What's Decidable about Causally Consistent Shared Memory?

ORI LAHAV, Tel Aviv University, Israel
UDI BOKER, Reichman University, Israel

While causal consistency is one of the most fundamental consistency models weaker than sequential consistency, the decidability of safety verification for (finite-state) concurrent programs running under causally consistent shared memories is still unclear. In this paper, we establish the decidability of this problem for two standard and well-studied variants of causal consistency. To do so, for each variant, we develop an equivalent "lossy" operational semantics, whose states track possible futures, rather than more standard semantics that record the history of the execution. We show that these semantics constitute well-structured transition systems, thus enabling decidable verification. Based on a key observation, which we call the "shared-memory causality principle", the two novel semantics may also be of independent use in the investigation of weakly consistent models and their verification. Interestingly, our results are in contrast to the undecidability of this problem under the Release/Acquire fragment of the C/C++11 memory model, which forms another variant of causally consistent memory that, in terms of allowed outcomes, lies strictly between the two models studied here. Nevertheless, we show that all these three variants coincide for write/write-race-free programs, which implies the decidability of verification for such programs under Release/Acquire.

CCS Concepts: • **Software and its engineering → Software verification**; **Concurrent programming languages**; • **Theory of computation → Concurrency**; **Logic and verification**; **Program verification**; • **Information systems →** *Distributed database transactions.*

Additional Key Words and Phrases: weak memory models, causal consistency, release/acquire, shared-memory, concurrency, verification, decidability, well-structured transition systems

ACM Reference Format:
Ori Lahav and Udi Boker. 2021. What's Decidable about Causally Consistent Shared Memory?. *ACM Trans. Program. Lang. Syst.* 1, 1, Article 1 (January 2021), 54 pages. https://doi.org/10.1145/3505273

1 INTRODUCTION

Causal consistency is one of the most fundamental consistency models weaker than sequential consistency, which is especially common and well studied in distributed data stores (see, e.g., [44, 57]). Roughly speaking, by allowing nodes to disagree on the relative order of some operations, and requiring global consensus only on the order of "causally related" operations, unlike sequential consistency, causal consistency allows scalable, partition-tolerant and available implementations, and considered as an "optimal tradeoff between user-perceived correctness and coordination overhead" [57]. Nowadays, causal consistency also plays a central role in shared memory multithreaded programming. For instance, the Release/Acquire fragment (RA) of the C/C++11 standard [15, 27, 28], which specifies the guarantees C and C++ ensure for their widely used `memory_order_release` and `memory_order_acquire` synchronization accesses, is a form of causal consistency. In addition, multiprocessor architectures like POWER, which is not "multi-copy atomic" (it allows different

Authors' addresses: Ori Lahav, Tel Aviv University, Tel Aviv, Israel, orilahav@tau.ac.il; Udi Boker, Reichman University, Herzliya, Israel, udiboker@idc.ac.il.

0164-0925/2021/1-ART1 $15.00
https://doi.org/10.1145/3505273

threads to detect stores of another thread at different times), provide barriers that can be used to ensure causal consistency, and are cheaper than the barriers needed for ensuring sequential consistency [10, 36, 50].

Despite their centrality, until recently not much was known about the safety verification problem under causal consistency models. That is: can we automatically verify that a given program satisfies a certain safety specification (e.g., it never crashes) when it runs under a causally consistent memory? When the program's data domain is bounded, this verification problem is trivially decidable under sequential consistency (SC). Indeed, such a program can be represented as a finite-state transition system; the SC memory constitutes another finite-state system; and their synchronization is easily expressible as a finite-state system as well. However, if the memory does not ensure sequential consistency, but rather provides weaker consistency guarantees, causal consistency in particular, the decidability of the safety verification problem becomes completely unclear.

The challenge arises since causally consistent memories are inherently infinite-state. In these models threads may generally read from an unbounded past, and whether or not a thread can read some value depends on the arbitrarily long execution history. More technically speaking, by "operationalizing" the declarative (a.k.a. axiomatic) formulations of causal consistency, one obtains *infinite*-state machines where each state records the (partially ordered) unbounded execution history that led to this state (either explicitly or implicitly using, e.g., timestamps). A more concrete evidence for this verification challenge is provided by the reduction of Atig et al. [12] from reachability in lossy FIFO channel machines to safety verification under x86-TSO semantics. This reduction straightforwardly applies to causally consistent models, which implies a tough non-primitive recursive lower bound on the safety verification problem under causal consistency. In fact, recently, Abdulla et al. [2] proved that for the RA fragment of C/C++11 this verification problem is *undecidable*.

The main contribution of this paper is a novel operational semantics for two causally consistent models that is equivalent to their original semantics and allows us to establish the *decidability* of safety verification for these models. The two models, called SRA (for Strong Release/Acquire) and WRA (for Weak Release/Acquire), are standard well-studied variants of causal consistency. The SRA model is the causal consistency model employed in distributed data stores as defined in [19]. As shown in [36] it also precisely captures the guarantees provided by the POWER architecture for programs compiled from the C/C++'s RA fragment. In turn, the WRA model provides the most minimal guarantees required from a model to satisfy causal consistency; it is equivalent to the model called CC studied in [17]; and it was considered as a useful candidate for shared-memory concurrency semantics [31, 34].[1]

The key idea in the new semantics for the SRA or WRA memory models is that, instead of keeping track of the execution *past* (a.k.a. history) in the system's states as often done in weakly consistent models, we maintain the possible execution *future*. Concretely, the states of the new memory systems record the potential of each thread that prescribes what sequences of operations the thread may perform. Thus, read transitions are simple—they deterministically consume a prefix of the potential. The complexity is left to write transitions that non-deterministically "set the future": what thread will read from the executed write and when. This requires us to identify how to increase the potentials of the threads when a write is performed in a way that is defined solely in terms of the threads' potentials before the write, and is both sound (sufficiently constrained to ensure only causally consistent behaviors) and complete (sufficiently free to allow every causally consistent behavior). To do so, we identify a key property that characterizes causal consistency in terms of threads' potentials, which we call the *shared-memory causality principle* (see §5). We prove the correspondence of our semantics to SRA's and WRA's original formulations using simulation

[1] We refer the reader to §3.1 for a detailed discussion on the relation between SRA and WRA to other models.

arguments (forward simulation for one direction and backward simulation for the converse). We believe that the framework of potential-based semantics may be applicable for other variants of causal consistency and also beyond the context of causal consistency.

Decidability of verification in the new semantics follows by using the framework of well-structured transition systems [1, 8, 25]. Intuitively speaking, this framework allows one to establish decidability of control state reachability under infinite-state *"lossy"* systems, where (i) states may non-deterministically forget some information they include; and (ii) the relation determining whether one state is obtained from another by losing information constitutes a well-quasi-ordering. When states consist of execution *histories* this approach cannot be applied. First, in many cases forgetting information from an execution history results in strictly weaker constraints that allow outcomes that *cannot* be obtained without losing the information. Second, execution histories are only *partially ordered* and embedding between (general) partial orders is *not* a well-quasi-ordering. On the other hand, the potential-based semantics, that tracks possible *futures* easily lends itself to verification in this framework. It is naturally "lossy": losing some parts of a possible potential never allows for additional behaviors. In addition, unlike histories, potentials are represented using total orders (lists of future actions), whose embedding relation (based on the ordinary subsequence relation) is a well-quasi-ordering.

Interestingly, the RA model, which induces an undecidable verification problem [2], is placed in between WRA and SRA—the behaviors allowed under SRA are a subset of those allowed by RA, which are a subset of those allowed by WRA. Thus, if one is specifically interested in verification under RA, our results provide both an over-approximation (successful verification under WRA implies safety under RA) and an under-approximation (a bug under SRA implies a bug under RA). Furthermore, we show that RA, SRA and WRA coincide on *write/write-race-free* programs, and hence, we obtain the decidability of safety verification also under RA for this large and widely used class of programs (see §3.2).

Outline. The rest of this paper is organized as follows. In §2 we define the safety verification problem under general declarative models. In §3 we present the WRA, RA and SRA declarative models and prove that they coincide for write/write-race-free programs. In §4 we present operational presentations of these models and define their induced reachability problem. In §5 and §6 we introduce our novel operational lossy semantics of SRA and WRA based on the "shared-memory causality principle" (starting with SRA since its semantics is simpler). In §7 we establish the correspondence of the lossy systems to the original semantics. In §8, we show how the lossy systems are used to decide the safety verification problem. In §9 we survey related work. We conclude and discuss several avenues for future work in §10. Appendix A presents the full proofs of the equivalence results sketched in §7.

To establish confidence, we have formalized the equivalence proofs in the Coq proof assistant. Claims that were proved in Coq are marked with a  symbol, and the formalization is available at https://www.cs.tau.ac.il/~orilahav/papers/causal_verification/.

Differences with the conference version of this paper. This paper is an extension and a continuation of the conference paper [35]. The latter studied *only* the SRA model and naturally skipped most of the proofs. In turn, in this paper we show that the potential technique is more widely applicable, by extending it to the much weaker WRA model. Thus, interestingly, we "surround" the RA model whose verification problem is undecidable with two models, one stronger and one weaker, and for both of which we establish the decidability of verification. We also include more examples, detailed discussions, proof outlines, and full proofs.

$v \in \mathsf{Val} \subseteq \mathbb{N}$	values
$x, y, z \in \mathsf{Loc} \subseteq \{\mathsf{x}, \mathsf{y}, \ldots\}$	locations
$r \in \mathsf{Reg} \subseteq \{\mathsf{a}, \mathsf{b}, \ldots\}$	registers
$\tau, \pi, \eta \in \mathsf{Tid} \subseteq \{\mathsf{T}_1, \mathsf{T}_2, \ldots\}$	thread identifiers
$S \in \mathsf{SProg} \triangleq \{0, 1, \ldots, N\} \to \mathsf{Inst}$	sequential programs
$P : \mathsf{Tid} \to \mathsf{SProg}$	(concurrent) programs

$$e ::= r \mid v \mid e + e \mid e = e \mid e \neq e \mid \ldots$$

$$\mathsf{Inst} \ni \mathit{inst} ::= r := e \mid \texttt{if } e \texttt{ goto } n \mid x := e \mid r := x \mid r := \texttt{FADD}(x, e) \mid r := \texttt{XCHG}(x, e) \mid r := \texttt{CAS}(x, e, e)$$

Fig. 1. Domains, metavariables and programming language syntax.

