d}\rfloor$. By [valid message times], $t'_d \leq t_{x_d}$.
  2. $\lfloor x_d{:=}v_d@t'_d\rfloor \notin L_i$. By [valid skips], $\lfloor x_d{:=}v'@t\rfloor$ appears in L_i for some $v' \in \mathsf{Val}$ and $t > t'_d$. Then, $\lfloor x_d{:=}v'@t\rfloor$ does not appear in L_i after $\lfloor x_d{:=}v_{x_d}@t_{x_d}\rfloor$. By [valid message times], $t \leq t_{x_d}$, and so $t'_d < t_{x_d}$.

  Now, let $m' = I_i(j) + m$ and $I'_i = I_i[j \mapsto m']$. By applying m (SKIP) steps, we obtain $[S_k, T_k] \rightarrow^* [S, T_k]$, where $S = S_k[i \mapsto \langle M_i, L_i, I'_i\rangle]$. We define S_{k+1}, such that $[S, T_k] \rightarrow [S_{k+1}, T_{k+1}]$. Note that we have $L_j[m'] = \lfloor x{:=}v@\#(a)\rfloor$. Let $M'_i = M_i[x \mapsto v@\#(a)]$, $L'_i = L_i; \lfloor x{:=}v@\#(a)\rfloor$, and $I''_i = I_i[j \mapsto m'+1]$. Let $t \in \mathbb{N}$ such that $M_i(x) = -@t$. Consider two cases:

  - $t < \#(a)$. We take $S_{k+1} = S[i \mapsto \langle M'_i, L'_i, I''_i\rangle]$. We have $[S, T_k] \rightarrow [S_{k+1}, T_{k+1}]$ using a (PROCESS) step.
  - $tm(a) \leq t$. We take $S_{k+1} = S[i \mapsto \langle M_i, L_i, I''_i\rangle]$. We have $[S, T_k] \rightarrow [S_{k+1}, T_{k+1}]$ using another (SKIP) step.

  We prove [valid indices] for S_{k+1}. Recall that $A'_{k+1} = A'_k$, $S_{k+1}(i') = S_k(i')$ for every $i' \neq i$, and $I''_i(i') = I_i(i')$ for every $i' \neq j$. Hence, it suffices to show that for every $b \in A'_k$ with $tid(b) = j$, if $b@i \notin B_{k+1}$, then $\lfloor loc(b){:=}val_w(b)@\#(b)\rfloor \in L_j[I_i(j)...]$. Let $b \in A'_k$ with $tid(b) = j$, such that $b@i \notin B_{k+1}$. Let $x' = loc(b)$ and $v' = val_w(b)$. By [valid indices] (for stage k), we have $\lfloor x'{:=}v'@\#(b)\rfloor \in L_j[I_i(j)...]$. First, note that $b \neq a$ (since $a@i \in B_{k+1}$), and hence, $\lfloor x'{:=}v'@\#(b)\rfloor \neq \lfloor x{:=}v@\#(a)\rfloor$. It remains to show that $\lfloor x'{:=}v'@\#(b)\rfloor \neq \lfloor x_d{:=}v_d@t'_d\rfloor$ for every $1 \leq d \leq m$. Suppose for contradiction that $\lfloor x'{:=}v'@\#(b)\rfloor = \lfloor x_d{:=}v_d@t'_d\rfloor$ for some $1 \leq d \leq m$. Then, $x_d = x'$ and $\#(b_d) = t'_d = \#(b)$, and it follows that $b_d = b$. This contradicts the fact that $b_d@i \in B_k$. □

F. Proofs for §5 (Fence Commands)

Proof (of Prop. 8). Let $G = \langle A, po, rf\rangle$ be an RA-coherent complete well-formed execution, and let A_f be the set of all events $a \in A$ with $loc(a) = f$. Since G is well-formed, A_f consists only of fence events, except for one event a_0 with $lab(a_0) = \langle \mathtt{W}, f, 0\rangle$ for which we have $\langle a_0, a\rangle \in po$ for every $a \in A_f \setminus \{a_0\}$. Let mo_f be an f-modification order in G. Then, mo_f is a total strict order on A_f. Let $a_0, a_1, \ldots, a_n$ be an enumeration of all events in A_f according to their order in mo_f (note that a_0 must appear first, since $po; mo_f$ is irreflexive). We prove that $\langle a_i, a_{i+1}\rangle \in rf$ for every $0 \leq i \leq n-1$. It follows that rf^+ is total on fence events. Let $0 \leq i \leq n-1$. Since G is complete, we have $\langle b, a_{i+1}\rangle \in rf$ for some event $b \in A$. Then, $loc(b) = f$, and so $b \in A_f$. Let $0 \leq j \leq n$, such that $b = a_j$. Since rf is acyclic, $j \neq i+1$. Since $rf; mo_f$ is irreflexive, $j \not> i+1$. Similarly, since $\langle b, a_{i+1}\rangle \in rf$ and $\langle a_i, a_{i+1}\rangle \in mo_f$ implies $\langle b, a_i\rangle \notin mo_f$, we also have $j \not< i$. Hence, $j = i$, and so $\langle a_i, a_{i+1}\rangle \in rf$. □

Proof sketch (of Thm. 3). For every $x \in \mathsf{Loc}$, let mo_x be an x-modification order in G. Let $K = \bigcup_x (mo_x \cup fr_x \cup (mo_x; rf) \cup (fr_x; rf))$ (where $fr_x = (rf^{-1}; mo_x) \setminus Id_A$ for every $x \in \mathsf{Loc}$). Using the fact that mo_x is an x-modification order in G, it is straightforward to prove that K is transitive and $(K; rf) \cup (rf; K) \subseteq K$. Since $\bigcup_x mo_x \cup \bigcup_x fr_x \subseteq K$, it suffices to show that $po \cup rf \cup K$ is acyclic.

Consider a cycle in $po \cup rf \cup K$ with a minimal number of $K \setminus (po \cup rf)^+$ edges and (among those with the same number of such edges) with a minimal total length. Such a minimal cycle must have at least two $K \setminus (po \cup rf)^+$ edges because cycles with at most one K edge are ruled out since mo_x is an x-modification order in G (see Prop. 2).

Consider, therefore, a cycle with two or more $K \setminus (po \cup rf)^+$ edges, i.e., $\langle a, b\rangle \in K \setminus (po \cup rf)^+$ and $\langle b, c\rangle \in (po \cup rf)^+$ and $\langle c, d\rangle \in K \setminus (po \cup rf)^+$ and $\langle d, e\rangle \in (po \cup rf)^+$ and either $e = a$ or there exists g such that $\langle e, g\rangle \in K \setminus (po \cup rf)^+$ and $\langle g, a\rangle \in (po \cup rf \cup K)^*$.

The events a, b, c, d, and e are all G-racy (a races with b, c with d and so on). We observe that the first edge in the path from b and c must be a po edge (i.e., $\langle b, c\rangle \in po; (po \cup rf)^*$), because if it were an rf edge, it could be folded into the preceding K yielding a shorter cycle. Similarly, the last edge in this path must be a po

edge, and so $\langle b,c\rangle \in po \cup (po;(po \cup rf)^*;po)$. Symmetrically, $\langle d,e\rangle \in po \cup (po;(po \cup rf)^*;po)$. In addition, by the definition of K, $loc(a) = loc(b)$ and $loc(c) = loc(d)$. If $loc(b) = loc(c)$, we obtain that $\langle a,d\rangle \in K$, which contradicts the minimality of the cycle. Similarly, we have $loc(d) \neq loc(e)$. Therefore, from the theorem's assumptions, there exist fence events f_1, f_2 such that $\langle b,f_1\rangle, \langle f_1,c\rangle, \langle d,f_2\rangle, \langle f_2,e\rangle \in (po \cup rf)^+$. From Prop. 8, we have $\langle f_1,f_2\rangle \in rf^+$ or $\langle f_2,f_1\rangle \in rf^+$, both cases yielding a cycle with fewer $K \setminus (po \cup rf)^+$ edges. □

Proof (of Thm. 4). For every location $x \in \mathsf{Loc}$, let mo_x be an x-modification order in G. We prove that $po \cup rf \cup \bigcup_x mo_x \cup \bigcup_x fr_x$ is acyclic (where $fr_x = (rf^{-1};mo_x) \setminus Id_A$ for every $x \in \mathsf{Loc}$). By Prop. 2, $po \cup rf \cup fr_x$ is acyclic for every $x \in \mathsf{Loc}$. Since G is WW-race free, we have that $\bigcup_x mo_x \subseteq (po \cup rf)^+$ (see the proof of Prop. 3). Furthermore, for every $x \in \mathsf{Loc}$, we have $fr_x \cap (G.\mathtt{U} \times A) \subseteq (po \cup rf)^+$. Indeed, if $\langle a,b\rangle \in fr_x$ and $a \in G.\mathtt{U}$, then by WW-race freedom, we must have $\langle a,b\rangle \in (po \cup rf)^+$ or $\langle b,a\rangle \in (po \cup rf)^+$. But, $\langle b,a\rangle \in (po \cup rf)^+$ implies a $po \cup rf \cup fr_x$ cycle.

Therefore, it suffices to prove that $R = po \cup rf \cup \bigcup_x fr_x{}'$ is acyclic, where $fr_x{}' = (fr_x \cap (G.\mathtt{R} \times A)) \setminus (po \cup rf)^+$ for every $x \in \mathsf{Loc}$. Let $R_B = (po \cup rf)^+ \cap (B \times B)$. R_B is a strict total order on B, that is contained in R^+. Since $po \cup rf \cup fr_x$ is acyclic for every $x \in \mathsf{Loc}$, we clearly have that R is irreflexive, and $R_B;R$ is irreflexive. By Lemma 2, to show that R is acyclic, it remains to prove the following, where $C = A \setminus B$, and $R_C = R \cap (C \times C)$:

- R_C is acyclic. This follows from the fact that $R_C \subseteq (po \cup rf)^+$. Indeed, suppose for contradiction that $\langle a,b\rangle \in R_C$ but $\langle a,b\rangle \notin (po \cup rf)^+$. Then, $\langle a,b\rangle \in fr_x \cap (C \times C)$ for some $x \in \mathsf{Loc}$. Since $po \cup rf \cup fr_x$ is acyclic, we cannot have $\langle b,a\rangle \in (po \cup rf)^+$. It follows that a and b race in G, and $a,b \notin B$ is not possible by our assumption.
- For every $b \in B$ and $c_1,c_2 \in C$, it is not the case that $\langle c_1,b\rangle, \langle b,c_2\rangle \in R$ and $\langle c_2,c_1\rangle \in R_C^*$. Suppose otherwise. Since $R_C \subseteq (po \cup rf)^+$, we have $\langle c_2,c_1\rangle \in (po \cup rf)^+$. Since $po \cup rf$ is acyclic, either $\langle c_1,b\rangle \in fr_x{}'$ or $\langle b,c_2\rangle \in fr_x{}'$ for some $x \in \mathsf{Loc}$. Both cannot hold, as together they imply that $b \in G.\mathtt{U}$. Hence, either $\langle c_1,b\rangle \in (po \cup rf)^+$ or $\langle b,c_2\rangle \in (po \cup rf)^+$. In both case, we obtain a contradiction to the fact that $po \cup rf \cup fr_x$ is acyclic for every $x \in \mathsf{Loc}$.
- For every $b_1,b_2 \in B$ and $c_1,c_2 \in C$, it is not the case that $\langle c_1,b_1\rangle, \langle b_2,c_2\rangle \in R$, $\langle b_1,b_2\rangle \in R_B$, and $\langle c_2,c_1\rangle \in R_C^*$. Suppose otherwise. In this case, we must have $\langle c_1,b_1\rangle \in fr_x{}'$, and $\langle b_2,c_2\rangle \in fr_y{}'$ for some $x \neq y$ (otherwise, we obtain a cycle in either $po \cup rf \cup fr_x$ or $po \cup rf \cup fr_y$). Then, $b_1,c_2 \in G.\mathtt{W} \cup G.\mathtt{U}$ and $b_2,c_1 \in G.\mathtt{R}$. In addition, we cannot have $\langle b_1,c_1\rangle \in (po \cup rf)^+$, and so c_1 and b_1 race in G. Similarly, c_2 and b_2 race in G. Our assumptions entail that $\langle b_1,f_1\rangle, \langle f_1,b_2\rangle \in (po \cup rf)^+$ for some fence event f_1, and $\langle c_2,f_2\rangle, \langle f_2,c_1\rangle \in (po \cup rf)^+$ for some fence or protected event f_2. Now, if f_2 is a protected event (i.e., $f_2 \in B$), then either $\langle f_2,b_1\rangle \in (po \cup rf)^+$ or $\langle b_1,f_2\rangle \in (po \cup rf)^+$. The first option implies a cycle in $po \cup rf \cup fr_y$, and the latter implies a cycle in $po \cup rf \cup fr_x$. Hence, f_2 must be a fence event. From Prop. 8, we have $\langle f_1,f_2\rangle \in rf^+$ or $\langle f_2,f_1\rangle \in rf^+$, and again we obtain a cycle in either $po \cup rf \cup fr_x$ or $po \cup rf \cup fr_y$. □

G. Proofs for §6 (Relation to TSO)

To prove Thm. 5, we first establish the following property that relates the $(po \cup rf)^+$ with the total store order.

Proposition 10. Let tso be a total store order for an execution $G = \langle A,po,rf\rangle$. Then, $(po \cup rf)^+ \subseteq tso \cup (po \cap (G.\mathtt{W} \times G.\mathtt{R}))$.

Proof. We use induction on the length of the $po \cup rf$ path to show that $(po \cup rf)^+ \subseteq tso \cup (po \cap (G.\mathtt{W} \times G.\mathtt{R}))$. The base case easily follows from the facts that $po \subseteq tso \cup (G.\mathtt{W} \times G.\mathtt{R})$ and $rf \subseteq tso \cup po$. For the induction step, suppose the claim holds for paths of size $< n$, and let $a_1,\ldots,a_n \in A$, such that $\langle a_k,a_{k+1}\rangle \in po \cup rf$ for every $1 \leq k \leq n-1$. If $\langle a_k,a_{k+1}\rangle \in po$ for every $1 \leq k \leq n-1$, then $\langle a_1,a_n\rangle \in po$, and we are done using the base case. Otherwise $\langle a_k,a_{k+1}\rangle \in rf \setminus po$ for some $1 \leq k \leq n-1$. By the induction hypothesis, $\langle a_1,a_k\rangle \in tso \cup (po \cap (G.\mathtt{W} \times G.\mathtt{R}))$ and $\langle a_{k+1},a_n\rangle \in tso \cup (po \cap (G.\mathtt{W} \times G.\mathtt{R}))$. Since $\langle a_k,a_{k+1}\rangle \in rf$, $typ(a_k) \neq \mathtt{R}$ and $typ(a_{k+1}) \neq \mathtt{W}$. Hence, $\langle a_1,a_k\rangle \in tso$ and $\langle a_{k+1},a_n\rangle \in tso$. In addition, $\langle a_k,a_{k+1}\rangle \in rf \setminus po$ implies that $\langle a_k,a_{k+1}\rangle \in tso$. The transitivity of tso implies that $\langle a_1,a_n\rangle \in tso$. □

Proof (of Thm. 5).

(⇒) Suppose that G is TSO-coherent, and let $tso \subseteq A \times A$ be a total store order for G. By Prop. 10, $\langle a,a\rangle \in (po \cup rf)^+$ implies $\langle a,a\rangle \in tso$. Hence, acyclicity of $po \cup rf$ follows from the fact that tso is irreflexive. Let $mo = tso \cap ((G.\mathtt{W} \cup G.\mathtt{U}) \times (G.\mathtt{W} \cup G.\mathtt{U}))$. We first show that mo is a modification order in G. Since tso is a strict partial order that is total on $G.\mathtt{W} \cup G.\mathtt{U}$, mo is a strict total order on $G.\mathtt{W} \cup G.\mathtt{U}$. To see that $mo;(po \cup rf)^+$ is irreflexive, let $\langle a,b\rangle \in (po \cup rf)^+$. If $b \in \mathcal{R}$, then clearly $\langle b,a\rangle \notin mo$. Otherwise, Prop. 10 entails that $\langle a,b\rangle \in tso$, and so $\langle b,a\rangle \notin mo$. Note that using Lemma 3 it also follows that $po \cup rf \cup mo$ is acyclic. Next, let $a,b,c \in A$, such that $\langle a,c\rangle \in rf$, $\langle b,c\rangle \in (po \cup rf)^+ \cup mo$, and $loc(a) = loc(b)$. If $b \notin G.\mathtt{W} \cup G.\mathtt{U}$, then clearly $\langle a,b\rangle \notin mo$. Suppose otherwise. We show that $\langle b,c\rangle \in tso \cup po$. It follows that $\langle a,b\rangle \notin tso$, and so $\langle a,b\rangle \notin mo$. First, if $\langle b,c\rangle \in (po \cup rf)^+$, then $\langle b,c\rangle \in tso \cup po$ follows from Prop. 10. Otherwise, $\langle b,c\rangle \in mo$, and by definition, $\langle b,c\rangle \in tso$.
Next, we prove that mo satisfies the two additional conditions of the theorem. Let $R = (mo;(rf \setminus po);po) \cup ((mo \cap (A \times G.\mathtt{U}));po)$, and let $a,b,c \in A$, such that $\langle a,c\rangle \in rf$, $\langle b,c\rangle \in R$, and $loc(a) = loc(b)$. If $b \notin G.\mathtt{W} \cup G.\mathtt{U}$, then clearly $\langle a,b\rangle \notin mo$. Suppose otherwise. Again, we show that $\langle b,c\rangle \in tso \cup po$. Consider the two possible cases:
  - $\langle b,c\rangle \in mo;(rf \setminus po);po$. In this case, we have $\langle b,b_1\rangle \in mo$, $\langle b_1,b_2\rangle \in rf \setminus po$, and $\langle b_2,c\rangle \in po$ for some $b_1,b_2 \in A$. By definition, $\langle b,b_1\rangle \in tso$. Since tso is a total store order in G, we also have $\langle b_1,b_2\rangle \in tso$. Since $\langle b_1,b_2\rangle \in rf$, $b_2 \notin G.\mathtt{W}$, and so, since tso is a total store order in G, we have $\langle b_2,c\rangle \in tso$ as well. The transitivity of tso implies that $\langle b,c\rangle \in tso$.
  - $\langle b,c\rangle \in (mo \cap (A \times G.\mathtt{U}));po$. In this case, $\langle b,b_1\rangle \in mo$ and $\langle b_1,c\rangle \in po$ for some $b_1 \in G.\mathtt{U}$. By definition, $\langle b,b_1\rangle \in tso$. Since $b_1 \in G.\mathtt{U}$, and since tso is a total store order in G, we have $\langle b_1,c\rangle \in tso$ as well. Again, the transitivity of tso implies that $\langle b,c\rangle \in tso$.

(⇐) Suppose that $po \cup rf$ is acyclic, and let mo be a modification order in G the additional conditions of the theorem. We show that $tso = (mo \cup (po \setminus (G.\mathtt{W} \times A)) \cup (rf \setminus po))^+$ is a total store order for G, and so G is TSO-coherent. First, by Lemma 3, since $po \cup rf$ is acyclic, the fact that $mo;(po \cup rf)^+$ is irreflexive entails that $mo \cup po \cup rf$ is acyclic. Thus, tso is a strict partial order on A. Since mo is total on $G.\mathtt{W} \cup G.\mathtt{U}$, so is tso, and we clearly also have $rf \subseteq tso \cup po$. To see that $po \subseteq tso \cup (G.\mathtt{W} \times G.\mathtt{R})$, let $\langle a,b\rangle \in po$. If $a \notin \mathcal{W}$, then $\langle a,b\rangle \in tso$ by definition. If $a \in \mathcal{W}$ and $b \in \mathcal{R}$, then we are obviously done. Otherwise, $a \in \mathcal{W}$ and $b \in \mathcal{W} \cup \mathcal{U}$. Since mo is total on $G.\mathtt{W} \cup G.\mathtt{U}$, and $mo;po$ is irreflexive, we have $\langle a,b\rangle \in mo$, and so $\langle a,b\rangle \in tso$ in this case as well.

Next, let $a, b, c \in A$, such that $\langle a, c\rangle \in rf$, $\langle b, c\rangle \in po \cup tso$, $b \in G.\mathtt{W} \cup G.\mathtt{U}$, and $loc(a) = loc(b)$. We show that $\langle a, b\rangle \notin tso$. Since mo is total on $G.\mathtt{W} \cup G.\mathtt{U}$, it suffices to show that $\langle a, b\rangle \notin mo$. Let $R = (po \cup rf)^+ \cup mo \cup (mo; (rf \setminus po); po) \cup ((mo \cap (A \times G.\mathtt{U})); po)$. We prove that $\langle b, c\rangle \in R$, and $\langle a, b\rangle \notin mo$ follows from the properties of mo. First, if $\langle b, c\rangle \in po$, then we are clearly done. Otherwise, $\langle b, c\rangle \in tso$. Let $a_1, \ldots, a_n \in A$, such that $a_1 = b$, $a_n = c$, and $\langle a_k, a_{k+1}\rangle \in mo \cup (po \setminus (G.\mathtt{W} \times A)) \cup (rf \setminus po)$ for every $1 \leq k \leq n-1$. Let m be the maximal index such that $a_m \in G.\mathtt{W} \cup G.\mathtt{U}$ (such a_m exists since $b \in G.\mathtt{W} \cup G.\mathtt{U}$). Since $\langle b, c\rangle \notin po$, we have $m > 1$. Since $po \cup rf \cup mo$ is acyclic, mo is total on $G.\mathtt{W} \cup G.\mathtt{U}$, and $b \in G.\mathtt{W} \cup G.\mathtt{U}$, we must have $\langle b, a_m\rangle \in mo$. If $a_m = c$, then $\langle b, c\rangle \in mo$, and we are done. Assume otherwise. Then, we have $\langle a_{m+1}, c\rangle \in po$. Now, if $\langle a_m, a_{m+1}\rangle \in rf \setminus po$, then $\langle b, c\rangle \in mo; (rf \setminus po); po$, and we are done. Otherwise, $\langle a_m, a_{m+1}\rangle \in po \setminus (G.\mathtt{W} \times A)$. Thus, $a_m \in G.\mathtt{U}$, and so $\langle b, c\rangle \in (mo \cap (A \times G.\mathtt{U})); po$. □

Proof (of Thm. 6). By Proposition 7, there exists a relation mo that is a modification order in G. Let $fr_B = \{\langle a, b\rangle \in rf^{-1}; mo \mid a \in B, loc(a) = loc(b)\}$. First, we show that $po \cup rf \cup fr_B$ is acyclic. We will prove by induction on n that fr_B cannot appear in such a cycle n times. For $n = 0$, this follows from the fact that $po \cup rf$ is acyclic. For $n = 1$, this follows from the fact that mo is a modification order in G (see Prop. 2). For $n \geq 2$, take the sources of two distinct fr_B edges. As they both belong to B, they are ordered by $(po \cup rf)^+$, thereby witnessing a smaller cycle.

Now, take mo' to be any strict total order on $G.\mathtt{W}$ extending $(po \cup rf \cup fr_B)^+ \cap (G.\mathtt{W} \times G.\mathtt{W})$. We show that mo' satisfies the conditions of Thm. 5, and hence, G is TSO-coherent. By construction, mo' is a strict total order on $G.\mathtt{W}$, and $mo'; (po \cup rf)^+$ is irreflexive. To see that mo' is a modification order in G, let $a, b, c \in A$, such that $\langle a, c\rangle \in rf$, $\langle b, c\rangle \in (po \cup rf)^+$, and $loc(a) = loc(b)$ (note that $G.\mathtt{U}$ is empty, so we do not need to consider the case that $\langle b, c\rangle \in mo'$). We prove that $\langle a, b\rangle \notin mo'$. Suppose otherwise. Then, both a and b are write events and $\langle b, a\rangle \notin (po \cup rf)^+$. Since G is WW-race free, it follows that $\langle a, b\rangle \in (po \cup rf)^+$. Since mo is a modification order in G, we also have $\langle a, b\rangle \in mo$. This contradicts the fact that $\langle b, c\rangle \in (po \cup rf)^+$ (again, since mo is a modification order in G).

Next, let $a, b, c \in A$, such that $\langle a, c\rangle \in rf$, $\langle b, c\rangle \in mo'; (rf \setminus po); po$, and $loc(a) = loc(b)$ (note that $G.\mathtt{U}$ is empty, so we do not need to consider the case that $\langle b, c\rangle \in (mo' \cap (A \times G.\mathtt{U})); po$). We prove that $\langle a, b\rangle \notin mo'$. Suppose otherwise. As above, a and b are write events, and we have $\langle a, b\rangle \in (po \cup rf)^+$, $\langle a, b\rangle \in mo$, and $\langle b, c\rangle \notin (po \cup rf)^+$. Let $b_1, b_2 \in A$, such that $\langle b, b_1\rangle \in mo'$, $\langle b_1, b_2\rangle \in rf \setminus po$, and $\langle b_2, c\rangle \in po$. Note that $\langle b_1, b\rangle \notin (po \cup rf \cup fr_B)^+$ (otherwise, $\langle b_1, b\rangle \in mo'$, and this contradicts the fact that mo' is acyclic). Hence, the following hold:

1. $c \notin B$. Indeed, otherwise we have $\langle c, b\rangle \in fr_B$, and so $\langle b_1, b\rangle \in (po \cup rf \cup fr_B)^+$.
2. $b \in B$. Indeed, otherwise, c and b cannot race in G, and hence we must have $\langle c, b\rangle \in (po \cup rf)^+$, and again $\langle b_1, b\rangle \in (po \cup rf \cup fr_B)^+$.
3. $b_1 \notin B$. Indeed, otherwise we must have $\langle b, b_1\rangle \in (po \cup rf)^+$, and this contradicts the fact that $\langle b, c\rangle \notin (po \cup rf)^+$.

Since $\langle b_1, b_2\rangle \in rf \setminus po$ and $b_1 \notin B$, we have $b_2 \in B$. Hence, we have either $\langle b, b_2\rangle \in (po \cup rf)^+$ or $\langle b_2, b\rangle \in (po \cup rf)^+$. In the first case, we obtain a contradiction to the fact that $\langle b, c\rangle \notin (po \cup rf)^+$. The latter again contradicts the fact that $\langle b_1, b\rangle \notin (po \cup rf \cup fr_B)^+$. □

H. Proofs for §7 (Relation to Power)

H.1 General Properties of the Power Model

Let $G = \langle A, po, deps, rf, At\rangle$ be a Power execution.

Proposition 11. Assuming [SC-per-loc] holds, [Observation] is equivalent to $fre; base$ being irreflexive.

Proof. In Coq, lemma `observation_condition_simpl`. □

Define the following:

$$fence_{weak} \triangleq fence \cap ((\mathcal{R} \times (\mathcal{R} \cup \mathcal{W})) \cup (\mathcal{W} \times \mathcal{W}))$$
$$hb_{weak} \triangleq ppo \cup fence_{weak} \cup rfe$$
$$base_{weak} \triangleq rfe^?; fence_{weak}; hb^*_{weak}$$
$$prop_{weak} \triangleq (base_{weak} \cap (\mathcal{W} \times \mathcal{W})) \cup chapo^?; base^*_{weak}; sync; hb^*_{weak}$$

Proposition 12. The following properties hold:

- $hb = hb_{weak} \cup (sync \cap (\mathcal{W} \times \mathcal{R}))$
- $hb^* = hb^*_{weak}; (sync \cap (\mathcal{W} \times \mathcal{R}))^?$
- $hb^+ = hb^+_{weak} \cup (hb^*_{weak}; (sync \cap (\mathcal{W} \times \mathcal{R}))$
- $base = base_{weak} \cup (sync \cap \mathcal{W} \times \mathcal{R})$
- $prop = prop_{weak}$

Proof. In Coq, lemmas `hbE`, `rt_hbE`, `t_hbE`, `prop_baseE`, and `baseE`. □

Proposition 13. The following properties hold:

- hb is acyclic iff hb_{weak} is acyclic.
- Assuming [SC-per-loc] and [No-thin-air] hold, then [Observation] is equivalent to $fre; base_{weak}$ being irreflexive.

Proof. In Coq, lemmas `acyclic_hbE`, and `observation_condition_alternative`. □

H.2 Power Fences

We consider Power executions $G_s = \langle A, po, deps_s, rf, At\rangle$ and $G_r = \langle A_r, po_r, deps_r, rf_r, At_r\rangle$ such that $G_s \approx G_r$ ($\approx$ is defined in §7). Moreover we define the following:

$$po_{\mathtt{S}}^{\rightarrow} \triangleq po \cap ((\mathcal{W} \cup \mathcal{R}) \times \mathcal{S}) \text{ and}$$
$$po_{\mathtt{S}}^{\leftarrow} \triangleq po \cap (\mathcal{S} \times (\mathcal{W} \cup \mathcal{R}))$$

Notice that $sync \equiv po_{\mathtt{S}}^{\rightarrow}; po_{\mathtt{S}}^{\leftarrow}$.