2 PRELIMINARIES: SAFETY VERIFICATION UNDER DECLARATIVE MODELS

In this section, we describe the safety verification problem for finite-state concurrent programs running under a (general) declarative memory model. For this matter, we introduce a toy programming language and the interpretation of its programs as transition systems (§2.1), and present the generic framework of declarative shared-memory semantics using execution graphs (§2.2).

2.1 Programming Language

Let $\mathsf{Val} \subseteq \mathbb{N}$, $\mathsf{Loc} \subseteq \{\mathsf{x}, \mathsf{y}, \ldots\}$, $\mathsf{Reg} \subseteq \{\mathsf{a}, \mathsf{b}, \ldots\}$ and $\mathsf{Tid} \subseteq \{\mathsf{T}_1, \mathsf{T}_2, \ldots\}$ be *finite* sets of values, (shared) memory locations, register names and thread identifiers. Figure 1 presents our toy programming language. Its expressions are constructed from registers (local variables) and values. Instructions include assignments and conditional branching, as well as memory operations. Intuitively speaking, an assignment $r := e$ assigns the value of e to register r (involving no memory access); $\texttt{if } e \texttt{ goto } n$ sets the program counter to n iff the value of e is not 0; a "write" $x := e$ stores the value of e in x; a "read" $r := x$ loads the value of x to register r; $r := \texttt{FADD}(x, e)$ atomically increments x by the value of e and loads the old value of x to r; $r := \texttt{XCHG}(x, e)$ atomically swaps x to the value of e and loads the old value of x to r; and $r := \texttt{CAS}(x, e_{\mathsf{R}}, e_{\mathsf{W}})$ atomically loads the value of x to r, compares it to the value of e_{R}, and if the two values are equal, replaces the value of x by the value of e_{W}.

A sequential program S is a function from a finite subset of $\mathbb{N} = \{0, 1, 2, \ldots\}$ (possible values of the program counter) to instructions. We denote by SProg the set of all sequential programs. A (concurrent) program P is a top-level parallel composition of sequential programs, defined as a mapping from Tid of thread identifiers to SProg. In our examples, we often write sequential programs as sequences of instructions delimited by line breaks, use '$\|$' for parallel composition, ignore threads that are mapped to the empty sequential program and refer to the program threads as $\mathsf{T}_1, \mathsf{T}_2, \ldots$ following their left-to-right order in the program listing (see, e.g., Ex. 3.3 on Page 9).

Sequential and concurrent programs induce labeled transition systems.

Labeled transition systems. A *labeled transition system* (LTS) A over an alphabet Σ is a triple $\langle Q, Q_0, T \rangle$, where Q is a set of *states*, $Q_0 \subseteq Q$ is the set of *initial states*, and $T \subseteq Q \times \Sigma \times Q$ is a set of *transitions*. We denote by $A.\mathtt{Q}$, $A.\mathtt{Q}_0$ and $A.\mathtt{T}$ the three components of an LTS A; write $\xrightarrow{\sigma}_A$ for the relation $\{\langle q, q' \rangle \mid \langle q, \sigma, q' \rangle \in A.\mathtt{T}\}$ and $\to_A$ for $\bigcup_{\sigma \in \Sigma} \xrightarrow{\sigma}_A$. A state $q \in A.\mathtt{Q}$ is *reachable* in A if $q_0 \to_A^* q$ for some $q_0 \in A.\mathtt{Q}_0$. A sequence $\sigma_1, \ldots, \sigma_n$ is a *trace* of A if $q_0 \xrightarrow{\sigma_1}_A \cdots \xrightarrow{\sigma_n}_A q$ for some $q_0 \in A.\mathtt{Q}_0$ and $q \in A.\mathtt{Q}$. The set of *predecessors* of a set $S \subseteq A.\mathtt{Q}$ *w.r.t. a symbol* $\sigma \in \Sigma$, denoted by $\mathsf{pred}_A^\sigma(S)$, is given by $\{q \in A.\mathtt{Q} \mid \exists q' \in S.\ q \xrightarrow{\sigma}_A q'\}$. The set of *predecessors* of a set $S \subseteq A.\mathtt{Q}$, denoted by $\mathsf{pred}_A(S)$, is given by $\bigcup_{\sigma \in \Sigma} \mathsf{pred}_A^\sigma(S)$.

For sequential programs the alphabet Σ is the set of *labels* (extended with ε for silent transitions), as defined next.

Definition 2.1. A *label* is either $\mathtt{R}(x, v_{\mathsf{R}})$ (*read label*), $\mathtt{W}(x, v_{\mathsf{W}})$ (*write label*) or $\mathtt{RMW}(x, v_{\mathsf{R}}, v_{\mathsf{W}})$ (*read-modify-write label*), where $x \in \mathsf{Loc}$ and $v_{\mathsf{R}}, v_{\mathsf{W}} \in \mathsf{Val}$. We denote by Lab the set of all labels. The

$$\frac{S(pc) = r := e \quad \phi' = \phi[r \mapsto \phi(e)]}{\langle pc, \phi\rangle \xrightarrow{\varepsilon} \langle pc+1, \phi'\rangle} \qquad \frac{S(pc) = \texttt{if } e \texttt{ goto } n \quad \phi(e) \neq n \implies pc' = 0 \quad \phi(e) = 0 \implies pc' = pc+1}{\langle pc, \phi\rangle \xrightarrow{\varepsilon} \langle pc', \phi\rangle}$$

$$\frac{S(pc) = x := e \quad l = \mathtt{W}(x, \phi(e))}{\langle pc, \phi\rangle \xrightarrow{l} \langle pc+1, \phi\rangle} \qquad \frac{S(pc) = r := x \quad l = \mathtt{R}(x, v) \quad \phi' = \phi[r \mapsto v]}{\langle pc, \phi\rangle \xrightarrow{l} \langle pc+1, \phi'\rangle}$$

$$\frac{S(pc) = r := \mathtt{FADD}(x, e) \quad l = \mathtt{RMW}(x, v, v + \phi(e)) \quad \phi' = \phi[r \mapsto v]}{\langle pc, \phi\rangle \xrightarrow{l} \langle pc+1, \phi'\rangle} \qquad \frac{S(pc) = r := \mathtt{XCHG}(x, e) \quad l = \mathtt{RMW}(x, v, \phi(e)) \quad \phi' = \phi[r \mapsto v]}{\langle pc, \phi\rangle \xrightarrow{l} \langle pc+1, \phi'\rangle}$$

$$\frac{S(pc) = r := \mathtt{CAS}(x, e_{\mathtt{R}}, e_{\mathtt{W}}) \quad l = \mathtt{RMW}(x, \phi(e_{\mathtt{R}}), \phi(e_{\mathtt{W}})) \quad \phi' = \phi[r \mapsto \phi(e_{\mathtt{R}})]}{\langle pc, \phi\rangle \xrightarrow{l} \langle pc+1, \phi'\rangle} \qquad \frac{S(pc) = r := \mathtt{CAS}(x, e_{\mathtt{R}}, e_{\mathtt{W}}) \quad l = \mathtt{R}(x, v) \quad v \neq \phi(e_{\mathtt{R}}) \quad \phi' = \phi[r \mapsto v]}{\langle pc, \phi\rangle \xrightarrow{l} \langle pc+1, \phi'\rangle}$$

Fig. 2. Transitions of LTS induced by a sequential program $S \in \mathsf{SProg}$.

functions typ, loc, $\mathtt{val}_{\mathtt{R}}$ and $\mathtt{val}_{\mathtt{W}}$ return (when applicable) the type (R/W/RMW), location, read value and written value of a given label $l \in \mathsf{Lab}$.

A sequential program $S \in \mathsf{SProg}$ induces an LTS over $\mathsf{Lab} \cup \{\varepsilon\}$. Its states are pairs $s = \langle pc, \phi\rangle$ where $pc \in \mathbb{N}$ (called *program counter*) and $\phi : \mathsf{Reg} \to \mathsf{Val}$ (called *local store*, and extended to expressions in the obvious way). Its only initial state is $\langle 0, \lambda r \in \mathsf{Reg}.\ 0\rangle$ and its transitions are given in Fig. 2, following the informal description above. Note that at this level, the loaded values are not restricted whatsoever, so that, in particular, a read instruction in S induces $|\mathsf{Val}|$ transitions with different read values. The execution of a sequential program S terminates when pc reaches a value that is not in the domain of S. In the sequel, we identify sequential programs with their induced LTSs (when writing, e.g., $S.\mathtt{Q}$ and $\to_S$ for a sequential program S).

In turn, a concurrent program P is identified with an LTS over the alphabet $\mathsf{Tid} \times (\mathsf{Lab} \cup \{\varepsilon\})$. Its states are functions, denoted by $\overline{p}$, assigning a state in $P(\tau).\mathtt{Q}$ to every $\tau \in \mathsf{Tid}$; its initial states set is $\{\overline{p} \mid \forall \tau.\ \overline{p}(\tau) \in P(\tau).\mathtt{Q}_0\}$; and its transitions are "interleaved transitions" of P's components, given by:

$$\frac{l \in \mathsf{Lab} \qquad \overline{p}(\tau) \xrightarrow{l}_{P(\tau)} \langle pc, \phi\rangle}{\overline{p} \xrightarrow{\tau, l} \overline{p}[\tau \mapsto \langle pc, \phi\rangle]} \qquad \frac{\overline{p}(\tau) \xrightarrow{\varepsilon}_{P(\tau)} \langle pc, \phi\rangle}{\overline{p} \xrightarrow{\tau, \varepsilon} \overline{p}[\tau \mapsto \langle pc, \phi\rangle]}$$

2.2 Declarative Memory Models and their Reachability Problem

A declarative memory model is formulated as a collection of constraints on execution graphs, which determine the *consistent* execution graphs—the ones allowed by the model. Each execution graph describes a (partially ordered) history of a particular program run. Next, we present the general notions used to assign such semantics to concurrent programs. First, we define execution graphs, starting with their nodes, called *events*.