Notation 6. Let R be a binary relation on events. We denote by $\mathtt{T_1T_2}(R)$ the restriction of R to events of types $\mathtt{T_1}$ and $\mathtt{T_2}$, i.e., $R = \{\langle a, b\rangle \in R \mid typ(a) = \mathtt{T_1}, typ(b) = \mathtt{T_2}\}$.

H.2.1 Our Fences are Stronger than Power's Strong Fences

In this section we assume a Power-coherent execution with RMW semantics for strong fences ($G_r = \langle A_r, po_r, deps_r, rf_r, At_r\rangle$) and prove that the corresponding execution with `sync` fences ($G_s = \langle A, po, deps, rf, At\rangle$) is also Power-coherent. We define an order on write events in G_s such that it is identical to co_r but does not include the writes to the special location f. Formally, $co \triangleq \{\langle a, b\rangle \mid \langle a, b\rangle \in co_r \wedge loc(a) \neq f\}$. Obviously, co is a coherence order in G_s.

Lemma 7. $hb \setminus (\mathcal{W} \times \mathcal{R}) \subseteq hb_r$.

Proof. The only case that is not straightforward is when two events are ordered by a `sync` fence. Suppose a, b such that $\langle a, b\rangle \in sync$, i.e. a is po-before a `sync` fence and b is po-after it. In the execution with release-acquire RMWs as fences this implies that there will

be a `lwsync` fence po-between a, b thus $\langle a, b\rangle \in lwsync$ which is sufficient to order by hb_r any events $\langle a, b\rangle \notin \mathcal{W} \times \mathcal{R}$. □

Lemma 8. $hb^+ \setminus po \subseteq hb_r^+$.

Proof. By induction on $\langle a, b\rangle \in hb^+$.

[Base Case] $\langle a, b\rangle \in hb$. Since $\langle a, b\rangle \notin po$ it must be that $\langle a, b\rangle \in rfe$ thus since $rfe \subseteq rfe_r$ we obtain $\langle a, b\rangle \in hb_r$.

[Inductive Case] $\langle a, c\rangle \in hb^+$ and $\langle c, b\rangle \in hb^+$. We distinguish four cases:

- $\langle a, c\rangle \in po$ and $\langle c, b\rangle \in po$. Then $\langle a, b\rangle \in po$, contradiction.
- $\langle a, c\rangle \notin po$ and $\langle c, b\rangle \in po$ then by induction hypothesis $\langle a, c\rangle \in hb_r^+$. Consider the non-trivial case where $\langle c, b\rangle \in sync$, $typ(c) = \mathtt{W}$, and $typ(b) = \mathtt{R}$ (notice that in any other case $hb \subseteq hb_r$). Then since $\langle a, c\rangle \notin po$ there must exist some write d and some read e such that $\langle d, e\rangle \in rfe$, $\langle e, c\rangle \in po$ and thus $\langle e, b\rangle \in hb_r$ because there is a `lwsync` po-between e and b (placed before the release-acquire RMW that is used instead of a `sync`). Thus $\langle a, b\rangle \in hb_r^+$.
- $\langle a, c\rangle \in po$ and $\langle c, b\rangle \notin po$. Then by induction hypothesis $\langle c, b\rangle \in hb_r^+$. We examine the same case as above ($typ(a) = \mathtt{W}, typ(c) = \mathtt{R}, \langle a, c\rangle \in sync$), obviously c and b are not on the same thread, thus there must exist some write d and some read e such that $\langle c, d\rangle \in hb_r$, $\langle d, e\rangle \in rfe_r$ and $\langle e, b\rangle \in hb_r^*$. But instead of the `sync` there is a `lwsync` (placed before the RMW) po-between a and d which is sufficient to order a, d by hb_r thus $\langle a, b\rangle \in hb_r^+$.
- $\langle a, c\rangle \notin po$ and $\langle c, b\rangle \notin po$. By induction hypotheses we obtain $\langle a, b\rangle \in hb_r^+$. □

Lemma 9. For every $x \in \mathsf{Loc}$, the relation $po|_x \cup rf \cup co \cup fr$ is acyclic.

Proof. Follows by the *SC-per-loc* axiom because $po|_x \subseteq po_r|_x$, $rf \subseteq rf_r$, $co \subseteq co_r$, and $fr \subseteq fr_r$. □

Lemma 10. The relation $rmw \cap (fre; coe)$ is empty.

Proof. Notice that $rmw \subseteq rmw_r$ (rmw does not include the RMWs at location f), thus since $rmw_r \cap (fre_r; coe_r)$ is empty, so is $rmw \cap (fre; coe)$. □

Lemma 11. The relation hb is acyclic.

Proof. Suppose by contradiction that there is a cycle in hb. Then by Prop. 13 there is a cycle in $hb \setminus \mathtt{WR}(sync)$. This implies that $\langle a, b\rangle \in \mathtt{WR}(hb) \implies \langle a, b\rangle \in rfe$, thus a cycle in $hb \setminus \mathtt{WR}(sync)$ implies a cycle in $\mathtt{RT}(hb) \cup \mathtt{WW}(hb) \cup rfe$ and by Lemma 7 and the fact that $rfe \subseteq rfe_r$ we obtain a cycle in hb_r, contradiction. □

Remark 2.

$$
\begin{array}{ll}
 & acyclic(hb) \\
\wedge & acyclic(fre; rfe^?; base^*; sync; hb^*) \\
\wedge & acyclic(co \cup \mathtt{WW}(base) \cup po\text{-}aa \\
 & \quad \cup (rfe^?; base^*; sync; hb^*); \\
 & \quad\ (fre; rfe^?; base^*; sync; hb^*)^*) \\
\implies & acyclic(co \cup \mathtt{WW}(base) \cup po\text{-}aa \\
 & \quad \cup (rfe^?; base^*; sync; hb^*) \\
 & \quad \cup (fre; rfe^?; base^*; sync; hb^*))
\end{array}
$$

Notation 7. We will use a numerical subscript (1 and 2) to denote the read and write event respectively of a rmw, e.g. $\langle a_1, a_2\rangle \in rmw$.

Let $\mathcal{L}$ be an order on events of type $\langle \mathtt{sync}\rangle$ such that $\langle a, b\rangle \in \mathcal{L}$ iff $\langle a_2, b_2\rangle \in co_r$ where a_2, b_2 are the atomic writes that are used in the G_r execution instead of the $sync$ fences a, b.

Lemma 12. Let $a, b \in \mathcal{S}$ s.t. $\langle a, b\rangle \in \mathcal{L}$ and $\langle a, b\rangle \notin po$, then for all a', b' s.t. $typ(a'), typ(b') \in \{\mathtt{W}, \mathtt{R}\}$, $\langle a', a\rangle \in po$, and $\langle b, b'\rangle \in po$ it holds that $\langle a', b'\rangle \in base_r$.

Proof. Notice that in the execution G_r the strong fences a, b correspond to release-acquire RMWs. This implies that there is a light fence (`lwsync`) before the RMW and a control dependency followed by an `isync` after it. Thus for any event a' that po-precedes a it is the case that $\langle a', a_2\rangle \in lwsync$ where a_2 is the write that corresponds to the fence event a. Moreover since every atomic read c_1 reads from the write that immediately precedes c_2 in the coherence order co_r, we conclude that $\langle a', b_1\rangle \in fence; hb_r^+$. Finally due to the control-dependency and the `isync` fence that po-succeeds b_1 we have that $\langle a', b'\rangle \in fence; hb_r^+; ppo$ which implies $\langle a', b'\rangle \in base_r$. □

Remark 3. Let T be some strict total order on a set A. Consider any finite enumeration $a_1 \ldots a_n$ of A, then either $\forall\ i \in [1..n).\ \langle a_i, a_{i+1}\rangle \in T$ or there exists some i such that $\langle a_{i+1}, a_i\rangle \in T$.

Lemma 13. The relation $(fre; rfe^?; base^*; sync; hb^*)$ is acyclic.

Proof. Let $S \triangleq fre; rfe^?; base^*; sync; hb^*$. There are two cases:

- If $\langle a, a\rangle \in S$.
  Equivalently
  $\langle a, b\rangle \in fre$
  $\wedge\ \langle b, c\rangle \in rfe^?$
  $\wedge\ \langle c, a\rangle \in base^*; sync; hb^*$
  By case analysis on $\langle b, c\rangle \in rfe^?$
  - ◇ If $\langle b, c\rangle \in rfe$ and $\langle c, a\rangle \in po$. This contradicts the *sc-per-loc* axiom in the G_r execution.
  - ◇ If $\langle b, c\rangle \in rfe$ and $\langle a, c\rangle \in po$. Thus there must exist some event d s.t. $typ(d) = \mathtt{W}$, $\langle c, d\rangle \in fence$, and $\langle d, a\rangle \in hb^+$. Notice that since $typ(d) = \mathtt{W}$, it must be that $\langle a, d\rangle \in fence_r$ and moreover by Lemma 8 we have that $\langle d, a\rangle \in hb_r^+$. Thus $\langle a, a\rangle \in hb_r^+$ which contradicts the *no-thin-air* axiom of the G_r execution.
  - ◇ If $\langle b, c\rangle \in rfe$ and $\langle a, c\rangle \notin po \wedge \langle c, a\rangle \notin po$. Then there must exist some event d s.t. $typ(d) = \mathtt{W}$, $\langle c, d\rangle \in fence_s$, and $\langle d, a\rangle \in hb^+$. Notice that since $typ(d) = \mathtt{W}$ it must be that $\langle c, d\rangle \in fence_r$ and thus $\langle b, d\rangle \in \mathtt{WW}(base_r)$. Moreover, by Lemma 8 $\langle d, a\rangle \in hb_r^+$ thus $\langle a, a\rangle \in fre_r; prop; hb_r^+$ which contradicts the *observation* axiom of the G_r execution.
  - ◇ If $b = c$. Then there must exist some event d s.t. $typ(d) = \mathtt{W}$, $\langle b, d\rangle \in fence$, and $\langle d, a\rangle \in hb^+$ which implies $\langle b, d\rangle \in fence_r$ and $\langle d, a\rangle \in hb_r^+$ by Lemma 8. Thus $\langle a, a\rangle \in fre_r; prop; hb_r^+$ which contradicts the *observation* axiom of the G_r execution.
- If $\langle a, a\rangle \in S; S^+$. Notice that $\mathcal{L}$ is a strict total order on the set of events $G_s.\langle \mathtt{sync}\rangle$. Take an enumeration of $G_s.\langle \mathtt{sync}\rangle$ s.t. for any $s_i, s_j \in G_s.\langle \mathtt{sync}\rangle$, $i < j$ iff there is a subpath of $\langle a, a\rangle \in S^+$ that starts at s_i and ends at s_j. By Remark 3 we distinguish two cases:
  - ◇ In case there exists $s_i \in G_s.\langle \mathtt{sync}\rangle$ s.t. $\langle s_{i+1}, s_i\rangle \in \mathcal{L}$:
    - ○ If for some b, c it holds that $\langle b, c\rangle \in sync; hb^+_{\setminus sync}; sync$ and $\langle c_s, b_s\rangle \in \mathcal{L}$ (where $\langle b, b_s\rangle \in po_{\mathtt{S}}^{\rightarrow}$ and $\langle c_s, c\rangle \in po_{\mathtt{S}}^{\leftarrow}$) then if $\langle c_s, b_s\rangle \in po$ then either po is cyclic (contradiction by *sc-per-loc* axiom) or otherwise there

is some write b' such that $\langle b, b'\rangle \in fence^?$. Notice then that $\langle b', c\rangle \in hb_r^+$ by Lemma 8 and $\langle c, b'\rangle \in hb_r$ (because $\langle c, b'\rangle \in po$ and there is an `lwsync` fence between them). Otherwise if $\langle c_s, b_s\rangle \notin po$, then notice that $hb_{\setminus sync} \subseteq hb_r$, hence $\langle b', c'\rangle \in hb_r^+$ where ($\langle b, b'\rangle \in sync$, $\langle c', c\rangle \in sync$). Moreover, by Lemma 12 we have that $\langle c', b'\rangle \in base_r$ which implies $\langle c', b'\rangle \in hb_r^+$ and thus $\langle c', c'\rangle \in hb_r^+$. But this contradicts the *no-thin-air* axiom.

- If for some b, c it holds that $\langle b, c\rangle \in sync; hb^*_{\setminus sync}; fre; rfe^?; base^*_{\setminus sync}; sync$ or equivalently that
  1. $\langle b, d\rangle \in sync; hb^*_{\setminus sync}$
  2. $\langle d, c'\rangle \in fre; base^*_{\setminus sync}$
  3. $\langle c', c\rangle \in sync$

  and $\langle c_s, b_s\rangle \in \mathcal{L}$ then if $\langle c_s, b_s\rangle \in po$, we distinguish the following cases:
  - ⋆ If $\langle b, d\rangle \in po$ then if $\langle d, c'\rangle \in fre$ we have a cycle in $po|l \cup fr$ where $l = loc(c')$.
  - ⋆ If $\langle b, d\rangle \in po$, if $\langle d, c'\rangle \in fre; base_{\setminus sync}$ then consider the case where $\langle d, c'\rangle \notin fre; po$ (otherwise it is the same as above). Then $\langle d, c'\rangle \in fre; base_{\setminus sync}$ implies $\langle d, d'\rangle \in fre$ and $\langle d', d''\rangle \in \mathtt{WW}(base_{\setminus sync})$, $\langle d'', c'\rangle \in hb^*_{\setminus sync}$ (because $\langle d', c'\rangle \in base_{\setminus sync}$ implies that $\langle d', c'\rangle \in rfe^?; lwsync; hb^*_{\setminus sync}$, equivalently $\langle d', d''\rangle \in rfe^?; lwsync$ which implies that either $\langle d', d''\rangle \in lwsync$ and d'' must be a write because $\langle d'', c'\rangle \notin po$, or $\langle d', d'''\rangle \in rfe$ and if $\langle d''', d\rangle \in po$ then we have a violation of the *sc-per-loc* axiom or $\langle d''', d\rangle \notin po$ and there must be some write t such that $\langle d''', t\rangle \in fence$ and $\langle t, d''\rangle \in rfe$). Finally, there must be some read c'' such that $\langle c'', c'\rangle \in hb_r^?$, $\langle d'', c''\rangle \in hb_r^+$ and $\langle c'', d\rangle \in ppo$. Thus we have that $\langle d, d\rangle \in fre; prop$.
  - ⋆ If $\langle b, d\rangle \in sync; hb^+_{\setminus sync}$ (and $\langle b, d\rangle \notin po$) then there must exist some write b' such that $\langle b, b'\rangle \in sync$ which implies $\langle b, b'\rangle \in hb_r$ by Lemma 8 and by the fact that $hb_{\setminus sync} \subseteq hb_r$ we have that $\langle b', d\rangle \in hb_r^+$. Moreover $\langle d, c'\rangle \in fre; base^*_{\setminus sync}$ implies $\langle d, c'\rangle \in fre_r; base_r^*$ and finally $\langle c', b'\rangle \in hb_r$ (there will be an `lwsync` fence between them from the RMW) which implies that $\langle d, b'\rangle \in \mathtt{WW}(base_r^+)$ thus again we have that $\langle d, d\rangle \in fre_r; prop$.

  Otherwise, we have that $\langle b', d\rangle \in hb_r^+$ and $\langle d, c'\rangle \in fre_r; rfe_r^?; base_r^*$. Moreover, by Lemma 12 we have that $\langle c', b'\rangle \in hb_r^+$. Notice that $\langle c', c_2\rangle \in fence_r$ (where c_2 is the atomic write that corresponds to c_s). Thus $\langle d, c_2\rangle \in fre_r; \mathtt{WW}(base_r^+)$ which implies $\langle d, c_2\rangle \in fre_r; prop$. Finally it must be that $\langle c_2, d\rangle \in hb_r^+$ hence we have a contradiction of the *observation* axiom.

◇ In case $\langle s_i, s_{i+1}\rangle \in \mathcal{L}$ for all i. Suppose $\langle a, a\rangle \in fre; rfe^?; base^*_{\setminus sync}; sync; hb^*; S^*; S$. This implies that there exists b_s, c_s s.t

1. $\langle a, b_s\rangle \in fre; rfe^?; base^*_{\setminus sync}; po_{\mathtt{S}}^{\rightarrow}$,
2. $\langle b_s, c_s\rangle \in po_{\mathtt{S}}^{\leftarrow}; hb^*; S^*; fre; rfe^?; base^*; po_{\mathtt{S}}^{\rightarrow}$,
3. $\langle c_s, a\rangle \in po_{\mathtt{S}}^{\leftarrow}; hb^*_{\setminus sync}$ and
4. $\langle b_s, c_s\rangle \in \mathcal{L}$.

We distinguish two cases:

- If $\langle b_s, c_s\rangle \in po$ then a case analysis similar to the previous case is required and we reach a contradiction either through *sc-per-loc* or through the *observation* axiom.
- If $\langle b_s, c_s\rangle \notin po$ then notice that $\langle a, b_2\rangle \in fre_r; \mathtt{WW}(base_r)$ where b_2 is the atomic write that corresponds to b_s. Moreover $\langle b_2, c_1\rangle \in hb_r^+$ (where c_1 is the atomic read that corresponds to c_s). Finally $\langle c_1, a\rangle \in ppo; hb_r^*$ thus $\langle a, a\rangle \in fre_r; prop; hb_r^+$ which contradicts the *observation* axiom. □

Lemma 14. The relation $(co \cup \mathtt{WW}(base) \cup po\text{-}aa \cup (rfe^?; base^*; sync; hb^*); (fre; rfe^?; base^*; sync; hb^*)^*)$ is acyclic.

Proof.

- Notice that $co \cup \mathtt{WW}(base) \cup po\text{-}aa$ is acyclic because it is included in $co_r \cup \mathtt{WW}(base_r) \cup po\text{-}aa_r$ which is acyclic by the *propagation* axiom.
- If there is a cycle in the second relation then a similar argument to the one used in the proof of Lemma 13 applies. In particular there are two cases again. In the first case we have again that $\langle s_{i+1}, s_i\rangle \in \mathcal{L}$ for some i, and the proof is exactly as the first case of Lemma 13. In the other case, we can easily conclude that there is a cycle in hb_r and thus the *no-thin-air* axiom is contradicted.
- Finally if there is a cycle in the union of the above relations, we again distinguish between the two cases of Remark 3. If $\langle s_i, s_{i+1}\rangle \in \mathcal{L}$ for all i then notice that for all subpaths of the cycle of the form $\langle b, c\rangle \in rfe^?; base^*; sync; hb^*;$ $(fre; rfe^?; base^*; sync; hb^*)^*$ it must be that $typ(b) = typ(c) = \mathtt{W}$ because $(co \cup \mathtt{WW}(base) \cup po\text{-}aa) \subseteq \mathcal{W} \times \mathcal{W}$. We will prove that $\langle b, c\rangle \in \mathtt{WW}(base_r)$ We distinguish two cases:
  - ◇ If $\langle b, c\rangle \in base^*; sync; hb^*$ then if $\langle b, c\rangle \in po$ we obviously have $\langle b, c\rangle \in \mathtt{WW}(base_r)$ (because there will be a `lwsync` fence po-between b, c). Otherwise if $\langle b, c\rangle \notin po$ then we have that
    1. $\langle b, b'\rangle \in (rfe^?; fence; hb^*)^?$
    2. $\langle b', c\rangle \in sync; hb^*$

    The proof proceeds by case analysis on the above relations. In all cases we have that $\langle b, c\rangle \in \mathtt{WW}(base_r)$.
  - ◇ Let $S \triangleq fre; rfe^?; base^*; sync; hb^*$ then we have that
    1. $\langle b, b'\rangle \in rfe^?; base^*_{\setminus sync}$
    2. $\langle b', b_s\rangle \in po_{\mathtt{S}}^{\rightarrow}$
    3. $\langle b_s, c_s\rangle \in po_{\mathtt{S}}^{\leftarrow}; hb^*; S^*; fre; rfe^?; base^*_{\setminus sync}; po_{\mathtt{S}}^{\rightarrow}$
    4. $\langle c_s, c'\rangle \in po_{\mathtt{S}}^{\leftarrow}$
    5. $\langle c', c\rangle \in hb^*_{\setminus sync}$

    If $\langle b_s, c_s\rangle \in po$ then, we have two cases if $\langle b', c\rangle \in po$ since $typ(b') = typ(c) = \mathtt{W}$ it must be that $\langle b', c\rangle \in fence_r$ thus we have that $\langle b, c\rangle \in rfe^?; base^*_{\setminus sync}; fence_r$ which implies $\langle b, c\rangle \in \mathtt{WW}(base_r)$, otherwise if $\langle b', c\rangle \notin po$ there must be some write t such that $\langle b', t\rangle \in fence_r$ and $\langle t, c\rangle \in hb^*_{\setminus sync}$ which implies that $\langle b, c\rangle \in base_r$. Otherwise if $\langle b_s, c_s\rangle \notin po$ then by Lemma 12 we have that $\langle b', c'\rangle \in base_r$ and by the fact that $hb_{\setminus sync} \subseteq hb_r$ and $rfe^?; base^*_{\setminus sync} \subseteq base_r^*$ we have that $\langle b, c\rangle \in \mathtt{WW}(base_r)$.

  Hence we can reduce the cycles to the form $co_{base} \cup \mathtt{WW}(base_r) \cup po\text{-}aa_r$ which is acyclic by the *propagation* axiom. Otherwise, if $\langle s_{i+1}, s_i\rangle \in \mathcal{L}$ for some i then there are three cases: Either there exists b, c s.t. $\langle b, c\rangle \in sync; hb^*_{\setminus sync}; fre; rfe^?; base^*_{\setminus sync}; sync$ and $\langle c_s, b_s\rangle \in \mathcal{L}$ which leads to a contradiction as shown in the proof of Lemma 13 or $\langle b, c\rangle \in sync; hb^*_{\setminus sync}; rfe^?; base^*_{\setminus sync}; sync$ and $\langle c_s, b_s\rangle \in \mathcal{L}$ which leads to a contradiction of the *no-thin-air* or *sc-per-loc* axioms.
  or $\langle b, c\rangle \in sync; hb^*_{\setminus sync};$ $(co \cup po\text{-}aa \cup \mathtt{WW}(base_{\setminus sync}))^+; rfe^?; base^*_{\setminus sync}; sync$ and $\langle c_s, b_s\rangle \in \mathcal{L}$.

Equivalently $\langle b,d\rangle \in sync; hb^*_{\setminus sync}$,
$\langle d,e\rangle \in (co \cup po\text{-}aa \cup \mathtt{WW}(base_{\setminus sync}))^+$,
$\langle e,g\rangle \in rfe^?; base^*_{\setminus sync}; sync$ and $\langle g_s, b_s\rangle \in$. Notice that $typ(e) = typ(d) = \mathtt{W}$ and thus $\langle e,d\rangle \in \mathtt{WW}(base_r)$ which implies that there is a cycle in $co_{base} \cup \mathtt{WW}(base_r) \cup po\text{-}aa_r$ which contradicts the *propagation* axiom. □

Lemma 15. $(co \cup prop \cup po\text{-}aa)$ is acyclic.

Proof. Notice that $chapo^?; base^*; sync; hb^*$ is equivalent to $(coe \cup fre)^?; rfe^?; base^*; sync; hb^*$ and hence $co \cup po\text{-}aa \cup \mathtt{WW}(base) \cup (coe \cup fre)^?; rfe^?; base^*; sync; hb^*$ is equivalent to $co \cup \mathtt{WW}(base) \cup po\text{-}aa \cup (rfe^?; base^*; sync; hb^*) \cup (fre; rfe^?; base^*; sync; hb^*))$. We conclude by using Remark 2 and Lemmas 11, 13 and 14. □

Lemma 16. $fre; prop; hb^*$ is irreflexive.

Proof. By Prop. 13 and Lemmas 9 and 11, it suffices to show that $fre; base_{weak}$ is irreflexive. But since $fre \subseteq fre_r$ and $base_{weak} \subseteq base_r$, this follows from the *observation* axiom for G_r. □

Theorem 9. G_s is Power-coherent.

Proof. The result is a direct consequence of Lemmas 9 to 11, 15 and 16. □

H.2.2 Power's Strong Fences are Stronger than our Fences

We assume now that $G_s = \langle A_s, po_s, deps_s, rf_s, At_s\rangle$ is a complete and Power-coherent execution and prove that the corresponding execution $G^0_r = \langle A, po, deps, \emptyset, At\rangle$ is Power-consistent.

Notation 8. We make use of the following notations for any binary relation R on events:

1. $R|_f \equiv \{\langle a,b\rangle \in R \mid loc(a) = loc(b) = f\}$
2. $R_{fx} \equiv \{\langle a,b\rangle \in R \mid loc(a) = f, loc(b) \neq f\}$
3. $R_{xf} \equiv \{\langle a,b\rangle \in R \mid loc(a) \neq f, loc(b) = f\}$
4. $R_{xy} \equiv \{\langle a,b\rangle \in R \mid loc(a) \neq f, loc(b) \neq f\}$

Definition 23. We define a partial order on all events of type $\langle\mathtt{sync}\rangle$:

$$B \triangleq \begin{pmatrix} po_s; (hb_{\setminus sync})^*; \\ (co_s \cup po\text{-}aa \cup \mathtt{WW}(base_{\setminus sync}))^*; \\ chapo^?; (base_{\setminus sync})^*; po_s \end{pmatrix} \cap (\mathcal{S} \times \mathcal{S})$$

where

$$fence_{\setminus sync} \triangleq fence_s \setminus (\mathcal{W} \times \mathcal{R})$$
$$hb_{\setminus sync} \triangleq ppo_s \cup fence_{\setminus sync} \cup rfe_s$$
$$base_{\setminus sync} \triangleq (fence_{\setminus sync} \cup (rfe_s; fence_{\setminus sync})); hb^*_{\setminus sync}$$

Definition 24. We define a partial order on write events such that it agrees with co_s on all write events at a location different than f and moreover we require that two write events $a, b \in A$ s.t. $loc(a) = loc(b) = f$ are ordered by co_{base} iff their corresponding sync events, a_s, b_s are ordered by B. Formally:

$co_{base} = co_s \cup \{\langle a,b\rangle \in (\mathcal{W} \times \mathcal{W}) \mid loc(a) = loc(b) = f \wedge \langle a_s, b_s\rangle \in B\}$

Lemma 17. The relation co_{base} is acyclic.

Proof. First, note that if $\langle a,b\rangle \in chapo^?; (base_{\setminus sync})^*; po_{\mathtt{S}}^{\rightarrow}$ and $\langle b,c\rangle \in po_{\mathtt{S}}^{\leftarrow}; (hb_{\setminus sync})^*$, then $\langle a,c\rangle \in prop$. Call this property (*). From the definition of co_{base}, it suffices to show that the relation B is acyclic. Notice from property (*) that a cycle in B implies a cycle in $prop$ which is acyclic from the *propagation* axiom. □

Furthermore, we define co as a linear extension of co_{base}. Finally we provide a complete and Power-coherent execution G_r by defining the relation rf_r. We require that every read on f reads from the most recent write according the order co. Formally:

$rf_r \triangleq rf_s \cup \{\langle a, b_1\rangle \mid \textit{immediate } \langle a, b_2\rangle \in co \wedge \langle b_1, b_2\rangle \in rmw \wedge loc(a) = f\}$

Remark 4. co is acyclic.

Lemma 18 (hbf-included-in-cof)**.** Let $a, b \in A$, such that $\langle a,b\rangle \in (hb^+)|_f$. Then the following hold:

1. If $a, b \in \mathcal{R}$, then there exist a', b' such that $\langle a', b'\rangle \in (co|_f)^+$, $\langle a, a'\rangle \in rmw$, and $\langle b, b'\rangle \in rmw$.
2. If $a \in \mathcal{R}$ and $b \in \mathcal{W}$, then there exists a' such that $\langle a', b\rangle \in (co|_f)^*$ and $\langle a, a'\rangle \in rmw$.
3. If $a \in \mathcal{W}$ and $b \in \mathcal{R}$, then there exists b' such that $\langle a, b'\rangle \in (co|_f)^+$ and $\langle b, b'\rangle \in rmw$.
4. If $a, b \in \mathcal{W}$, then $\langle a,b\rangle \in (co|_f)^+$.

Proof. By induction on $\langle a,b\rangle \in (hb^+)|_f$.

[Base Case] If $\langle a,b\rangle \in hb|_f$.
For cases 1,4 $\langle a,b\rangle \in hb|_f$ implies $\langle a,b\rangle \in (fence \cup ppo)|_f$ which implies that $\langle a_s, b_s\rangle \in po; po$ and thus by Def. 24 $\langle a', b'\rangle \in co|_f$. For case 2, either the above relations hold or if $\langle a,b\rangle \in rmw$ we have that $a' = b$. For case 3, there is one more possibility, in particular that $\langle a,b\rangle \in rfe$. In that case $\langle a, b'\rangle \in co|_f$ by construction.

[Inductive Case] If $\langle a,c\rangle \in (hb^+)|_f$ and $\langle c,b\rangle \in (hb^+)|_f$. All cases are direct consequence of the induction hypotheses.