Definition 2.2. An *event* is a triple $e = \langle \tau, n, l\rangle$, where $\tau \in \mathsf{Tid}$ is a thread identifier, $n \in \mathbb{N}$ is a serial number and $l \in \mathsf{Lab}$ is a label (of the form $\mathtt{R}(x, v_{\mathtt{R}})$, $\mathtt{W}(x, v_{\mathtt{W}})$ or $\mathtt{RMW}(x, v_{\mathtt{R}}, v_{\mathtt{W}})$, as defined in Def. 2.1). The function tid returns the thread identifier of an event. The functions typ, loc, $\mathtt{val}_{\mathtt{R}}$ and $\mathtt{val}_{\mathtt{W}}$ are lifted to events in the obvious way. We denote by E the set of all events, and use R, W, RMW for its subsets: $\mathsf{R} \triangleq \{e \mid \mathsf{typ}(e) \in \{\mathtt{R}, \mathtt{RMW}\}\}$, $\mathsf{W} \triangleq \{e \mid \mathsf{typ}(e) \in \{\mathtt{W}, \mathtt{RMW}\}\}$ and $\mathsf{RMW} \triangleq \mathsf{R} \cap \mathsf{W}$. Sub/superscripts are used to restrict these sets to certain location (e.g., $\mathsf{W}_x = \{w \in \mathsf{W} \mid \mathsf{loc}(w) = x\}$) and/or thread identifier (e.g., $\mathsf{E}^\tau = \{e \in \mathsf{E} \mid \mathsf{tid}(e) = \tau\}$).

Our representation of events induces a partial order $<$ on them: events of the same thread are ordered according to their serial numbers (i.e., $\langle \tau_1, n_1, l_1\rangle < \langle \tau_2, n_2, l_2\rangle$ iff $\tau_1 = \tau_2$ and $n_1 < n_2$). In

turn, an execution graph consists of a set of events, a *reads-from* mapping that determines the write event from which each read event reads its value, and a *modification order* (a.k.a. coherence order or store order) that totally orders the writes to each location.[2]

Definition 2.3. A relation rf is a *reads-from* relation for a set E of events if the following hold:

- If $\langle w, r\rangle \in rf$, then $w \in E \cap \mathsf{W}$, $r \in E \cap \mathsf{R}$, $\mathtt{loc}(w) = \mathtt{loc}(r)$ and $\mathtt{val}_{\mathsf{W}}(w) = \mathtt{val}_{\mathsf{R}}(r)$.
- If $\langle w_1, r\rangle, \langle w_2, r\rangle \in rf$, then $w_1 = w_2$ (that is, $rf^{-1} = \{\langle r, w\rangle \mid \langle w, r\rangle \in rf\}$ is functional).
- $\forall r \in E \cap \mathsf{R}.\ \exists w.\ \langle w, r\rangle \in rf$ (each read event reads from some write event).

Definition 2.4. A relation mo is a *modification order* for a set E of events if mo is a disjoint union of relations $\{mo_x\}_{x \in \mathsf{Loc}}$ where each mo_x is a strict total order on $E \cap \mathsf{W}_x$.

Definition 2.5. An *execution graph* is a triple $G = \langle E, rf, mo\rangle$ where E is a finite set of events, rf is a reads-from relation for E and mo is a modification order for E. We denote by EGraph the set of all execution graphs. The components of G are denoted by $G.\mathsf{E}$, $G.\mathtt{rf}$ and $G.\mathtt{mo}$. The *program order* in G, denoted by $G.\mathsf{po}$, is the restriction of $<$ to $G.\mathsf{E}$ (i.e., $G.\mathsf{po} \triangleq \{\langle e_1, e_2\rangle \in E \times E \mid e_1 < e_2\}$). For a set $E \subseteq \mathsf{E}$, we write $G.E$ for $G.\mathsf{E} \cap E$ (e.g., $G.\mathsf{W}_x = G.\mathsf{E} \cap \mathsf{W}_x$).

The next definition is used to associate execution graphs to programs. Multiple examples below (e.g., on Pages 9 and 10) illustrate execution graphs of different programs.

Notation 2.6. For a set E of events, thread identifier $\tau \in \mathsf{Tid}$ and label $l \in \mathsf{Lab}$, $\mathsf{NextEvent}(E, \tau, l)$ denotes the event given by $\langle \tau, 1 + \max(\{n \in \mathbb{N} \mid \exists l' \in \mathsf{Lab}.\ \langle \tau, n, l'\rangle \in E\}), l\rangle$.

Definition 2.7. An execution graph G is *generated by a program P with final state $\overline{p}$* if $\langle \overline{p}_0, G_\emptyset\rangle \rightarrow^* \langle \overline{p}, G\rangle$ for some $\overline{p}_0 \in P.\mathsf{Q}_0$, where $G_\emptyset$ denotes the empty execution graph (given by $G_\emptyset \triangleq \langle \emptyset, \emptyset, \emptyset\rangle$) and $\rightarrow$ is defined by:

$$\frac{\overline{p} \xrightarrow{\tau, l}_P \overline{p}' \qquad E' = E \cup \{\mathsf{NextEvent}(E, \tau, l)\} \qquad rf \subseteq rf' \qquad mo \subseteq mo'}{\langle \overline{p}, \langle E, rf, mo\rangle\rangle \rightarrow \langle \overline{p}', \langle E', rf', mo'\rangle\rangle} \qquad \frac{\overline{p} \xrightarrow{\tau, \varepsilon}_P \overline{p}'}{\langle \overline{p}, G\rangle \rightarrow \langle \overline{p}', G\rangle}$$

The rf and mo components are arbitrary at this stage, except for the fact that they have to satisfy the conditions of Definitions 2.3 and 2.4 (so that $\langle E, rf, mo\rangle$ at each step is indeed an execution graph).[3] Restrictions on rf and mo are determined by the particular model at hand (see §3).

Definition 2.8. A *declarative model* X is a set of execution graphs. We often refer to the elements of X as X-*consistent* execution graphs.

Then, reachable program states under a declarative model are formally defined as follows.

Definition 2.9. A state $\overline{p}$ of a concurrent program P is *reachable* under a declarative model X if some X-consistent execution graph is generated by P with final state $\overline{p}$.

In turn, for a declarative model X, the X *reachability problem* asks whether for a given concurrent program P and "bad state" $\overline{p} \in P.\mathsf{Q}$, we have that $\overline{p}$ is reachable under X. Unfolding the definitions, this is equivalent to asking whether the state $\overline{p}$ is reachable in the transition system induced by the given program P via a program trace $\langle \tau_1, l_1\rangle, \ldots, \langle \tau_n, l_n\rangle$ and some graph that is generated by this trace according to Def. 2.7 is X-consistent.

[2] To define the WRA model below, we do not need the modification order. Nevertheless, for uniformity, we include it in the general definition.

[3] Since rf must be an inverse of a function from $E \cap \mathsf{R}$ (by Def. 2.3) and we require $rf \subseteq rf'$ at each step, we can only generate graphs G with $G.\mathsf{po} \cup G.\mathtt{rf}$ being acyclic. This suffices for the purpose of this paper, but will require certain generalization if applied for other weak memory models.

3 DECLARATIVE CAUSALLY CONSISTENT MEMORY MODELS

In this section, we formulate the three variants of causal consistency discussed in this paper as declarative models: Weak Release/Acquire (WRA), Release/Acquire (RA), and Strong Release/Acquire (SRA). Our presentation generally follows [34]. Figure 3 illustrates the different consistency constraints described below, and Table 1 summaries the constraints of each model.

After presenting the models and various examples, in §3.1 we discuss alternative formulations from the literature that result in models that are similar or equivalent to the ones presented here; and in §3.2 we establish a race freedom guarantee showing that the three models coincide for write-write-race free programs.

To formulate constraints on execution graphs, we use several additional notations.

Notation 3.1 (Relations). Given a relation R, $dom(R)$ denotes its domain; $R^?$ and R^+ denote its reflexive and transitive closures; and R^{-1} denotes its inverse. The (left) composition of relations R_1, R_2 is denoted by $R_1;R_2$. We denote by $[A]$ the identity relation on a set A, and so $[A];R;[B] = R \cap (A \times B)$.

The causal consistency models are based on the following basic derived "happens-before" relation:

$$G.\mathsf{hb} \triangleq (G.\mathsf{po} \cup G.\mathtt{rf})^+$$

The happens-before relation captures the "causality relation" in execution graphs. In words, hb is the smallest transitive relation that contains the program order (po) and the reads-from (rf) relations. We note that every read synchronizes with the write it reads from ($\mathtt{rf} \subseteq \mathsf{hb}$), in contrast to more elaborate models like RC11 [40], where only certain reads-from edges induce synchronization. Causality is assumed to be a partial order, and accordingly, the first fundamental condition in all causal consistency models is:

$$G.\mathsf{hb} \text{ is irreflexive} \qquad \text{(irr-hb)}$$

In particular, this condition forbids so-called "load-buffering" behaviors [46], which are allowed in weaker models that aim to support write-after-read reorderings (and unless restricted appropriately lead to the infamous "out-of-thin-air" problem [14, 30]).

The next condition requires that the modification order mo "agrees" with the causality order. There are two natural ways to formally state this property. The first, followed by the RA model, requires a *local* agreement:

$$G.\mathtt{mo} \,;\, G.\mathsf{hb} \text{ is irreflexive} \qquad \text{(write-coherence)}$$

In words, if hb orders two writes to the same location, then mo must follow the same order. (Recall that, by definition, mo orders *every* pair of distinct writes to the same location.) A stronger condition, followed by SRA, requires a *global* agreement:

$$(G.\mathsf{hb} \cup G.\mathtt{mo})^+ \text{ is irreflexive} \qquad \text{(strong-write-coherence)}$$

Note that $(\mathsf{hb} \cup \mathtt{mo})$-cycles involving only one location are already disallowed by write-coherence (using the fact that mo is total on writes to the same location). But, strong-write-coherence imposes constraints on the relation between $[\mathrm{W}_x] \,;\, \mathtt{mo} \,;\, [\mathrm{W}_x]$ and $[\mathrm{W}_y] \,;\, \mathtt{mo} \,;\, [\mathrm{W}_y]$ also for $x \neq y$ (see the 2+2W program in Ex. 3.6 below). In turn, in WRA, the modification order mo plays *no role*, and imposing either write-coherence or strong-write-coherence (or none of them) has no effect on the outcomes allowed under WRA.