□

Lemma 19 (hbf-hb-included-in-cof)**.** Let $a, b \in A$, such that

$$\langle a,b\rangle \in (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^+.$$

Then, the following hold:

1. If $a, b \in \mathcal{R}$, then there exist a', b' such that $\langle a', b'\rangle \in (co|_f)^+$, $\langle a, a'\rangle \in rmw$ and $\langle b, b'\rangle \in rmw$.
2. If $a \in \mathcal{W}$ and $b \in \mathcal{R}$, then there exists b' such that $\langle a, b'\rangle \in (co|_f)^+$ and $\langle b, b'\rangle \in rmw$.
3. If $a \in \mathcal{R}$ and $b \in \mathcal{W}$, then there exists a' such that $\langle a', b\rangle \in (co|_f)^+$ and $\langle a, a'\rangle \in rmw$.
4. If $a, b \in \mathcal{W}$, then $\langle a,b\rangle \in (co|_f)^+$.

Proof. By induction on $\langle a,b\rangle \in (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^+$.

[Base Case] We have: $\langle a,c\rangle \in hb_{fx}$, $\langle c,d\rangle \in (hb_{xy})^*$, $\langle d,e\rangle \in hb_{xf}$, and $\langle e,b\rangle \in (hb|_f)^*$. The first three imply that $\langle a_s, e_s\rangle \in po; hb^*_s; po$ and thus by Def. 24 $\langle a', e'\rangle \in co|_f$. By cases on $\langle e,b\rangle \in (hb|_f)^*$ either $e = b$ and thus $e' = b'$ or $\langle e,b\rangle \in (hb|_f)^+$) and by Lemma 18 $\langle e', b'\rangle \in (co|_f)^*$ (by weakening the result) and thus $\langle a, b'\rangle \in (co|_f)^+$, where for a', b' it holds that either $a = a'$ (resp. $b = b'$) or $\langle a, a'\rangle \in rmw$ (resp. $\langle b, b'\rangle \in rmw$) depending on which case we are trying to prove.

[Inductive Case] If $\langle a,c\rangle \in hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^+$ and $\langle c,b\rangle \in hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^+$. The result is immediate by case analysis and the induction hypotheses on the two relations above.

□

Lemma 20. Let $a, b \in A$ s.t. $loc(a) = f$ and $\langle a,b\rangle \in fre$ then it must be that if $\langle a, a'\rangle \in rmw$ we have $\langle a', b\rangle \in co$.

Proof. Suppose by contradiction that $\langle b, a' \rangle \in co$. Then we have that $\langle a, a' \rangle \in fre; co$ and $\langle a, a' \rangle \in rmw$ which contradicts the axiom *atomic* which holds by Lemma 22. □

Lemma 21. For every $x \in \mathsf{Loc}$, the relation $po|_x \cup rf \cup co \cup fr$ is acyclic.

Proof. For all locations $x \neq f$ this is a direct result of the *sc-per-loc* axiom for the execution G_s. For $x = f$ notice that it must be that $fr \subseteq po|_f \cup co$ because either $\langle a, b \rangle \in fre$ and then it follows from Lemma 20 or $\langle a, b \rangle \in fr \wedge \langle a, b \rangle \in po$. Thus it suffices to show that $po|_f \cup rf|_f \cup co|_f$ is acyclic. Notice that $co|_f$ is a total order on all writes to location f and moreover

1. $\mathtt{RR}(po|_f)$ is acyclic.
2. There are no $b \in \mathcal{W}|_f$, $c_1, c_2 \in \mathcal{R}|_f$ s.t. $\langle c_1, b \rangle \in po|_f$, $\langle b, c_2 \rangle \in rf|_f \cup po|_f$, and $\langle c_2, c_1 \rangle \in (po|_f)^*$, because that would imply a cycle in $co|_f$ by construction.
3. There are no $b_1, b_2 \in \mathcal{W}|_f$, $c_1, c_2 \in \mathcal{R}|_f$ s.t. $\langle c_1, b_1 \rangle \in po|_f$, $\langle b_2, c_2 \rangle \in po|_f \cup rf|_f$, $\langle b_1, b_2 \rangle \in co|_f$, and $\langle c_2, c_1 \rangle \in (po|_f)^*$, because that would imply a cycle in $co|_f$ by construction.

Thus by Lemma 2 $po|_f \cup rf|_f \cup co|_f$ is acyclic. □

Lemma 22. The relation $rmw \cap (fre; coe)$ is empty.

Proof. For RMWs on locations different than f it is a direct result of the axiom *atomic* of the execution G_s. For RMWs on location f, by construction all read events a_1 on f read the write that immediately precedes the write a_2 in co. Suppose $\langle a_1, b \rangle \in fre$ this implies that there exists a write c such that $\langle c, a_1 \rangle \in rfe$ and $\langle c, b \rangle \in co$. If $\langle b, a_2 \rangle \in co$ then a_1 did not read from the immediate co-preceding write which contradicts our construction. □

Lemma 23. The relation hb is acyclic.

Proof. It suffices to show that hb_{xy} and
$(hb|_f)^+; (hb_{fx}; (hb_{xy})^*; hb_{xf})^?$ are acyclic. Notice that $hb_{xy} \subseteq hb_s$ thus it must be acyclic. Now suppose that there exists a such that
$\langle a, a \rangle \in ((hb|_f)^+; (hb_{fx}; (hb_{xy})^*; hb_{xf})^?)^+$. We can consider $typ(a) = \mathtt{W}$ without loss of generality (if $typ(a) = \mathtt{R}$ then since $rmw \subseteq hb|_f \langle a', a' \rangle \in ((hb|_f)^+; (hb_{fx}; (hb_{xy})^*; hb_{xf})^?)^+$). The above implies that:
$\langle a, a \rangle \in (hb|_f)^+; (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*$
which is equivalent to

1. $\langle a, b \rangle \in (hb|_f)^+$
2. $\langle b, a \rangle \in (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*$

By case analysis on the second relation, if $a = b$ then $\langle a, a \rangle \in (hb|_f)^+$ and by Lemma 18 we have that $\langle a', a' \rangle \in (co|_f)^+$. Otherwise, if
$\langle b, a \rangle \in (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^+$
by Lemma 19 we have that we have that $\langle b, a \rangle \in (co|_f)^+$ and by Lemma 18 we have that $\langle a, b \rangle \in (co|_f)^*$ thus $\langle a, a \rangle \in (co|_f)^+$ which by Remark 4 is acyclic, contradiction. □

Lemma 24 (hb-co-xf)**.** Let $a, b \in A$ then the following hold:

1. $\langle a, b \rangle \in rfe^?; fence_{xy}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^* \implies \langle a, b' \rangle \in rfe^?; fence_{xy}; (hb_{xy})^*; po; co^*|_f \wedge (b = b' \vee \langle b, b' \rangle \in rmw)$
2. $\langle a, b \rangle \in rfe^?; hb_{xf}; ((hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf})^?)^* \implies \langle a, b' \rangle \in rfe^?; po; co^*|_f \wedge (b = b' \vee \langle b, b' \rangle \in rmw)$

Proof. Both cases are a direct consequence of Lemmas 18 and 19 and the fact that hb_{xf} is included in po. □

Lemma 25 (hb-hbf-included-in-cof)**.** Let $a, b \in A$ such that $\langle a, b \rangle \in (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^+$. Then it holds that: $\langle a, b \rangle \in hb_{xf}; (co|_f)^*; po; (hb_{xy})^*$

Proof. By induction on $\langle a, b \rangle \in (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^+$

[Base Case] $\langle a, b \rangle \in hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*$ Equivalently:
$\langle a, c \rangle \in hb_{xf}$
$\wedge\ \langle c, d \rangle \in (hb|_f)^*$
$\wedge\ \langle d, e \rangle \in hb_{fx}$
$\wedge\ \langle e, b \rangle \in (hb_{xy})^*$
Using Lemma 18 and the second relation we obtain $\langle c', d' \rangle \in hb_{xf}; (co|_f)^*; hb_{fx}; (hb_{xy})^*$ (where $c = c' \vee \langle c, c' \rangle \in rmw$ and resp. for d, d'). Notice that $\langle a, c \rangle \in hb_{xf}$ implies that $\langle a, c' \rangle \in hb_{xf}$ because there is a `lwsync` before the RMW. On the other hand, if $\langle d, b \rangle \in hb_{fx}$ then we have that $\langle d', b \rangle \in po$. Thus we conclude by transitivity.

[Inductive Case] If $\langle a, c \rangle \in (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^+$ and $\langle c, b \rangle \in (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^+$ then by the induction hypotheses we have that
$\langle a, c \rangle \in hb_{xf}; (co|_f)^*; po; (hb_{xy})^*$
$\wedge\ \langle c, b \rangle \in hb_{xf}; (co|_f)^*; po; (hb_{xy})^*$
It is easy to see that $po; (hb_{xy})^*; hb_{xf}$ is included in $po; hb^*_{\setminus sync}; po$, thus using Def. 24 we obtain
$\langle a, b \rangle \in hb_{xf}; (co|_f)^*; po; (hb_{xy})^*$.

□

Lemma 26 (hb-co-xx)**.** Let $a, b \in A$ then the following hold:

1. $\langle a, b \rangle \in rfe^?; fence_{xy}; (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^* \implies \langle a, b \rangle \in rfe^?; fence_{xy}; (hb_{xy})^*; (hb_{xf}; (co|_f)^*; po; (hb_{xy})^*)^?$
2. $\langle a, b \rangle \in rfe^?; (fence_{xf}; (hb|_f)^*; hb_{fx}; (hb|_{xy})^*)^+ \implies \langle a, b \rangle \in rfe^?; po_{xf}; (co|_f)^*; po; (hb_{xy})^*$

Proof.

1. The premise is equivalent to:
$\langle a, c \rangle \in rfe^?; fence; (hb_{xy})^*$
$\wedge\ \langle c, b \rangle \in (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*$
Using Lemma 25 we obtain that $\langle c, b \rangle \in (hb_{xf}; (co|_f)^*; po; (hb_{xy})^*)^?$ and we conclude using transitivity.
2. The premise is equivalent to:
$\langle a, c \rangle \in rfe^?$
$\wedge\ \langle c, b \rangle \in (fence_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^+$
and we conclude using Lemma 25 and transitivity.

□

Lemma 27 (hb-co-fx)**.** Let $a, b \in A$ then the following hold:

1. $\langle a, b \rangle \in rfe|_f^?; fence|_f; (hb|_f)^*; hb_{fx}; (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^* \implies \exists a'.\ \langle a', b \rangle \in (co|_f)^+; po; (hb_{xy})^* \wedge (a' = a \vee \langle a, a' \rangle \in rmw)$
2. $\langle a, b \rangle \in rfe_r|_f^?; fence_{fx}; (hb_{xy})^*; ((hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^* \implies \exists a'.\ \langle a', b \rangle \in (co|_f)^*; po; (hb_{xy})^* \wedge (a' = a \vee \langle a, a' \rangle \in rmw)$

Proof.

1. The premise is equivalent to:
   $\langle a,c\rangle \in rfe|_f^?; fence|_f; (hb|_f)^*$
   $\wedge\ \langle c,d\rangle \in (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*$
   $\wedge\ \langle d,b\rangle \in hb_{fx}; (hb_{xy})^*$
   The first relation implies that $\langle a,c\rangle \in hb^+$ and by Lemma 18 we obtain $\langle a',c'\rangle \in (co|_f)^+$ (where $a' = a \vee \langle a,a'\rangle \in rmw$ resp. for c'). The second relation implies that $\langle c',d'\rangle \in (co|_f)^*$ by Lemma 19. Finally $\langle d,b\rangle \in hb_{fx}; (hb_{xy})^*$ implies $\langle d',b\rangle \in po; (hb_{xy})^*$.
   Thus by transitivity we obtain $\langle a',b\rangle \in (co|_f)^+; po; (hb_{xy})^*$.
2. The premise is equivalent to:
   $\langle a,c\rangle \in rfe|_f^?; fence_{fx}; (hb_{xy})^*$
   $\wedge\ \langle c,b\rangle \in (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*$
   Notice that the first relation implies $\langle a',c\rangle \in (co|_f)^*; po; (hb_{xy})^*$. By Lemma 25 and the second relation we obtain $\langle c,b\rangle \in (hb_{xf}; (co|_f)^*; po; (hb_{xy})^*)^?$. If $c = b$ then we are done.
   Otherwise, we have that $\langle a',d\rangle \in rfe|_f^?; fence_{fx}; (hb_{xy})^*; po$ and $\langle d,b\rangle \in (co|_f)^*; po; (hb_{xy})^*$. This implies $\langle a_s,d_s\rangle \in rfe_s{}^?; fence; hb_s^*; po_s$ and thus $\langle a',d\rangle \in co|_f$ and we conclude by transitivity.

□

Lemma 28. Let $a,b \in A$ such that $\langle a,b\rangle \in hb^*$. The following hold:

1. $loc(a) \neq f \wedge loc(b) \neq f \implies$
   $\langle a,b\rangle \in (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx})^?)^*$
2. $loc(a) \neq f \wedge loc(b) = f \implies$
   $\langle a,b\rangle \in (hb_{xy})^*; hb_{xf}; (hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*$
3. $loc(a) = f \wedge loc(b) \neq f \implies$
   $\langle a,b\rangle \in (hb|_f)^*; hb_{fx}; (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*$
4. $loc(a) = loc(b) = f \implies$
   $\langle a,b\rangle \in ((hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf})^?)^*$

Proof. By induction on $\langle a,b\rangle \in hb^*$.

[Base Case] We have that $a = b$.

1. Trivial (goal is reflexive).
2. In this case, $loc(a) \neq f$ and $loc(b) = f$ but $a = b$. Contradiction.
3. Likewise.
4. Trivial (goal is reflexive).

[Inductive Case] Consider $\langle a,c\rangle \in hb^*$ and $\langle c,b\rangle \in hb$. By case analysis on $loc(c) = f$:

- If $loc(c) = f$.
  1. $loc(a) \neq f$ and $loc(b) \neq f$. By induction hypothesis we have that
     $\langle a,c\rangle \in (hb_{xy})^*; hb_{xf}; (hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*$
     which is equivalent to
     $\langle a,c\rangle \in (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*; hb_{xf}; (hb|_f)^*$
     by the induction hypothesis for $\langle c,b\rangle \in hb^*$ we obtain
     $\langle c,b\rangle \in (hb|_f)^*; hb_{fx}; (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*$
     which implies
     $\langle c,b\rangle \in (hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*; hb_{fx}; (hb_{xy})^*$
     thus by transitivity we have that
     $\langle a,b\rangle \in (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*; hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*$
     which reduces to
     $\langle a,b\rangle \in (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*$
  2. $loc(a) \neq f$ and $loc(b) = f$. From $\langle c,b\rangle \in hb$ we deduce that $\langle b,c\rangle \in hb|_f$. By induction hypothesis for $\langle a,c\rangle \in hb^*$ we have that
     $\langle a,c\rangle \in (hb_{xy})^*; hb_{xf}; (hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*$
     and hence $\langle a,b\rangle \in (hb_{xy})^*; hb_{xf}; (hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*; hb|_f$
     which implies
     $\langle a,c\rangle \in (hb_{xy})^*; hb_{xf}; (hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*$
  3. $loc(a) = f$ and $loc(b) \neq f$. From $\langle c,b\rangle \in hb$ we deduce $\langle c,b\rangle \in hb_{fx}$. By induction hypothesis, $\langle a,c\rangle \in ((hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf})^?)^*$ or equivalently $\langle a,c\rangle \in (hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf}); (hb|_f)^*)^*$.
     By transitivity we obtain that
     $\langle a,b\rangle \in (hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*; hb_{fx}$
     which is equivalent to $\langle a,b\rangle \in (hb|_f)^*; hb_{fx}; (hb_{xy})^*; ((hb_{xf}); (hb|_f)^*; hb_{fx}; (hb_{xy})^*;)^*$
     which concludes the case.
  4. $loc(a) = f$ and $loc(b) = f$.
     By induction hypothesis, we have that
     $\langle a,c\rangle \in ((hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf})^?)^*$. which is equivalent to $\langle a,c\rangle \in (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*$.
     Moreover by $\langle c,b\rangle \in hb$ we have that $\langle c,b\rangle \in hb|_f$ which implies that
     $\langle a,b\rangle \in (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*; hb|_f$
     which is equivalent to
     $\langle a,b\rangle \in (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*$
     which concludes the case.
- If $loc(c) \neq f$.
  1. $loc(a) \neq f$ and $loc(b) \neq f$. By induction hypothesis we have that
     $\langle a,c\rangle \in (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx})^?)^*$.
     Moreover, $\langle c,b\rangle \in hb$ implies $\langle c,b\rangle \in hb_{xy}$ and thus
     $\langle a,b\rangle \in (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*; (hb_{xy})$
     which implies
     $\langle a,b\rangle \in (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*$
  2. $loc(a) \neq f$ and $loc(b) = f$. From $\langle c,b\rangle \in hb$ we deduce that $\langle c,b\rangle \in hb_{xf}$. By induction hypothesis for $\langle a,c\rangle \in hb^*$ we have that
     $\langle a,c\rangle \in (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx})^?)^*$
     $\langle a,b\rangle \in (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*; hb_{xf}$
     which is equivalent to
     $\langle a,b\rangle \in (hb_{xy})^*; hb_{xf}; ((hb|_f)^*; hb_{fx}; (hb_{xy})^*; hb_{xf})^*$
     which is equivalent to
     $\langle a,b\rangle \in (hb_{xy})^*; hb_{xf}; (hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf})^*; (hb|_f)^*)^*$
     which concludes the case.
  3. $loc(a) = f$ and $loc(b) \neq f$. From $\langle c,b\rangle \in hb$ we deduce $\langle c,b\rangle \in hb_{xy}$. By induction hypothesis, $\langle a,c\rangle \in (hb|_f)^*; hb_{fx}; (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*$
     By transitivity we obtain that
     $\langle a,b\rangle \in (hb|_f)^*; hb_{fx}; (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*; hb_{xy}$
     which is equivalent to $\langle a,b\rangle \in (hb|_f)^*; hb_{fx}; (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^*$
     which concludes the case.
  4. $loc(a) = f$ and $loc(b) = f$.

By induction hypothesis, we have
$\langle a,c\rangle \in ((hb|_f)^*;(hb_{fx};(hb_{xy})^*;hb_{xf})^?)^*$ which is equivalent to
$\langle a,c\rangle \in (hb|_f)^*;(hb_{fx};(hb_{xy})^*;hb_{xf};(hb|_f)^*)^*$.
Moreover by $\langle c,b\rangle \in hb$ we have that $\langle c,b\rangle \in hb|_f$ which implies that
$\langle a,b\rangle \in (hb|_f)^*;(hb_{fx};(hb_{xy})^*;hb_{xf};(hb|_f)^*)^*;hb|_f$.
which is equivalent to
$\langle a,b\rangle \in (hb|_f)^*;(hb_{fx};(hb_{xy})^*;hb_{xf};(hb|_f)^*)^*$.
which concludes the case.

□

Lemma 29 (prop-base-analysis)**.** Let $a,b \in A$ such that $\langle a,b\rangle \in prop_r$, then it holds that:

1. $loc(a)\neq f \wedge loc(b)\neq f \implies$
$\langle a,b\rangle \in rfe^?;(po_{xf};(hb|_f)^*;hb_{fx};(hb_{xy})^*)^+$
$\vee\ \langle a,b\rangle \in rfe^?;fence;((hb_{xy})^*;(hb_{xf};(hb|_f)^*;hb_{fx})^?)^*$
2. $loc(a)\neq f \wedge loc(b)= f \implies$
$\langle a,b\rangle \in rfe^?;fence;(hb_{xy})^*;hb_{xf};(hb|_f)^*;$
$(hb_{fx};(hb_{xy})^*;hb_{xf};(hb|_f)^*)^*$
$\vee\ \langle a,b\rangle \in rfe^?;po_{xf};((hb|_f)^*;(hb_{fx};(hb_{xy})^*;hb_{xf})^?)^*$
3. $loc(a)= f \wedge loc(b)\neq f \implies$
$\langle a,b\rangle \in rfe|_f^?;fence|_f;(hb|_f)^*;hb_{fx};(hb_{xy})^*;$
$(hb_{xf};(hb|_f)^*;hb_{fx};(hb_{xy})^*)^*$
$\vee\ \langle a,b\rangle \in rfe|_f^?;fence_{fx};((hb_{xy})^*;$
$(hb_{xf};(hb|_f)^*;hb_{fx})^?)^*$
4. $loc(a)= f \wedge loc(b)= f \implies$
$\langle a,b\rangle \in rfe|_f^?;fence|_f;((hb|_f)^*;$
$(hb_{fx};(hb_{xy})^*;hb_{xf};)^?)^*$
$\vee\ \langle a,b\rangle \in rfe|_f^?;fence_{fx};(hb_{xy})^*;hb_{xf};(hb|_f)^*;$
$(hb_{fx};(hb_{xy})^*;hb_{xf};(hb|_f)^*)^*$

Proof. Notice that $\langle a,b\rangle \in prop$ is equivalent to:
$\langle a,c\rangle \in rfe^? \wedge \langle c,d\rangle \in fence \wedge \langle d,b\rangle \in hb_r^*$
$\wedge typ(a)=\mathtt{W} \wedge typ(b)=\mathtt{W}$
We present below the first case, the rest are similar:
If $loc(a)\neq f$ and $loc(b)\neq f$ then we have two cases:

- If $\langle c,d\rangle \in fence_{xy}$ then $loc(d)\neq f$ and by Lemma 28 we have that $\langle d,b\rangle \in ((hb_{xy})^*;(hb_{xf};(hb|_f)^*;hb_{fx})^?)^*$ and we conclude by transitivity.
- If $\langle c,d\rangle \in fence_{xf}$ then using Lemma 28 and transitivity we obtain $\langle c,b\rangle \in (hb_{xf};(hb|_f)^*;hb_{fx};(hb_{xy})^*)^+$ and the we conclude by transitivity.

□

Lemma 30. Let $a,b \in A$ such that $\langle a,b\rangle \in (prop_r \cup po\text{-}aa_{xy} \cup co)^+$. We can deduce that:

1. $loc(a)\neq f \wedge loc(b)\neq f \implies$
$\langle a,b\rangle \in (prop_{xy}\cup po\text{-}aa_{xy}\cup co_{xy})^*;$
$(rfe^?;po_{xf};(co|_f)^*;po_{fx};(hb_{xy})^*;$
$(prop_{xy}\cup po\text{-}aa_{xy}\cup co_{xy})^*)^?$
$\vee\ \langle a,b\rangle \in (prop_{xy}\cup po\text{-}aa\cup co_{xy})^*;$
$(rfe^?;fence;(hb_{xy})^*;po_{xf};(co|_f)^*;po_{fx};(hb_{xy})^*;$
$(prop_{xy}\cup po\text{-}aa_{xy}\cup co_{xy})^*)^?$
2. $loc(a)\neq f \wedge loc(b)= f \implies$
$\langle a,b\rangle \in (prop_{xy}\cup po\text{-}aa_{xy}\cup co_{xy})^*;$
$rfe^?;fence;(hb_{xy})^*;po_{xf};(co|_f)^*$
$\vee\ \langle a,b\rangle \in (prop_{xy}\cup po\text{-}aa_{xy}\cup co_{xy})^*;$
$rfe^?;po_{xf};(co|_f)^*$
3. $loc(a)= f \wedge loc(b)\neq f \implies$
$\langle a,b\rangle \in (co|_f)^*;po;(hb_{xy})^*;(co_{xy}\cup po\text{-}aa_{xy}\cup prop_{xy})^*$
4. $loc(a)= f \wedge loc(b)= f \implies \langle a,b\rangle \in (co|_f)^+$

Proof. By induction on $\langle a,b\rangle \in (prop_r \cup po\text{-}aa \cup co)^+$.

[Base Case] $\langle a,b\rangle \in (prop_r \cup po\text{-}aa \cup co)^+$.
- If $\langle a,b\rangle \in co$ then: Cases 1 and 4 are trivial, while for case 2 and 3 we have a contradiction ($loc(a)\neq loc(b)$).
- If $\langle a,b\rangle \in po\text{-}aa$ then: Cases 1-3 are trivial. For case 4, notice that $\langle a,b\rangle \in po\text{-}aa$ implies $\langle a,b\rangle \in po;po$ which by Def. 24 implies $\langle a,b\rangle \in co$.
- If $\langle a,b\rangle \in prop_r$ then using Lemma 29:
  1. In both cases, we conclude using Lemma 26.
  2. In both cases, we conclude using Lemma 24 and the fact that $typ(b)=\mathtt{W}$.
  3. Likewise, we conclude using Lemma 27 and the fact that $typ(a)=\mathtt{W}$.
  4. We distinguish the two cases given by Lemma 29:
     - ◇ The first case can be written in equivalent way as: $\langle a,b\rangle \in (hb|_f)^+;(hb_{fx};(hb_{xy})^*;hb_{xf};(hb|_f)^*)^*$ which by Lemmas 18 and 19 implies that $\langle a,b\rangle \in co|_f^+$ since $typ(a)=typ(b)=\mathtt{W}$.
     - ◇ The second case can be written in an equivalent way as: $\langle a,b\rangle \in rfe|_f^?;fence_{fx};(hb_{xy})^*;$ $(hb_{xf};(hb|_f)^*;hb_{fx};(hb_{xy})^*)^*;hb_{xf};(hb|_f)^*$ By Lemma 27 and the fact that $typ(a)=\mathtt{W}$ we have that $\langle a,b\rangle \in (co|_f)^*;po;(hb_{xy})^*;hb_{xf};(hb|_f)^*$ from which we deduce:
     (a) $\langle a,c\rangle \in (co|_f)^*$
     (b) $\langle c,d\rangle \in po;(hb_{xy})^*;po$
     (c) $\langle d,b\rangle \in (hb|_f)^*$
     Consider the execution G_s and the fence events c_s,d_s, since $hb_{xy} \subseteq hb_{\setminus sync}$ we have by Def. 24 that $\langle c,d\rangle \in co|_f$. We consider $typ(d)=\mathtt{W}$ without loss of generality (note that if $typ(d)=\mathtt{R}$ the same relations hold for d' s.t. $\langle d,d'\rangle \in rmw$). Using Lemma 18 and the last relation we obtain $\langle d,b\rangle \in (co|_f)^*$ and thus we conclude that $\langle a,b\rangle \in (co|_f)^+$.

[Inductive Case] $\langle a,c\rangle \in (prop_r \cup po\text{-}aa \cup co)^+$ and $\langle c,b\rangle \in (prop_r \cup po\text{-}aa \cup co)^+$. Case analysis on $loc(c)=f$:
- $loc(c)=f$
For the first case we have by induction hypotheses that:
$\langle a,c\rangle \in (prop_{xy}\cup po\text{-}aa_{xy}\cup co_{xy})^*;$
$rfe^?;fence;hb^*_{xy};po_{xf};(co|_f)^*$
$\vee\ \langle a,c\rangle \in (prop_{xy}\cup po\text{-}aa_{xy}\cup co_{xy})^*;$
$rfe^?;po_{xf};(co|_f)^*$
and that $\langle c,b\rangle \in (co|_f)^*;po;hb^*_{xy};(co_{xy}\cup po\text{-}aa_{xy}\cup prop_{xy})^*$. In both cases, the conclusion is trivial. Likewise for cases 2,3,4 the goal is a direct consequence of the induction hypotheses.
- $loc(c)\neq f$
All the cases are solved by a similar argument. We write down some cases of the first goal below and claim that the rest are similar.
  1. $loc(a)\neq f$, $loc(b)\neq f$, $loc(c)\neq f$. By induction hypotheses we have that
  $\langle a,c\rangle \in (prop_{xy}\cup po\text{-}aa_{xy}\cup co_{xy})^*;$
  $(rfe^?;po_{xf};(co|_f)^*;po_{fx};(hb_{xy})^*;$
  $(prop_{xy}\cup po\text{-}aa_{xy}\cup co_{xy})^*)^?$

$\vee\ \langle a,c\rangle \in (prop_{xy} \cup po\text{-}aa \cup co_{xy})^*;$
$(rfe^?; fence; hb^*_{xy}; po_{xf}; (co|_f)^*; po_{fx}; (hb_{xy})^*;$
$(prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*)^?$

and that

$\langle b,c\rangle \in (prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*;$
$(rfe^?; po_{xf}; (co|_f)^*; po_{fx}; (hb_{xy})^*;$
$(prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*)^?$
$\vee\ \langle b,c\rangle \in (prop_{xy} \cup po\text{-}aa \cup co_{xy})^*;$
$(rfe^?; fence; hb^*_{xy}; po_{xf}; (co|_f)^*; po_{fx}; (hb_{xy})^*;$
$(prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*)^?$

We proceed by case analysis on the above relations. Most cases are trivial by transitivity of the relations, the rest are similar with each other. We will outline some of them below.