The next condition intuitively requires that "a thread cannot read a value when it is aware of a later value written to the same location". There is more than one way to precisely interpret this requirement: what do "aware" and "later" mean? The three models agree on the interpretation of "aware", identifying a thread τ being aware of some write event w with hb from w to (some event

WRA	irr-hb	weak-read-coherence		weak-atomicity
RA	irr-hb	write-coherence	read-coherence	atomicity
SRA	strong-write-coherence	read-coherence		atomicity

Table 1. The constraints used in each model.

of) τ. They do, however, differ in their interpretation of one write being "later" than another. RA and SRA employ the modification order $\mathtt{mo}$ for this purpose. Thus, RA and SRA require that:

$$G.\mathtt{mo} \,;\, G.\mathtt{hb} \,;\, G.\mathtt{rf}^{-1} \text{ is irreflexive} \qquad \text{(read-coherence)}$$

Indeed, if a read event r reads from a write event w_1, while being aware of an $\mathtt{mo}$-later write event w_2 to the same location, we have $\langle w_1, w_2\rangle \in \mathtt{mo}$, $\langle w_2, r\rangle \in \mathtt{hb}$ and $\langle r, w_1\rangle \in \mathtt{rf}^{-1}$.

WRA imposes a weaker condition by using $\mathtt{hb}$ to decide whether a write is "later" than another write to the same location, thus only *partially* ordering the writes. To state WRA's formal condition, it is convenient to use a per-location restriction of the happens-before relation:

$$G.\mathtt{hb}|_{\mathtt{loc}} \triangleq \{\langle e_1, e_2\rangle \in G.\mathtt{hb} \mid \mathtt{loc}(e_1) = \mathtt{loc}(e_2)\}$$

Using $\mathtt{hb}|_{\mathtt{loc}}$, the condition of WRA is given by:

$$G.\mathtt{hb}|_{\mathtt{loc}} \,;\, [\mathrm{W}] \,;\, G.\mathtt{hb} \,;\, G.\mathtt{rf}^{-1} \text{ is irreflexive} \qquad \text{(weak-read-coherence)}$$

Again, if a read event r reads from a write event w_1, while being aware of an $\mathtt{hb}$-later write event w_2 to the same location, we have $\langle w_1, w_2\rangle \in \mathtt{hb}|_{\mathtt{loc}} \,;\, [\mathrm{W}]$, $\langle w_2, r\rangle \in \mathtt{hb}$ and $\langle r, w_1\rangle \in \mathtt{rf}^{-1}$. Note that write-coherence (or its stronger variant—strong-write-coherence) implies that $[\mathrm{W}];\mathtt{hb}|_{\mathtt{loc}};[\mathrm{W}] \subseteq \mathtt{mo}$, and so weak-read-coherence is implied by read-coherence, and thus it holds in RA and SRA.

Finally, an additional condition ensures the "atomicity" of RMWs (without such condition an RMW would be nothing more than a read followed by a write). In RA and SRA, RMWs can only read from their immediate $\mathtt{mo}$-predecessors:

$$G.\mathtt{mo} \,;\, G.\mathtt{mo} \,;\, G.\mathtt{rf}^{-1} \text{ is irreflexive} \qquad \text{(atomicity)}$$

In words, if an RMW event e is reading from a write event w, then no write event can intervene $\mathtt{mo}$-between w and e.[4] In WRA, $\mathtt{mo}$ is immaterial, and one only requires that different RMW events never read from the same write event. Formally:

$$\forall \langle w_1, e_1\rangle, \langle w_2, e_2\rangle \in G.\mathtt{rf} \,;\, [\mathrm{RMW}].\ w_1 = w_2 \implies e_1 = e_2 \qquad \text{(weak-atomicity)}$$

(That is, $G.\mathtt{rf} \,;\, [\mathrm{RMW}]$ is a partial function.) This simple condition suffices for implementing lock acquisitions using RMWs in WRA, as well as for implementing fences using RMWs to an otherwise-unused location (see Ex. 3.9). To see that atomicity implies weak-atomicity (in the presence of write-coherence or strong-write-coherence), assume a violation of weak-atomicity, and note that since $\mathtt{mo}$ must order the two RMWs and write-coherence (or strong-write-coherence) dictates that $\mathtt{mo} \,;\, \mathtt{rf}$ is irreflexive, it entails a violation of atomicity.

Figure 3 illustrates the different constrains, and Table 1 lists the constraints of each model. Since write-coherence and read-coherence together imply weak-read-coherence; write-coherence and atomicity together imply weak-atomicity; and strong-write-coherence implies both irr-hb and write-coherence, the following proposition trivially holds.

Proposition 3.2. SRA-*consistency implies* RA-*consistency, which in turn implies* WRA-*consistency.*

[4]Note that because of the domain restrictions on $\mathtt{rf}$ and $\mathtt{mo}$, only RMW events can have both an incoming $\mathtt{rf}$ edge and an incoming $\mathtt{mo}$ edge, so atomicity can be equivalently stated as $G.\mathtt{mo} \,;\, G.\mathtt{mo} \,;\, [\mathrm{RMW}] \,;\, G.\mathtt{rf}^{-1}$ is irreflexive.

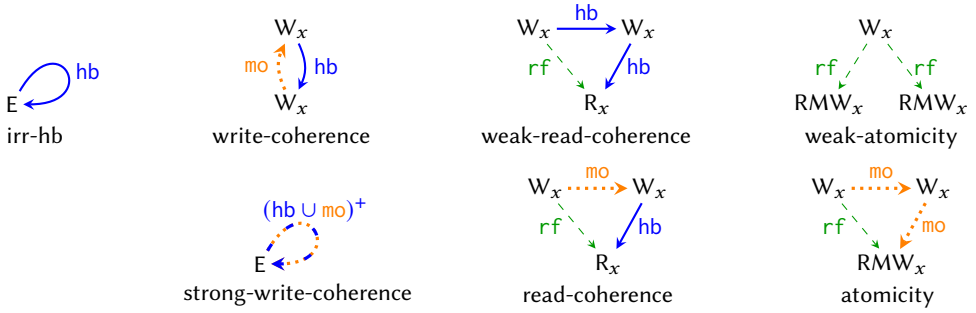


Fig. 3. Illustration of forbidden patterns in the causally consistent models.

Consequently, we clearly have that all states of a program P that are reachable under SRA are also reachable under RA; and all states of P reachable under RA are also reachable under WRA. The converses of the claims in Prop. 3.2 do not hold in general (see examples below), but, as we show in §3.2, they do hold for the class of write/write-race-free programs.

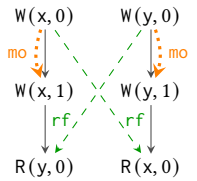
Next, we list some examples to demonstrate the different models (some of which are revisited in the sequel). Most of the examples are well-known litmus tests. To simplify the presentation, instead of referring to reachable program states, we consider possible *program outcomes* assigning final values to (some) registers. An outcome $O : \mathrm{Reg} \rightharpoonup \mathrm{Val}$ is allowed for a program under a declarative model X if some state in which the registers have their values in O and the program counters have their maximal values is reachable under X (see Def. 2.9). We use program comment annotations ("// ") to denote particular outcomes.

Remark 1. To simplify our presentation, we require explicit initialization of memory locations and adapt well-known examples to include explicit initialization. Reading from an uninitialized location *blocks* the thread. (For example, only the initial execution graph $G_\emptyset$ is generated by a program consisting of a single thread that reads from some location, without previously writing to it.) This is only a presentation matter: one may always achieve implicit initialization by augmenting the program with an additional thread that sets each variable to its initial value, and then signals all other threads (using an additional flag) to start running.

Example 3.3 (Store buffering). The following program outcome is allowed by all three causal consistency models. The justifying execution graph is presented on the right.

The rf edges are forced because of the read values, whereas the mo edges in RA and SRA are forced due to write-coherence and strong-write-coherence. It can be easily verified that the execution graph is SRA-consistent, and thus it is also RA-consistent and WRA-consistent.

```
x := 0  | y := 0
x := 1  | y := 1
a := y // 0 | b := x // 0
✓ WRA  ✓ RA  ✓ SRA
```



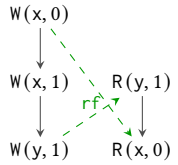
(SB)

Example 3.4 (Message passing). Causal consistency models support "flag-based" synchronization (which makes them useful in shared-memory concurrent programs). That is, the following outcome is disallowed under each of the models defined above.

An execution graph for this outcome must have rf edges as depicted on the right. However, we have $\mathsf{hb}|_{\mathsf{loc}}$ from W(x, 0) to W(x, 1), hb from W(x, 1) to R(x, 0) and rf from W(x, 0) to R(x, 0). Hence, weak-read-coherence does not hold, and the execution graph is not WRA-consistent.

x := 0 || a := y // 1
x := 1 || b := x // 0
y := 1

✗ WRA ✗ RA ✗ SRA

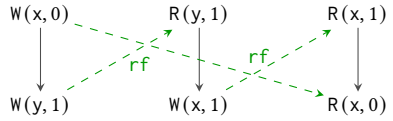


(MP)

Note that po and rf edges equally contribute to hb in causal consistency. Hence, for the same reason the following outcome is disallowed as well:

x := 0 || a := y // 1 || b := x // 1
y := 1 || x := 1 || c := x // 0

✗ WRA ✗ RA ✗ SRA

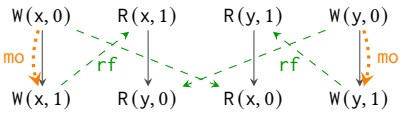


(MP-trans)

Example 3.5 (Independent reads of independent writes). A main difference between the causal consistency models and the x86-TSO model [48] is that the former are *non-multi-copy-atomic*: a write by some thread could become visible to some other threads before becoming visible to all other threads. Thus, unlike x86-TSO, the three causal consistency models allow the following outcome, in which T_2 observes W(x, 1) but not W(y, 1), while T_3 observes W(y, 1) but not W(x, 1). The justifying execution graph appears on the right:

x := 0 || a := x // 1 || c := y // 1 || y := 0
x := 1 || b := y // 0 || d := x // 0 || y := 1

✓ WRA ✓ RA ✓ SRA

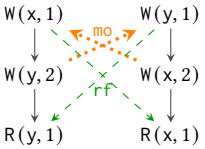


(IRIW)

Example 3.6. The following example, adapted from [58], demonstrates the fact that the *local* agreement between mo and hb required in RA is indeed weaker than the *global* agreement required by SRA:

x := 1 || y := 1
y := 2 || x := 2
a := y // 1 || b := x // 1

✓ WRA ✓ RA ✗ SRA



(2+2W)

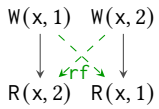
An execution graph for this outcome must have rf and mo edges as depicted above (to satisfy read-coherence), and it contains a $(\mathsf{hb} \cup \mathsf{mo})$-cycle, which is allowed by RA and disallowed by SRA.

Example 3.7. Unlike RA and SRA, WRA does not provide "sequential-consistency-per-location"—even programs with a single location may exhibit non-sequentially-consistent behaviors. For instance, this happens in the following programs:

(WW)

x := 1 || x := 2
a := x // 2 || b := x // 1

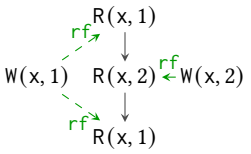
✓ WRA ✗ RA ✗ SRA



(Oscillating)

x := 1 || a := x // 1
|| b := x // 2
|| c := x // 1 || x := 2

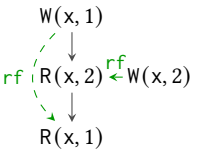
✓ WRA ✗ RA ✗ SRA



(SF)

x := 1
a := x // 2 || x := 2
b := x // 1

✓ WRA ✗ RA ✗ SRA

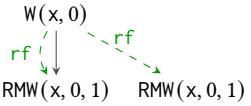


Interestingly, WRA validates a particular form of the *store forwarding* optimization that applies when a certain read is preceded (in program order) by a write to the same location and there are no writes between these two operations. In this case, the compiler may eliminate the read by assuming that it reads the value written by the write. This optimization, performed by certain Java compilers (see [24, §2.2]), is particularly applicable when pointers are involved, e.g., $x := 1; a := *p; b := x$ can be optimized to $x := 1; a := *p; b := 1$ without any pointer analysis (that is, without knowing whether p points to x or not). The (SF) example above shows that it is an unsound optimization for RA and SRA—the annotated outcome is disallowed under these models, but if we apply the above optimization, we may replace $b := x$ by $b := 1$, and the get $a = 2$ and $b = 1$ even under SC. We note that the standard store forwarding that only applies when the read *immediately* follows the write (with no operations in between) is sound in all three models.

Example 3.8. For implementing locks using RMWs it is crucial that two different RMWs never read from the same write. This is enforced directly in WRA, and follows from atomicity in RA and SRA. Indeed, in the following example, any (total) mo order of the three events cannot place the write of 0 as the immediate predecessor of *both* RMWs.

$$\begin{array}{l} x := 0 \\ a := \texttt{CAS}(x, 0, 1) \mathbin{/\!/} 0 \end{array} \Big\| \; b := \texttt{CAS}(x, 0, 1) \mathbin{/\!/} 0$$

✗WRA ✗RA ✗SRA



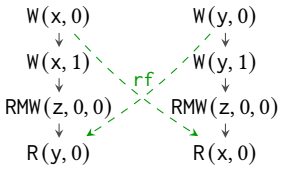
(2RMW)

Example 3.9. RMWs to an otherwise-unused (unique) location can be used as *fences*. Indeed, the consistency constraints (of any of the models) imply that if, except for the initialization write event, all write events to some location x in G are RMWs then hb must totally order $G.W_x$. For example, placing such fences forbids the weak outcome of the SB program (Ex. 3.3). An execution graph for this outcome must have the edges as depicted on the right, and any choice of the two missing rf edges (to the two RMW events) will violate some condition of WRA.

$$z := 0 \;\Big\|\; \begin{array}{l} x := 0 \\ x := 1 \\ a := \texttt{FADD}(z, 0) \\ b := y \mathbin{/\!/} 0 \end{array} \;\Big\|\; \begin{array}{l} y := 0 \\ y := 1 \\ a := \texttt{FADD}(z, 0) \\ c := x \mathbin{/\!/} 0 \end{array}$$

W(z, 0)

✗WRA ✗RA ✗SRA



(SB+RMWs)

3.1 Alternative Formulations

Our presentation follows C/C++11's mathematical formalization [15, 40], where the RA model above is the fragment of the C/C++11 model consisting of release stores, acquire reads and acquire-release RMWs. In turn, SRA is a strengthening of RA proposed in [36], whereas WRA is a natural weakening of RA that is a fragment of the weak RC11 model proposed in [31], and is sufficiently strong for the race-freedom result (Thm. 3.12 below) to hold. The WRA and SRA models appear in the literature in multiple disguises, especially as correctness criteria for distributed data stores:

POWER. As proved in [36], SRA *precisely* coincides with the POWER model of [10] (which was validated by extensive testing against real hardware), when the latter is restricted to programs that result from compiling C/C++11 programs in the release/acquire fragment, using the standard compilation scheme [45] (that is, placing `lwsync` before every store and `ctrl+isync` after every load). While POWER's plain instructions result in a model that does not provide causal consistency guarantees, this compilation scheme ensures that POWER provides causal consistency, and the observation of [36] this form of causal consistency precisely matches SRA.

Causal Convergence. Ignoring RMWs, the SRA model is equivalent to the *causal convergence* model, denoted by CCv, of [17] (when the latter is applied to the standard sequential specification of a key-value store supporting read and write operations), as well as to the causal consistency model of [44] when restricted to single-instruction transactions. These models are formulated in [19, 21] in terms of *visibility* (*vis*) and *arbitration* (*ar*) relations. For example, the graph on the left for the IRIW program (Ex. 3.5) is captured in these terms by the graph on the right (where the dotted arrow is used for the visibility relation and circled numbers denote the arbitration order):

One direction of the correspondence between our formulation of SRA and the alternative one formulated in terms of *vis* and *ar* follows by setting $vis = \mathsf{hb}$ and taking *ar* to be some total order extending $\mathsf{hb} \cup \mathsf{mo}$. For the converse, one takes $\mathtt{rf}$ to relate each read r with the *ar*-maximal write to the same location that is *vis*-before r, and sets $\mathsf{mo} = \bigcup_{x \in \mathsf{Loc}} [\mathsf{W}_x] \,;\, ar \,;\, [\mathsf{W}_x]$. Furthermore, our program order (po) corresponds to session order (*so*), and SRA's consistency ensures *strong session guarantees* ($so \subseteq vis$) [54].

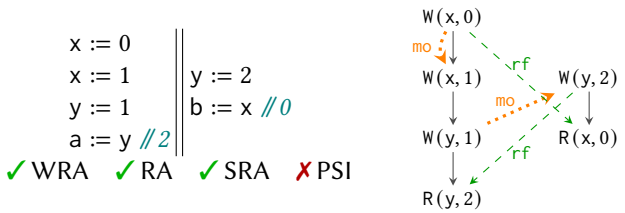
RMWs in distributed databases require expensive global coordination. A naive implementation of RMWs as transactions that read and write from/to the same location does not guarantee atomicity, as it allows the *lost update* anomaly (e.g., it will allow the outcome in Ex. 3.8). In the particular case when a certain location is *only* accessed by RMWs, its accesses are totally ordered by hb, which corresponds to marking of certain transactions as serializable, as in the *Red-Blue* model of [16, 43].

Basic Causal Consistency. WRA (without RMWs) is equivalent to a basic causal consistency model called CC in [17], when CC is applied to the standard sequential specification of a key-value store supporting read and write operations. The CC model requires the existence of a partial "causal" order S such that for every read event r, the restriction of S on $dom(S; [\{r\}])$ can be extended to a total order in which the value written by the last write to $\mathtt{loc}(r)$ is $\mathtt{val}_{\mathsf{R}}(r)$. This condition is equivalent to the constraints of WRA.

Parallel-Snapshot-Isolation. Parallel snapshot isolation (PSI) is a standard transactional consistency model used in databases and distributed systems that offers scalability and availability in large-scale geo-replicated systems [11, 16, 20, 49, 52]. When restricted to single-instruction transactions, PSI is captured by strengthening read-coherence to require:

$$G.\mathsf{mo} \,;\, (G.\mathsf{po} \cup G.\mathtt{rf} \cup G.\mathsf{mo})^{+} \,;\, G.\mathtt{rf}^{-1} \text{ is irreflexive} \qquad \text{(strong-read-coherence)}$$

Example 3.10. The following behavior is allowed by SRA but disallowed by PSI:



(WW-MP)

An execution graph for this outcome must have $\mathtt{rf}$ and mo edges as depicted above (to satisfy read-coherence), and it violates strong-read-coherence.

It can be shown that when all store instructions are implemented using atomic exchanges (implementing $x := e$ as $\_ := \mathtt{XCHG}(x, e)$), SRA precisely captures PSI. Hence, our decidability result

for SRA entails the decidability for PSI with single-instruction transactions, via a simple reduction substituting all stores in a given program by atomic exchanges that do not use the values being exchanged. For instance, if we use in the above example $\mathtt{c} := \mathtt{XCHG}(\mathtt{y}, 2)$ instead of $\mathtt{y} := 2$, then the mo edge between the two writes to y would become an rf edge (and the second write would be labeled with $\mathtt{RMW}(\mathtt{y}, 1, 2)$), so the annotated outcome would violate read-coherence and be forbidden also under SRA.

3.2 Write/Write-Race Freedom Guarantee

Following Prop. 3.2, we have that WRA is weaker than RA, which is weaker than SRA. The examples above show that these relations are strict: the annotated behaviors of the programs in Ex. 3.7 are allowed by WRA but not by RA; and the annotated behavior of the 2+2W program in Ex. 3.6 is allowed by RA but not by SRA. We note that in all these examples, the programs exhibit write/write races, namely two different threads write to the same location with no happens-before relation between the conflicting writes. Roughly speaking, since the difference between the models concerns the mo relation, only a write/write race might expose the gap between them. In this section, we formally prove this fact by showing that the three models coincide on write/write-race-free programs. We note that the vast majority of concurrent algorithms we know of do not employ write/write races (in fact, it is rather hard to locate ones that do), which makes the next theorem widely applicable.[5]

Inspired by DRF models and results [9, 14, 40], which ensure SC semantics for programs that are data-race-free *under* SC-*semantics*, we show that write/write-race freedom of all SRA-*consistent* execution graphs of a given program suffices for the established correspondence. This allows programmers to adhere to a safe programming discipline (that is, avoid write/write races, e.g., using locks) without even understanding the two weaker models, WRA and RA. Indeed, to establish the premise of the following theorem, one only needs to know the SRA-consistency predicate.