- $\diamond$ If $\langle a,c\rangle \in (prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*;$
$rfe^?; po_{xf}; (co|_f)^*; po_{fx}; (hb_{xy})^*;$
$(prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*$
then by cases on the relation between b and c:
  - $\circ$ If $\langle b,c\rangle \in (prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*;$
$rfe^?; po_{xf}; (co|_f)^*; po_{fx}; (hb_{xy})^*;$
$(prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*$
then we have that:
    (a) $\langle a,d\rangle \in (prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*;$
$rfe^?; po; (co|_f)^*$
    (b) $\langle d,e\rangle \in po; (hb_{xy})^*; (prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*; rfe^?; po$
    (c) $\langle e,b\rangle \in (co|_f)^*; po; (hb_{xy})^*;$
$(prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*$
Notice that $(prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*$ is included in $(co_s \cup po\text{-}aa \cup \mathtt{WW}(base_{\setminus sync}))^*$. By Def. 24 we have that $\langle d,e\rangle \in co|_f$ and transitivity concludes the proof.
  - $\circ$ If $\langle b,c\rangle \in (prop_{xy} \cup po\text{-}aa \cup co_{xy})^*;$
$rfe^?; fence; hb^*_{xy}; po_{xf}; (co|_f)^*; po_{fx}; (hb_{xy})^*;$
$(prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*$
then we have that:
    (a) $\langle a,d\rangle \in (prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*;$
$rfe^?; po; (co|_f)^*$
    (b) $\langle d,e\rangle \in po; (hb_{xy})^*; (prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*; rfe^?; fence; (hb_{xy})^*; po$
    (c) $\langle e,b\rangle \in fence; (co|_f)^*; po; (hb_{xy})^*;$
$(prop_{xy} \cup po\text{-}aa_{xy} \cup co_{xy})^*$
Similar to the previous case we have that, $\langle d,e\rangle \in co|_f$ and we conclude the prove by transitivity.
- $\diamond$ Like the previous case.

□

Lemma 31. The relation $prop_r \cup po\text{-}aa \cup co$ is acyclic.

Proof. Suppose by contradiction that for some $a \in A_r$ we have $\langle a,a\rangle \in (prop_r \cup po\text{-}aa \cup co)^+$. We distinguish two cases:

- If $loc(a) = f$ then by Lemma 30 we have that $\langle a,a\rangle \in (co|_f)^+$ which contradicts Remark 4.
- If $loc(a) \neq f$ then by Lemma 30 we have two cases:
  - $\diamond$ $\langle a,a\rangle \in (prop_{xy} \cup po\text{-}aa \cup co_{xy})^*; (rfe^?; po_{xf}; (co|_f)^*; po_{fx}; hb^*_{xy}; (prop_{xy} \cup co_{xy})^*)^?.$
If $\langle a,a\rangle \in (prop_{xy} \cup co_{xy})^+$ then obviously there is a cycle in $prop_s \cup po\text{-}aa \cup co_s$ which contradicts the *propagation* axiom. Otherwise we have that there exists $c,d \in A$ such that $\langle a,c\rangle \in (prop_{xy} \cup po\text{-}aa \cup co_{xy})^*; rfe^?; po_{xf}, \langle c,d\rangle \in (co|_f)^*$, and $\langle d,a\rangle \in po_{fx}; hb^*_{xy}; (prop_{xy} \cup po\text{-}aa \cup co_{xy})^*$). The above imply that $\langle d,c\rangle \in po_{fx}; hb^*_{xy}; (prop_{xy} \cup po\text{-}aa \cup co_{xy})^*; rfe^?; po_{xf}$ and thus according to Def. 24 $\langle d,c\rangle \in co_{base}$ and hence $\langle c,c\rangle \in (co|_f)^+$, which contradicts Remark 4.
  - $\diamond$ $\langle a,b\rangle \in (prop_{xy} \cup po\text{-}aa \cup co_{xy})^*;$
$(rfe^?; fence; hb^*_{xy}; po_{xf}; (co|_f)^*; po_{fx}; hb^*_{xy};$
$(prop_{xy} \cup po\text{-}aa \cup co_{xy})^*)^?.$
This case proceeds exactly like the previous one.

□

Lemma 32. The relation $fre_r; prop_r; hb^*_r$ is irreflexive.

Proof. Suppose by contradiction that there exists $a \in A$ such that $\langle a,a\rangle \in fre; prop; hb^*$ or equivalently

1. $\langle a,b\rangle \in fre$
2. $\langle b,c\rangle \in prop$
3. $\langle c,a\rangle \in hb^*$

Notice that if $loc(a) = f$ then by Lemma 20 it must be that $\langle a',b\rangle \in co$. Moreover, $\langle c,a\rangle \in hb^*$ implies that $\langle c,a'\rangle \in hb^*$ and $\langle b,c\rangle \in prop$ implies $\langle b,c\rangle \in \mathtt{WW}(base)$, hence $\langle b,a'\rangle \in \mathtt{WW}(base)$ which implies that $\langle b,a'\rangle \in prop$ and thus we have a cycle in $co \cup prop$. Contradiction by Lemma 31. Therefore it must be that $loc(a) \neq f$.

Case analysis on $loc(c) = f$

- $loc(c) = f$ By applying Lemmas 24 and 29 on $\langle b,c\rangle \in prop$ we obtain:
$\langle b,c\rangle \in rfe^?; fence; (hb_{xy})^*; po; (co|_f)^*$
$\vee\ \langle b,c\rangle \in rfe^?; po; (co|_f)^*$
or in more compact form
$\langle b,c\rangle \in rfe^?; (fence; (hb_{xy})^*)^?; po; (co|_f)^*$ By applying Lemmas 19 and 28 on $\langle c,a\rangle \in hb^*$ we obtain
$\langle c,a\rangle \in (hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*;$
$hb_{fx}; (hb_{xy})^*$
or equivalently
  1. $\langle c,d\rangle \in (hb|_f)^*; (hb_{fx}; (hb_{xy})^*; hb_{xf}; (hb|_f)^*)^*$
  2. $\langle d,a\rangle \in hb_{fx}; (hb_{xy})^*$

  Then using Lemmas 18 and 19 we have that $\langle c,d'\rangle \in (co|_f)^*$ (where $d' = d$ or $\langle d,d'\rangle \in rmw$) and thus we have $\langle c,a\rangle \in (co|_f)^*; po; (hb_{xy})^*$.
  By transitivity we deduce that
  1. $\langle b,e\rangle \in rfe^?; (fence; (hb_{xy})^*)^?; po$
  2. $\langle e,g\rangle \in (co|_f)^*$
  3. $\langle g,b\rangle \in po; (hb_{xy})^*; fre$

  Thus $\langle g,e\rangle \in po; (hb_{xy})^*; fre; rfe^?; (fence; (hb_{xy})^*)^?; po$ which implies that
  $\langle g_s,e_s\rangle \in po_s; hb^*_s; fre_s; rfe_s{}^?; (fence; hb^*_s)^?; po_s$ and thus $\langle g,e\rangle \in co|_f$ and thus $\langle g,g\rangle \in (co|_f)^+$. Contradiction by Remark 4.
- $loc(c) \neq f$ By applying Lemmas 26 and 29 on $\langle b,c\rangle \in prop$ we obtain:
$\langle b,c\rangle \in rfe^?; fence; (hb_{xy})^*; (hb_{xf}; (co|_f)^*; po; (hb_{xy})^*)^?$
$\vee\ \langle b,c\rangle \in rfe^?; po; (co|_f)^*; po; (hb_{xy})^*$
By applying Lemma 28 on $\langle c,a\rangle \in hb^*$ we obtain
$\langle c,a\rangle \in ((hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx})^?)^*$
Moreover since $typ(c) = \mathtt{W}$, $typ(a) = \mathtt{R}$ it must be that
$\langle c,a\rangle \in (hb_{xy})^+$

$\vee$ $\langle c,a\rangle \in (hb_{xy})^*; (hb_{xf}; (hb|_f)^*; hb_{fx}; (hb_{xy})^*)^+$
the latter implies
$\langle c,a\rangle \in (hb_{xy})^*; hb_{xf}; (co|_f)^*; po; (hb_{xy})^*$ by Lemma 18.

We proceed by case analysis on the above relations and distinguish the following cases:

- $\diamond$ If $\langle b,c\rangle \in rfe^?; fence; (hb_{xy})^*$ then
  - If $\langle c,a\rangle \in (hb_{xy})^+$ then $\langle a,a\rangle \in fre; rfe^?; fence; (hb_{xy})^*$ which implies that $\langle a,a\rangle \in fre; rfe^?; fence; hb_s^*$. But this contradicts the *observation* axiom.
  - If $\langle c,a\rangle \in (hb_{xy})^*; hb_{xf}; (co|_f)^*; po; (hb_{xy})^*$ then
    1. $\langle b,d\rangle \in rfe^?; fence; (hb_{xy})^*; po$
    2. $\langle d,e\rangle \in (co|_f)^*$
    3. $\langle e,b\rangle \in po; (hb_{xy})^*; fre$

    which implies $\langle e,d\rangle \in co|_f$ by definition Def. 24 and thus $\langle e,e\rangle \in (co|_f)^+$, contradiction by Lemma 17.
- $\diamond$ If $\langle b,c\rangle \in rfe^?; fence; (hb_{xy})^*; hb_{xf}; (co|_f)^*; po; (hb_{xy})^*$ then
  - If $\langle c,a\rangle \in (hb_{xy})^+$ then using the inclusion hb_{xy} in hb_s, *rfe* in rfe_s, and *fre* in fre_s and by Def. 24 we obtain a cycle in $co|_f$ which contradicts Lemma 17.
  - Likewise.
- $\diamond$ If $\langle b,c\rangle \in rfe^?; po; (co|_f)^*; po; (hb_{xy})^*$ then
  - If $\langle c,a\rangle \in (hb_{xy})^+$ which implies that $\langle b,b\rangle \in rfe^?; po; (co|_f)^*; po; (hb_{xy})^*; fre$ or equivalently
    1. $\langle b,d\rangle \in rfe^?; po$
    2. $\langle d,e\rangle \in (co|_f)^*$
    3. $\langle e,b\rangle \in po; (hb_{xy})^*; fre$

    Thus $\langle e_s,d_s\rangle \in po_s; hb_s^*; fre_s; {rfe_s}^?; po_s$ which by Def. 24 implies that $\langle e,d\rangle \in co|_f$. But then we have a cycle in $co|_f$ which contradicts Lemma 17.
  - If $\langle c,a\rangle \in ((hb_{xy})^*; (hb_{xf}; (hb|_f)^*; {hb_{fx}}^?)^*$ the same argument as before applies.

□

Theorem 10. G_r is Power-coherent.

Proof. The result is a direct consequence of Lemmas 21 to 23, 31 and 32. □

H.3 Proofs for relating SRA and Power executions

Let $G = \langle A, po, rf\rangle$ and $G_p = \langle A_p, po_p, deps, rf_p, At\rangle$ be SRA and Power executions such that $G \sim G_p$. Further, let $w : (G.\mathtt{W} \cup G.\mathtt{U}) \to G_p.\mathtt{W}$, $f_w : (G.\mathtt{W} \cup G.\mathtt{U}) \to G_p.\mathtt{lwsync}$, $r : (G.\mathtt{R} \cup G.\mathtt{U}) \to G_p.\mathtt{R}$, $f_r : (G.\mathtt{R} \cup G.\mathtt{U}) \to G_p.\mathtt{isync}$, $g : A_p \to A$ be the functions from the $\sim$ definition.

From the definition of $\sim$, note that G_p contains no `sync` events. Therefore,

$$sync = \emptyset \qquad prop = base \cap (\mathcal{W} \times \mathcal{W})$$

Proof of Prop. 9.

1. $ppo = po_p \cap (\mathcal{R} \times (\mathcal{R} \cup \mathcal{W}))$.
   By definition, we have $ppo = (deps \cup isync) \cap (\mathcal{R} \times (\mathcal{R} \cup \mathcal{W}))$, which is included in $po_p \cap (\mathcal{R} \times (\mathcal{R} \cup \mathcal{W}))$.
   So it remains to show that $po_p \cap (\mathcal{R} \times \mathcal{A}) \subseteq (deps \cup isync)$.
   Consider $\langle a,b\rangle \in po_p$ such that $a \in \mathcal{R}$ and $b \in \mathcal{A}$.
   By Def. 20, we have two cases:
   - $\langle g(a), g(b)\rangle \in po$ and $g(a) \in G.\mathtt{R} \cup G.\mathtt{U}$. But then, $\langle a, f_r(g(a))\rangle \in deps$ and $\langle f_r(g(a)), b\rangle \in po_p$, and hence $\langle a,b\rangle \in isync$.
   - There exists $u \in G.\mathtt{U}$ such that $a = r(u)$ and $b = w(r)$. Then, by Def. 20, we have $\langle a,b\rangle \in deps$.
2. $fence = \begin{pmatrix}((po_p \setminus rmw) \cap ((\mathcal{R} \cup \mathcal{W}) \times \mathcal{W}));\\ (po_p \cap (\mathcal{W} \times (\mathcal{R} \cup \mathcal{W})))^?\end{pmatrix}$.
   By definition and since $sync = \emptyset$, we have $fence = lwsync \cap ((\mathcal{R} \times (\mathcal{R} \cup \mathcal{W})) \cup (\mathcal{W} \times \mathcal{W}))$.
   - Let $\langle a,b\rangle, \langle b,c\rangle \in po_p$ such that $typ(b) = \mathtt{lwsync}$ and $\langle a,c\rangle \in ((\mathcal{R} \times (\mathcal{R} \cup \mathcal{W})) \cup (\mathcal{W} \times \mathcal{W}))$.
     Hence, by Def. 20, $\langle g(a), g(b)\rangle \in po$ and $\langle g(b), g(c)\rangle \in po$ with $g(b) \in G.\mathtt{W} \cup G.\mathtt{U}$. Therefore, $\langle b, w(g(b))\rangle \in po_p$ and $\langle w(g(b)), c\rangle \in po_p$. and since po_p is transitive, also $\langle a, w(g(b))\rangle \in po_p$. Moreover, note that $\langle a, w(g(b))\rangle \notin rmw$, and therefore
     $\langle a,c\rangle \in \begin{pmatrix}((po_p \setminus rmw) \cap ((\mathcal{R} \cup \mathcal{W}) \times \mathcal{W}));\\ (po_p \cap (\mathcal{W} \times (\mathcal{R} \cup \mathcal{W})))^?\end{pmatrix}$.
   - Let $\langle a,b\rangle \in po_p \setminus rmw$ such that $a \in \mathcal{R} \cup \mathcal{W}$ and $b \in \mathcal{W}$.
     By Def. 20, we have two cases:
     - $\langle g(a), g(b)\rangle \in po$ and $g(b) \in G.\mathtt{W} \cup G.\mathtt{U}$. Then, $\langle a, f_w(g(b))\rangle \in po_p$ and $\langle f_w(g(b)), g(b)\rangle \in po_p$, and hence $\langle a,b\rangle \in lwsync$.
     - There exists $u \in G.\mathtt{U}$ such that $a = r(u)$ and $b = w(r)$. But then, $\langle a,b\rangle \in rmw$. Contradiction.
   - Let $\langle a,b\rangle \in po_p \setminus rmw$ and $\langle b,c\rangle \in po_p$ such that $a, c \in \mathcal{R} \cup \mathcal{W}$ and $b \in \mathcal{W}$.
     By Def. 20, we have two cases:
     - $\langle g(a), g(b)\rangle \in po$ and $g(b) \in G.\mathtt{W} \cup G.\mathtt{U}$. Then, $\langle a, f_w(g(b))\rangle \in po_p$ and $\langle f_w(g(b)), g(b)\rangle \in po_p$, and hence $\langle a,c\rangle \in lwsync$.
     - There exists $u \in G.\mathtt{U}$ such that $a = r(u)$ and $b = w(r)$. But then, $\langle a,b\rangle \in rmw$. Contradiction. □

H.3.1 Soundness: Power-Coherence implies SRA-Coherence

Lemma 33.
$((po_p \cap \mathcal{A} \times \mathcal{A}) \cup rf_p)^+ \subseteq \begin{pmatrix}base \cup (po_p \cap (\mathcal{A} \cup \mathcal{R})) \cup\\ ((rfe; ppo^?) \cap (\mathcal{W} \times \mathcal{R}))\end{pmatrix}$.