Definition 3.11. An execution graph G is *write/write-race free* if for every $w_1, w_2 \in G.\mathsf{W}$ with $\mathtt{loc}(w_1) = \mathtt{loc}(w_2)$, we have $w_1 = w_2$, $\langle w_1, w_2\rangle \in G.\mathsf{hb}$ or $\langle w_2, w_1\rangle \in G.\mathsf{hb}$.

Theorem 3.12. *Let P be a concurrent program such that every* SRA*-consistent execution graph that is generated by P is write/write-race free. Then, the sets of states of P that are reachable under (1)* SRA*, (2)* RA *and (3)* WRA *all coincide.*

Proof. Using Prop. 3.2, it suffices to show that every state of P that is reachable under WRA is also reachable under SRA.

We call an execution graph G SRA-*pre-consistent* if some execution graph G' with $G'.\mathsf{E} = G.\mathsf{E}$ and $G'.\mathsf{rf} = G.\mathsf{rf}$ (but possibly $G'.\mathsf{mo} \neq G.\mathsf{mo}$) is SRA-consistent. Let $\mathcal{G}$ be the set of all WRA-consistent but *not* SRA-pre-consistent execution graphs that are generated by P. To show that every state of P that is reachable under WRA is also reachable under SRA, it suffices to show that $\mathcal{G}$ is empty.

Suppose otherwise and let G be a minimal element in $\mathcal{G}$, in the sense that every proper $G.\mathsf{hb}$-prefix of G is not in $\mathcal{G}$ (where a proper $G.\mathsf{hb}$-prefix of G is any execution graph of the form $\langle E_p, [E_p] ; G.\mathsf{rf} ; [E_p], [E_p] ; G.\mathsf{mo} ; [E_p]\rangle$ for some $E_p \subsetneq G.\mathsf{E}$ such that $dom(G.\mathsf{hb} ; [E_p]) \subseteq E_p$). Note that G cannot be empty, since the empty execution graph $G_\emptyset$ is trivially SRA-pre-consistent.

Let e be some $G.\mathsf{hb}$-maximal event in $G.\mathsf{E}$, and let $E' = G.\mathsf{E} \setminus \{e\}$. The minimality of G ensures that the restriction of G to E' (namely, the execution graph $\langle E', [E'] ; G.\mathsf{rf} ; [E'], [E'] ; G.\mathsf{mo} ; [E']\rangle$) is SRA-pre-consistent. Let mo' be a modification order for E' such that $G' = \langle E', [E'] ; G.\mathsf{rf} ; [E'], mo'\rangle$

[5]This fact was previously utilized in [31, Section 5] that provided an improved bounded model checking algorithm for write/write-race free programs, and identified that (sound but incomplete) separation-logic-based program logics for RA are essentially making a similar simplification, and do not support reasoning about concurrent writes.

is SRA-consistent. Note that our assumption on P ensures that G' is write/write-race free, thus using strong-write-coherence, it follows that $mo' \subseteq G'.\mathsf{hb}|_{\mathtt{loc}} \subseteq G.\mathsf{hb}|_{\mathtt{loc}}$.

We consider the possible types of e. In each case, we define a modification order $\widehat{mo}$ for $G.\mathsf{E}$ and show that $\widehat{G} = \langle G.\mathsf{E}, G.\mathtt{rf}, \widehat{mo}\rangle$ is SRA-consistent, which contradicts the fact that G is not SRA-pre-consistent.

- $\mathtt{typ}(e) = \mathtt{R}$: We define $\widehat{mo} = mo'$. Then, $\widehat{G}$ satisfies strong-write-coherence, as a $(\widehat{G}.\mathsf{hb} \cup \widehat{mo})$-cycle would have implied a cycle in $G.\mathsf{hb} \cup mo' \subseteq G.\mathsf{hb}$, which cannot exist, since G satisfies irr-hb. In addition, $\widehat{G}$ satisfies atomicity, since its violation does not involve read events, and would have occurred also in G'. Assume toward contradiction that $\widehat{G}$ does not satisfy read-coherence. Since e is $G.\mathsf{hb}$-maximal, there exist $w_1, w_2 \in E'$ such that $\langle w_1, w_2\rangle \in mo'$, $\langle w_2, e\rangle \in G.\mathsf{hb}$ and $\langle w_1, e\rangle \in G.\mathtt{rf}$. It follows that $\langle w_1, w_2\rangle \in G.\mathsf{hb}|_{\mathtt{loc}}$, and so G does not satisfy weak-read-coherence, which contradicts the fact that G is WRA-consistent.
- $\mathtt{typ}(e) = \mathtt{W}$: We define $\widehat{mo} = mo' \cup (G.\mathrm{W}_x \times \{e\})$ where $x = \mathtt{loc}(e)$. It is easy to see that $\widehat{G}$ is SRA-consistent.
- $\mathtt{typ}(e) = \mathtt{RMW}$: Let $x = \mathtt{loc}(e)$ and let $w \in G.\mathrm{W}$ such that $\langle w, e\rangle \in G.\mathtt{rf}$. We define $\widehat{mo} = mo' \cup (W \times \{e\}) \cup (\{e\} \times (G.\mathrm{W}_x \setminus W))$ where $W = \{w' \in G.\mathrm{W}_x \mid \langle w', w\rangle \in mo'^{?}\}$.
  Assume toward contradiction that $\widehat{G}$ is not SRA-consistent. At least one of the following hold:
  - strong-write-coherence is not satisfied by $\widehat{G}$: Then, since G' is SRA-consistent, there exists $w' \in E'$ such that $\langle e, w'\rangle \in \widehat{mo}$ and $\langle w', e\rangle \in G.\mathsf{hb}$. Hence, we have $\langle w, w'\rangle \in mo' \subseteq G.\mathsf{hb}|_{\mathtt{loc}}$, and since $\langle w, e\rangle \in G.\mathtt{rf}$, this contradicts the fact that G satisfies weak-read-coherence.
  - read-coherence is not satisfied by $\widehat{G}$: Then, since G' is SRA-consistent, there exist $w' \in E'$ such that $\langle w, w'\rangle \in \widehat{mo}$ and $\langle w', e\rangle \in G.\mathsf{hb}$. It follows that $\langle w, w'\rangle \in G.\mathsf{hb}|_{\mathtt{loc}}$, which again contradicts the fact that G satisfies weak-read-coherence.
  - atomicity is not satisfied by $\widehat{G}$: Then, since G' is SRA-consistent, it follows that there exist $w' \in E'.\mathrm{W}$ and $u \in E'.\mathrm{RMW}$, such that $\langle w', e\rangle, \langle e, u\rangle \in \widehat{mo}$ and $\langle w', u\rangle \in G.\mathtt{rf}$. The construction of $\widehat{mo}$ ensures that $\langle w', w\rangle \in mo'^{?}$ and $\langle w, u\rangle \in mo'$. Hence, $\langle w', w\rangle \in G.\mathsf{hb}|^{?}_{\mathtt{loc}}$ and $\langle w, u\rangle \in G.\mathsf{hb}|_{\mathtt{loc}}$. Now, if $\langle w', w\rangle \in G.\mathsf{hb}|_{\mathtt{loc}}$, then again we obtain a contradiction to the fact that G satisfies weak-read-coherence. Otherwise, we have $w' = w$. Thus, we have both $\langle w, e\rangle \in G.\mathtt{rf}$ and $\langle w, u\rangle \in G.\mathtt{rf}$ (where $e \neq u$ since $u \in E'$), which contradicts the fact the G satisfies weak-atomicity. □

4 AN OPERATIONAL LOOK AT CAUSAL CONSISTENCY AND ITS INDUCED REACHABILITY PROBLEM

While the above formulations of the casual consistency models are declarative, it is straightforward to "operationalize" these definitions. Indeed, for the models above, instead of first generating a program execution graph (using Def. 2.7) and a posteriori checking its consistency, one may impose consistency *at each step* during an incremental construction of the execution graph. This results in equivalent operational presentations, which are easier to relate to the alternative lossy semantics we define below. In this section, we present such operational reformulations of the declarative semantics above, formulating them as *memory systems*.[6] We will reuse this operational framework for the lossy semantics (§5 and §6).

Definition 4.1. A *memory system* is a (possibly infinite) LTS over the alphabet $(\mathsf{Tid} \times \mathsf{Lab}) \cup \{\varepsilon\}$.

The alphabet symbols of the memory system are either pairs in $\mathsf{Tid} \times \mathsf{Lab}$, representing the thread identifier and the label of the performed operation, or ε for internal (silent) memory actions.

[6] A similar construction appears in [10] for hardware memory models and the resulting memory systems are called "Intermediate Machines".

Example 4.2 (Sequential consistency as a memory system). The most well-known memory system is the one of sequential consistency, denoted here by SC. This memory system simply tracks the most recent value written to each location (or $\bot$ for uninitialized locations). Formally, it is defined by $\mathsf{SC}.\mathtt{Q} \triangleq \mathsf{Loc} \to (\mathsf{Val} \cup \{\bot\})$, $\mathsf{SC}.\mathtt{Q}_0 \triangleq \{\lambda x \in \mathsf{Loc}.\, \bot\}$ and $\to_{\mathsf{SC}}$ is given by:

$$\textsc{write}\ \frac{\mu' = \mu[x \mapsto v_{\mathtt{W}}]}{\mu \xrightarrow{\tau,\mathtt{W}(x,v_{\mathtt{W}})}_{\mathsf{SC}} \mu'} \qquad \textsc{read}\ \frac{\mu(x) = v_{\mathtt{R}}}{\mu \xrightarrow{\tau,\mathtt{R}(x,v_{\mathtt{R}})}_{\mathsf{SC}} \mu} \qquad \textsc{rmw}\ \frac{\mu(x) = v_{\mathtt{R}} \qquad \mu' = \mu[x \mapsto v_{\mathtt{W}}]}{\mu \xrightarrow{\tau,\mathtt{RMW}(x,v_{\mathtt{R}},v_{\mathtt{W}})}_{\mathsf{SC}} \mu'}$$

Note that SC is oblivious to the thread that takes the action (we have $\mu \xrightarrow{\tau,l}_{\mathsf{SC}} \mu'$ iff $\mu \xrightarrow{\pi,l}_{\mathsf{SC}} \mu'$), and it has no silent transitions.

By synchronizing a program and a memory system, we obtain a *concurrent system*:

Definition 4.3. A program P and a memory system M form a *concurrent system*, denoted by $P \parallel M$. It is an LTS over $(\mathsf{Tid} \times (\mathsf{Lab} \cup \{\varepsilon\})) \cup \{\varepsilon\}$ whose set of states is $P.\mathtt{Q} \times M.\mathtt{Q}$; its initial states set is $P.\mathtt{Q}_0 \times M.\mathtt{Q}_0$; and its transitions are "synchronized transitions" of P and M, given by:

$$\frac{l \in \mathsf{Lab} \qquad \overline{p} \xrightarrow{\tau,l}_P \overline{p}' \qquad m \xrightarrow{\tau,l}_M m'}{\langle \overline{p}, m\rangle \xrightarrow{\tau,l}_{P\parallel M} \langle \overline{p}', m'\rangle} \qquad \frac{\overline{p} \xrightarrow{\tau,\varepsilon}_P \overline{p}'}{\langle \overline{p}, m\rangle \xrightarrow{\tau,\varepsilon}_{P\parallel M} \langle \overline{p}', m\rangle} \qquad \frac{m \xrightarrow{\varepsilon}_M m'}{\langle \overline{p}, m\rangle \xrightarrow{\varepsilon}_{P\parallel M} \langle \overline{p}, m'\rangle}$$

To relate a declarative model X and a memory system M, we use the following definitions.

Definition 4.4. A state $\overline{p}$ of a concurrent program P is *reachable* under a memory system M if $\langle \overline{p}, m\rangle$ is reachable in $P \parallel M$ for some $m \in M.\mathtt{Q}$.

Definition 4.5. A memory system M *characterizes* a declarative model X if for every concurrent program P, the set of program states that are reachable under X (see Def. 2.9) coincides with the set of program states that are reachable under M.

Next, we present the memory systems opWRA, opRA, and opSRA that characterize the respective declarative model. (The opRA memory system is defined here for the completeness of the presentation, but it is not used in the sequel.) The states of these systems are execution graphs capturing (partially ordered) histories of executed actions, and the only initial state is $G_\emptyset$ (recall that $G_\emptyset$ denotes the empty execution graph $\langle \emptyset, \emptyset, \emptyset\rangle$). Formally, $M.\mathtt{Q} \triangleq \mathsf{EGraph}$ and $M.\mathtt{Q}_0 \triangleq \{G_\emptyset\}$ for $M \in \{\mathsf{opWRA}, \mathsf{opRA}, \mathsf{opSRA}\}$. Before providing the transitions, we refer the reader to Fig. 4 on Page 22, which illustrates a run of opSRA (or opRA, opWRA) for the SB program from Ex. 3.3.

Remark 2. Following [31], our formulation of the memory systems below does not directly refer to the consistency predicates, but rather articulate necessary and sufficient conditions that ensure that the target state is a consistent execution graph provided the consistency of the source state. It is possible to take a step further and develop an equivalent semantics with more economical states that may feel "more operational" and intuitive. Indeed, for the systems below, it suffices to maintain a partially ordered set of write events, together with a mapping of which writes each thread is already aware of (the "observed writes set" of [23]). When the writes to each location are totally ordered (as in RA and SRA), this can be implemented using *timestamps*, *messages* and *thread views*, as was done, e.g., in [29] for RA.

Weak Release/Acquire. The transitions of opWRA are given by:

$$\textsc{write}\ \frac{e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, \mathtt{W}(x, v_{\mathtt{W}})) \quad G'.\mathsf{E} = G.\mathsf{E} \cup \{e\} \quad G'.\mathtt{rf} = G.\mathtt{rf}}{G \xrightarrow{\tau, \mathtt{W}(x, v_{\mathtt{W}})}_{\mathsf{opWRA}} G'}$$

$$\textsc{read}\ \frac{e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, \mathtt{R}(x, v_{\mathtt{R}})) \quad G'.\mathsf{E} = G.\mathsf{E} \cup \{e\} \quad G'.\mathtt{rf} = G.\mathtt{rf} \cup \{\langle w, e\rangle\} \quad w \in G.\mathtt{W}_x \quad \mathtt{val}_{\mathtt{W}}(w) = v_{\mathtt{R}} \quad w \notin \mathit{dom}(G.\mathsf{hb}|_{\mathsf{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])}{G \xrightarrow{\tau, \mathtt{R}(x, v_{\mathtt{R}})}_{\mathsf{opWRA}} G'}$$

$$\textsc{rmw}\ \frac{e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, \mathtt{RMW}(x, v_{\mathtt{R}}, v_{\mathtt{W}})) \quad G'.\mathsf{E} = G.\mathsf{E} \cup \{e\} \quad G'.\mathtt{rf} = G.\mathtt{rf} \cup \{\langle w, e\rangle\} \quad w \in G.\mathtt{W}_x \quad \mathtt{val}_{\mathtt{W}}(w) = v_{\mathtt{R}} \quad w \notin \mathit{dom}(G.\mathsf{hb}|_{\mathsf{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau]) \quad w \notin \mathit{dom}(G.\mathtt{rf} ; [\mathsf{RMW}])}{G \xrightarrow{\tau, \mathtt{RMW}(x, v_{\mathtt{R}}, v_{\mathtt{W}})}_{\mathsf{opWRA}} G'}$$

A WRITE step simply adds a corresponding fresh write event to the graph placed in the end of the thread executing the write. A READ step adds a corresponding fresh read event and justifies it with a reads-from edge. Its source w must be a write event to the same location ($w \in G.\mathtt{W}_x$), writing the value being read ($\mathtt{val}_{\mathtt{W}}(w) = v$), and the thread executing the read must not be aware of an hb-later write to the same location ($w \notin \mathit{dom}(G.\mathsf{hb}|_{\mathsf{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$). An RMW step is similar to a READ step (adding an RMW event), with the additional condition on w: it should not be read by any RMW event in the current execution graph ($w \notin \mathit{dom}(G.\mathtt{rf} ; [\mathsf{RMW}])$). We note that the WRITE step in opWRA is deterministic, while the READ and RMW steps are non-deterministic—often more than one write can be chosen as the source of the new rf edges.

THEOREM 4.6. opWRA *characterizes* WRA.

PROOF. Given a WRA-consistent execution graph G, one obtains a run of opWRA by following any total order extending $G.\mathsf{hb}$. The preconditions required by each step follow directly from the fact that G is WRA-consistent. For the converse, it suffices to note that all reachable states of opWRA are WRA-consistent execution graphs. Hence, if $\langle \overline{p}, G\rangle$ is reachable in $P \parallel \mathsf{opWRA}$, then G is a WRA-consistent execution graph that is generated by P with final state $\overline{p}$. □

Remark 3. Instead of requiring $w \notin \mathit{dom}(G.\mathtt{rf} ; [\mathsf{RMW}])$ in the RMW step, we may equivalently require that $\{e \in \mathsf{RMW} \mid \langle w, e\rangle \in G.\mathtt{rf}\} \subseteq \mathsf{E}^\tau$ (namely, if w is read by an RMW event, then that RMW event is in thread τ). Indeed, $w \notin \mathit{dom}(G.\mathtt{rf} ; [\mathsf{RMW}])$ trivially implies this condition. Conversely, if this condition holds then since $w \notin \mathit{dom}(G.\mathsf{hb}|_{\mathsf{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$, we cannot have $w \in \mathit{dom}(G.\mathtt{rf} ; [\mathsf{RMW}])$. While this reformulation is an unnecessary complication at this stage, it plays a key role in the alternative lossy semantics for WRA in §6.

Release/Acquire. To handle modification order (mo) updates in transitions of opRA, we use the following notation:

Notation 4.7. Given a relation R that contains a total order on a set A, a subset $A_{\mathsf{before}} \subseteq A$ that is downward closed ($\mathit{dom}(R ; [A_{\mathsf{before}}]) \subseteq A_{\mathsf{before}}$) and an element $b \notin A$, $\mathsf{AddAfter}(R, A, A_{\mathsf{before}}, b)$ denotes the extension of R obtained by placing b after all elements in A_{before} and before all elements of $A \setminus A_{\mathsf{before}}$ (formally, $\mathsf{AddAfter}(R, A, A_{\mathsf{before}}, b) \triangleq R \cup (A_{\mathsf{before}} \times \{b\}) \cup (\{b\} \times (A \setminus A_{\mathsf{before}}))$).

The transitions of opRA are given by:

$$\textsc{write}\quad \frac{\begin{array}{c} e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, \mathtt{W}(x, v_{\mathtt{W}})) \\ G'.\mathsf{E} = G.\mathsf{E} \cup \{e\} \\ G'.\mathtt{rf} = G.\mathtt{rf} \\ G'.\mathtt{mo} = \mathsf{AddAfter}(G.\mathtt{mo}, G.\mathsf{W}_x, W, e) \\ dom(G.\mathtt{mo}\,;\,[W]) \subseteq W \subseteq G.\mathsf{W}_x \\ w = \max_{G.\mathtt{mo}} W \\ w \notin dom(G.\mathtt{mo}\,;\,G.\mathtt{hb}^?\,;\,[\mathsf{E}^\tau]) \\ w \notin dom(G.\mathtt{rf}\,;\,[\mathrm{RMW}]) \end{array}}{G \xrightarrow{\tau, \mathtt{W}(x, v_{\mathtt{W}})}_{\mathrm{opRA}} G'}$$

$$\textsc{read}\quad \frac{\begin{array}{c} e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, \mathtt{R}(x, v_{\mathtt{R}})) \\ G'.\mathsf{E} = G.\mathsf{E} \cup \{e\} \\ G'.\mathtt{rf} = G.\mathtt{rf} \cup \{\langle w, e\rangle\} \\ G'.\mathtt{mo} = G.\mathtt{mo} \\ w \in G.\mathsf{W}_x \qquad \mathsf{val}_{\mathtt{W}}(w) = v_{\mathtt{R}} \\ w \notin dom(G.\mathtt{mo}\,;\,G.\mathtt{hb}^?\,;\,[\mathsf{E}^\tau]) \end{array}}{G \xrightarrow{\tau, \mathtt{R}(x, v_{\mathtt{R}})}_{\mathrm{opRA}} G'}$$

$$\textsc{rmw}\quad \frac{\begin{array}{c} e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, \mathtt{RMW}(x, v_{\mathtt{R}}, v_{\mathtt{W}})) \\ G'.\mathsf{E} = G.\mathsf{E} \cup \{e\} \\ G'.\mathtt{rf} = G.\mathtt{rf} \cup \{\langle w, e\rangle\} \\ G'.\mathtt{mo} = \mathsf{AddAfter}(G.\mathtt{mo}, G.\mathsf{W}_x, W, e) \\ W = \{w' \in G.\mathsf{W}_x \mid \langle w', w\rangle \in G.\mathtt{mo}^?\} \\ w \in G.\mathsf{W}_x \qquad \mathsf{val}_{\mathtt{W}}(w) = v_{\mathtt{R}} \\ w \notin dom(G.\mathtt{mo}\,;\,G.\mathtt{hb}^?\,;\,[\mathsf{E}^\tau]) \\ w \notin dom(G.\mathtt{rf}\,;\,[\mathrm{RMW}]) \end{array}}{G \xrightarrow{\tau, \mathtt{RMW}(x, v_{\mathtt{R}}, v_{\mathtt{W}})}_{\mathrm{opRA}} G'}$$