Proof. Let K be the relation on the r.h.s. We proceed by induction on $(po \cup rf)^+$:

[Base case 1] $\langle a,b\rangle \in po_p$. If $b \in \mathcal{R}$, then it is trivial. Otherwise, $b \in \mathcal{W}$ and $\langle a,b\rangle \in fence \subseteq base$.

[Base case 2] $\langle a,b\rangle \in rfe$. Trivial.

[Inductive case] $\langle a,b\rangle, \langle b,c\rangle \in K$. We conclude by case analysis noting that $base; base \subseteq base$, $base; po \cap (\mathcal{A} \cup \mathcal{R}) \subseteq base$, $base; hb^* \subseteq base$, $po; fence \subseteq fence$, $(ppo \cap (\mathcal{A} \times \mathcal{R})); base \subseteq base$, $rfe; base \subseteq base$, and $ppo; po \subseteq ppo$. □

Lemma 34. For all $a_p, b_p \in G_p.\mathtt{R} \cup G_p.\mathtt{W}$, if $\langle g(a_p), g(b_p)\rangle \in (po \cup rf)^+$, then $\langle a_p, b_p\rangle \in ((po_p \cap \mathcal{A} \times \mathcal{A}) \cup rf_p)^+$.

Proof. By induction on $\langle g(a_p), g(b_p)\rangle \in (po \cup rf)^+$:

[Base Case 1] $\langle g(a_p), g(b_p)\rangle \in po$. From the $\sim$ definition, we get $\langle a_p, b_p\rangle \in po_p$ and hence $\langle a_p, b_p\rangle \in ((po_p \cap \mathcal{A} \times \mathcal{A}) \cup rf_p)^+$.

[Base Case 2] $\langle g(a_p), g(b_p)\rangle \in rf$. From the $\sim$ definition, we get $\langle w(g(a_p)), r(g(b_p))\rangle \in rf_p$ and hence $\langle a_p, b_p\rangle \in {po_p}^?; rf_p; {po_p}^?$. Therefore, $\langle a_p, b_p\rangle \in ((po_p \cap \mathcal{A} \times \mathcal{A}) \cup rf_p)^+$.

[Inductive case] $\langle g(a_p), c\rangle \in (po \cup rf)^+ \wedge \langle c, g(b_p)\rangle \in (po \cup rf)^+$. Note that there exists $c_p \in (A_p \cap \mathcal{A})$ such that $c = g(c_p)$. (Let $c_p = r(c)$ if $c \in \mathcal{W}$, and $c_p = w(c)$ otherwise.) From the induction hypotheses, we get $\langle a_p, c_p\rangle \in ((po_p \cap \mathcal{A} \times \mathcal{A}) \cup rf_p)^+$ and $\langle c_p, b_p\rangle \in ((po_p \cap \mathcal{A} \times \mathcal{A}) \cup rf_p)^+$, and hence, $\langle a_p, b_p\rangle \in ((po_p \cap \mathcal{A} \times \mathcal{A}) \cup rf_p)^+$. □

Proof of Thm. 8, (⇒) direction. Let $G_p = \langle A_p, po_p, deps, rf_p, At\rangle$ be a complete Power-coherent execution and $G \sim G_p$. It suffices to show that there exists a *modification order* in G. We define mo as any strict total order extension of:

$$\{\langle w^{-1}(a_p), w^{-1}(b_p)\rangle \mid \langle a_p, b_p\rangle \in (co \cup prop)^+\}$$

on $(\mathcal{W} \cup \mathcal{U}) \times (\mathcal{W} \cup \mathcal{U})$. By Prop. 7 it suffices to show that:

1. By construction, mo is a strict total order on $\mathcal{W} \cup \mathcal{U}$.
2. Let $a, b \in A$ such that $\langle a, b\rangle \in mo$ and $\langle b, a\rangle \in (po \cup rf)^+$. Then by definition of mo, we know that $\langle w(b), w(a)\rangle \notin (co \cup prop)^+$. Since $b = g(w(b))$ and $a = g(w(a))$, by applying Lemma 34 to $\langle b, a\rangle \in (po \cup rf)^+$, we obtain $\langle w(b), w(a)\rangle \in ((po_p \cap \mathcal{A} \times \mathcal{A}) \cup rf_p)^+$. By Lemma 33, $\langle w(b), w(a)\rangle \in base$, and as $base \subseteq (co \cup prop)^+$, we get a contradiction.
3. We assume $\langle a, c\rangle \in rf$, $\langle b, c\rangle \in (po \cup rf)^+ \cup mo$, $loc(a) = loc(b)$, and $\langle a, b\rangle \in mo$, and try to derive a contradiction.
   From the definition of mo, we know that $\langle w(b), w(a)\rangle \notin (co \cup prop)^+$ and since $loc(w(a)) = loc(a) = loc(b) = loc(w(b))$, we have $\langle w(a), w(b)\rangle \in co$.
   Moreover, we have $\langle w(a), r(c)\rangle \in rf_p$, and hence $\langle r(c), w(b)\rangle \in fr_p$. Case analysis on $\langle b, c\rangle \in (po \cup rf)^+ \cup mo$.
   - $\langle b, c\rangle \in (po \cup rf)^+$. From Lemma 34, $\langle w(b), r(c)\rangle \in ((po_p \cap \mathcal{A} \times \mathcal{A}) \cup rf_p)^+$. Then either $\langle w(b), r(c)\rangle \in po_p$ which contradicts *SC-per-loc* or $\langle w(b), r(c)\rangle \notin po_p$ and hence $\langle w(b), r(c)\rangle \in base$ by Lemma 33, contradicting *Observation*.
   - $\langle b, c\rangle \in mo$. Then, $c \in \mathcal{U}$ and hence $\langle r(c), w(c)\rangle \in rmw$. From the definition of mo, we know that $\langle w(c), w(b)\rangle \notin (co \cup prop)^+$, and since $loc(w(c)) = loc(c) = loc(a) = loc(b) = loc(w(b))$, we know $\langle w(b), w(c)\rangle \in co$, which contradicts the *Atomicity* axiom. □

H.3.2 Completeness: SRA-Coherence implies Power-Coherence

Assuming a complete SRA-coherent execution $G = \langle A, po, rf\rangle$ and its matching Power execution $G_p = \langle A_p, po_p, deps, rf_p, At\rangle$ (formally $G \sim G_p$) we prove that G_p is a complete Power-coherent execution, i.e. there exists a coherence order co such that the Power execution satisfies the axioms of the model defined by Def. 19. This implication, which does not hold for the standard RA model, along with the other direction establishes that Power and SRA are equivalent (for the image of the compilation to Power).

We pick as coherence order in G_p the projection to the corresponding Power events, of SRA events that are at the same location and related by mo. Formally:

$$co \triangleq \{\langle w(a), w(b)\rangle \mid \langle a, b\rangle \in mo \wedge loc(a) = loc(b)\}$$

Lemma 35. Let $g : A_p \to A$ be the mapping from Power events to SRA events in the $\sim$ relation. Then,

1. $\langle a, b\rangle \in co \implies \langle g(a), g(b)\rangle \in mo$
2. $\langle a, b\rangle \in fr_p \setminus rmw \implies \langle g(a), g(b)\rangle \in fr$
3. $\langle a, b\rangle \in hb^+ \setminus rmw \implies \langle g(a), g(b)\rangle \in (po \cup rf)^+$.
4. $\langle a, b\rangle \in prop \implies \langle g(a), g(b)\rangle \in (po \cup rf)^+$.

Proof.

1. Immediate by the definition of co.
2. Follows from (1) and Def. 20.
3. By induction on $\langle a, b\rangle \in hb^+$:
   [Base Case] $\langle a, b\rangle \in hb \setminus rmw$. Follows directly from Def. 20.
   [Inductive case] $\langle a, c\rangle \in hb^+ \wedge \langle c, b\rangle \in hb^+$. We do a case analysis:
   - $\langle a, c\rangle \in rmw \wedge \langle c, b\rangle \in rmw$. Cannot happen.
   - $\langle a, c\rangle \in rmw \wedge \neg\langle c, b\rangle \in rmw$. Then $g(a) = g(c)$ and $\langle g(c), g(b)\rangle \in (po \cup rf)^+$.
   - $\neg\langle a, c\rangle \in rmw \wedge \langle c, b\rangle \in rmw$. Then $\langle g(a), g(c)\rangle \in (po \cup rf)^+$ and $g(c) = g(b)$.
   - $\neg\langle a, c\rangle \in rmw \wedge \neg\langle c, b\rangle \in rmw$. Then $\langle g(a), g(c)\rangle \in (po \cup rf)^+$ and $\langle g(c), g(b)\rangle \in (po \cup rf)^+$, and thus $\langle g(a), g(b)\rangle \in (po \cup rf)^+$.
4. Follows from (3) because $prop \subseteq hb^+ \cap (\mathcal{W} \times \mathcal{W}) \subseteq hb^+ \setminus rmw$. □

Lemma 36. If $\langle a_p, b_p\rangle \in ((po_p \cup rf_p \cup fr_p \cup co)|_x)^+ \setminus rmw$, then $\langle g(a_p), g(b_p)\rangle \in ((po \cup rf \cup fr \cup mo)|_x)^+$.

Proof. By induction on $\langle a_p, b_p\rangle \in ((po_p \cup rf_p \cup fr_p \cup co)|_x)^+$.

[Base Case] Immediate from Def. 20 and Lemma 35.

[Inductive Case] $\langle a_p, c_p\rangle, \langle c_p, b_p\rangle \in ((po_p \cup rf_p \cup fr_p \cup co)|_x)^+$. Case analysis on $\langle a_p, c_p\rangle \in rmw$:

- If $\langle a_p, c_p\rangle \in rmw$. Then $g(a_p) = g(c_p)$ and thus we conclude by the induction hypothesis.
- If $\langle a_p, c_p\rangle \notin rmw$. Then, by induction hypothesis we obtain $\langle g(a_p), g(c_p)\rangle \in ((po \cup rf \cup fr \cup mo)|_x)^+$. Case analysis on $\langle c_p, b_p\rangle \in rmw$:
  - ◇ If $\langle c_p, b_p\rangle \in rmw$ then $g(c_p) = g(b_p)$.
  - ◇ If $\langle c_p, b_p\rangle \notin rmw$ then by the induction hypothesis we get $\langle g(c_p), g(b_p)\rangle \in ((po \cup rf \cup fr \cup mo)|_x)^+$ and thus $\langle g(a_p), g(b_p)\rangle \in ((po \cup rf \cup fr \cup mo)|_x)^+$. □

Proof of Thm. 8, (⇐) direction. Let $G = \langle A, po, rf\rangle$ be a complete SRA-coherent execution and $G \sim G_p$. From the $\sim$ definition, it follows that G_p is complete. To establish that G_p is a Power-coherent execution, we prove each of the Power axioms.

no-thin-air Suppose by contradiction that $\langle a, a\rangle \in hb^+$. Obviously $\langle a, a\rangle \notin rmw$, thus $\langle g(a), g(a)\rangle \in (po \cup rf)^+$ by Lemma 35. But $(po \cup rf)$ is acyclic, contradiction.

sc-per-loc Suppose by contradiction that there exists $a_p \in A_p$ s.t. $\langle a_p, a_p\rangle \in ((po_p \cup rf_p \cup fr_p \cup co)|_x)^+$. Then by Lemma 36 and since obviously $\langle a_p, a_p\rangle \notin rmw$ we obtain $\langle g(a_p), g(a_p)\rangle \in ((po \cup rf \cup fr \cup mo)|_x)^+$, which is a contradiction because from Prop. 6 and Prop. 2 this relation is acyclic.

atomic Let a_p, b_p, c_p s.t. $\langle a_p, b_p\rangle \in rmw$ and $\langle a_p, c_p\rangle \in fre_p$ and $\langle c_p, b_p\rangle \in coe$. Since $\langle a_p, b_p\rangle \in rmw$, $g(a_p) = g(b_p)$. By Lemma 35, we obtain $\langle g(a_p), g(c_p)\rangle \in fr$ and $\langle g(c_p), g(a_p)\rangle \in mo$. $\langle g(a_p), g(c_p)\rangle \in fr$ implies that there exists $d \in A$ s.t. $\langle d, g(c_p)\rangle \in mo$ and $\langle d, g(a_c)\rangle \in rf$ and $loc(d) = loc(g(a_c))$, thereby contradicting SRA-coherence.

propagation Notice that $\langle a, b\rangle \in (co \cup prop)^+$ implies that $\langle g(a), g(b)\rangle \in (po \cup rf \cup mo)^+$ by Lemma 35. Thus since $po \cup rf \cup mo$ is acyclic it must be that $co \cup prop$ is acyclic.

observation Let $\langle a, a\rangle \in fre_p; prop; hb^* \subseteq fre_p; hb^+$. There exists $c \in A_p$ s.t. $\langle a, b\rangle \in fre_p$, $\langle b, a\rangle \in hb^+$. From the *sc-per-loc* property we have already established, we know $\langle b, a\rangle \notin po$ and hence $\langle b, a\rangle \notin rmw$. Thus, we have $\langle g(a), g(b)\rangle \in fr$ and $\langle g(b), g(a)\rangle \in (po \cup rf)^+$ by Lemma 35. But then we have a cycle in $po \cup rf \cup fr|_{loc(a)}$, which contradicts SRA-coherence of G. □