The WRITE step adds a corresponding fresh write event e to the graph (placed after all events of thread τ) and extends mo to order the freshly added event w.r.t. all previously added writes to the same location. The extension of mo must respect write-coherence ("local agreement" between mo and hb). Thus, all of e's successors in the new mo order cannot be events of which thread τ is aware. Equivalently, e should be placed as the immediate successor of some event $w = \max_{G.\mathtt{mo}} W$, such that thread τ is not aware of any mo-successors of w ($w \notin dom(G.\mathtt{mo}\,;\,G.\mathtt{hb}^?\,;\,[\mathsf{E}^\tau])$). In addition, for the extension of mo to respect atomicity, the new write e should not intervene between an RMW event and its reads-from source (which, according to atomicity, must be its immediate mo-predecessor). Hence, w cannot be read by an RMW event ($w \notin dom(G.\mathtt{rf}\,;\,[\mathrm{RMW}])$). We note that for the very first write to each location, we must have $W = \emptyset$, in which case we assume that the two conditions on w ($w \notin dom(G.\mathtt{mo}\,;\,G.\mathtt{hb}^?\,;\,[\mathsf{E}^\tau])$ and $w \notin dom(G.\mathtt{rf}\,;\,[\mathrm{RMW}])$) hold by definition.

A READ step by thread τ adds a corresponding fresh read event and justifies it with a reads-from edge. This is exactly as in opWRA, but to capture later writes, instead of using $G.\mathtt{hb}|_{\mathtt{loc}}$ (as per read-coherence), we now use mo (as per weak-read-coherence).

An RMW step is a combination of READ and WRITE. To respect atomicity, it forces the reads-from source of the freshly added RMW event to be its immediate predecessor in the extended mo.

THEOREM 4.8. opRA *characterizes* RA.

PROOF. The proof proceeds exactly as the proof for WRA (Thm. 4.6). Given an RA-consistent execution graph G, one obtains a run of opRA by following any total order extension of $G.\mathtt{hb}$. The preconditions required by each step follow directly from the fact that G is RA-consistent. For the converse, it suffices to note that all reachable states of opRA are RA-consistent execution graphs. Hence, if $\langle \overline{p}, G\rangle$ is reachable in $P \parallel \mathrm{opRA}$, then G is a RA-consistent execution graph that is generated by P with final state $\overline{p}$. □

Strong Release/Acquire. The transitions of opSRA are given by:

$$\textsc{write}\quad \frac{\begin{array}{c} e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, \mathtt{W}(x, v_{\mathtt{W}})) \\ G'.\mathsf{E} = G.\mathsf{E} \cup \{e\} \\ G'.\mathtt{rf} = G.\mathtt{rf} \\ G'.\mathtt{mo} = G.\mathtt{mo} \cup (G.\mathsf{W}_x \times \{e\}) \end{array}}{G \xrightarrow{\tau, \mathtt{W}(x, v_{\mathtt{W}})}_{\mathrm{opSRA}} G'}$$

$$\textsc{read}\quad \frac{\begin{array}{c} e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, \mathtt{R}(x, v_{\mathtt{R}})) \\ G'.\mathsf{E} = G.\mathsf{E} \cup \{e\} \\ G'.\mathtt{rf} = G.\mathtt{rf} \cup \{\langle w, e\rangle\} \\ G'.\mathtt{mo} = G.\mathtt{mo} \\ w \in G.\mathsf{W}_x \qquad \mathsf{val}_{\mathtt{W}}(w) = v_{\mathtt{R}} \\ w \notin dom(G.\mathtt{mo}\,;\,G.\mathtt{hb}^?\,;\,[\mathsf{E}^\tau]) \end{array}}{G \xrightarrow{\tau, \mathtt{R}(x, v_{\mathtt{R}})}_{\mathrm{opSRA}} G'}$$

$$\textsc{rmw}\quad \frac{\begin{array}{c} e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, \mathtt{RMW}(x, v_{\mathtt{R}}, v_{\mathtt{W}})) \\ G'.\mathsf{E} = G.\mathsf{E} \cup \{e\} \\ G'.\mathtt{rf} = G.\mathtt{rf} \cup \{\langle w, e\rangle\} \\ G'.\mathtt{mo} = G.\mathtt{mo} \cup (G.\mathsf{W}_x \times \{e\}) \\ w \in G.\mathsf{W}_x \qquad \mathsf{val}_{\mathtt{W}}(w) = v_{\mathtt{R}} \\ w \notin dom(G.\mathtt{mo}) \end{array}}{G \xrightarrow{\tau, \mathtt{RMW}(x, v_{\mathtt{R}}, v_{\mathtt{W}})}_{\mathrm{opSRA}} G'}$$

A WRITE step by thread τ adds a fresh write event e placed after all events of thread τ and extends mo to order e *after* all existing writes to the same location. The READ is identical to the read step of opRA. The RMW is also similar to the RMW step of opRA, but it must pick w to be the mo-*maximal*

write to the relevant location in the current execution graph. We note that the WRITE and RMW steps in opSRA are deterministic, while the READ step is non-deterministic.

This semantics exploits the fact that hb ∪ mo is acyclic in SRA-consistent execution graphs ("global agreement" between mo and hb, as per strong-write-coherence). Hence, to generate an SRA-consistent execution graph in a run of an operational semantics, we can follow a total order extending hb ∪ mo, which guarantees that writes are executed following their mo-order. In turn, since RMWs should read from their immediate mo-predecessor, we require that RMWs read from the current mo-maximal write. Accordingly, the next theorem is proved as for WRA and RA, using $G.\mathsf{hb} \cup G.\mathsf{mo}$ instead of $G.\mathsf{hb}$ when traversing an SRA-consistent execution graph G.

THEOREM 4.9. opSRA *characterizes* SRA.

4.1 The Reachability Problem for Memory Systems

When an operational semantics for a declarative model X is available (in the form of a memory system as defined above), the X reachability problem (formulated in §2.2) can be stated in more standard terms.

PROPOSITION 4.10. *If a memory system M characterizes a declarative model* X*, then the* X *reachability problem is equivalent to the problem given by:*

Input: *a concurrent program P and a "bad state"* $\overline{p} \in P.\mathsf{Q}$.

Question: *is* $\overline{p}$ *reachable under M (i.e., by Def. 4.4, is* $\overline{p}$ *reachable in the concurrent system* $P \parallel M$ *for some* $m \in M.\mathsf{Q}$*)?*

For the causal models defined above, as mentioned in the introduction to this paper, the challenge in solving this problem stems from the fact that $P \parallel M$ is an *infinite* transition system (since opWRA, opRA and opSRA are all infinite state). This is in contrast to $P \parallel \mathsf{SC}$ (see Ex. 4.2), which is a finite system of size polynomial in the size of P (since SC is of size quadratic in the number of locations and values), thus inducing a PSPACE-complete reachability problem [32].

The reduction of Atig et al. [12] from reachability in lossy FIFO channel machines to reachability under the x86-TSO model holds without any change for WRA, RA, and SRA.

THEOREM 4.11. *For* $\mathsf{X} \in \{\mathsf{WRA}, \mathsf{RA}, \mathsf{SRA}\}$*, the* X *reachability is non-primitive-recursive.*

In fact, it was recently shown that RA reachability is *undecidable* via a delicate reduction from Post correspondence problem [2]. The rest of this paper is devoted to establishing decidability for SRA and WRA. To do so, we use the framework of well-structured transition systems (see §8.1 for a brief reminder). We note that we are unable to directly use opSRA and opWRA in this framework. Roughly speaking, the challenges here stem from: (i) losing parts of the state (the current execution graph) may allow for behaviors that were not allowed without losing this part (e.g., if we discard a write event, we may read overwritten writes); and (ii) naive ordering of partial orders via their induced embedding relation is not a well-quasi order. In the sequel, we overcome these challenges by introducing alternative memory systems for SRA and WRA that are still infinite, but fit well in the framework of well-structured transition systems.

5 MAKING STRONG RELEASE/ACQUIRE LOSSY: THE loSRA MEMORY SYSTEM

In this section, we introduce an alternative memory system, which we call loSRA (for "lossy-SRA"). Later, we will establish the equivalence of loSRA to opSRA, and show how loSRA is used to decide the reachability problem in the framework of well-structured transition systems. We begin with an intuitive discussion to motivate our definitions, and later spell out the formal details.

A memory state of loSRA maintains for each thread a set of "option lists", called the *potential* of the thread, where each (read) option o contains a location $\mathtt{loc}(o)$, a value $\mathtt{val}(o)$ and two other

components that are explained below. Each option list stands for a sequence of possible future reads of the thread, listing the values that it may read in the order that it may read them. For example, the list $o_1 \cdot o_2$ allows the thread to read $\mathtt{val}(o_1)$ from location $\mathtt{loc}(o_1)$ and then $\mathtt{val}(o_2)$ from location $\mathtt{loc}(o_2)$. These lists do not ascribe mandatory continuations, but rather possible futures (hence, *options*). In the beginning, the empty list is assigned to all threads—before any write is executed, no reads are possible (recall that we assume explicit initialization, see Remark 1). In addition, the semantics is designed so that option lists are *"lossy"*, allowing a non-deterministic step that removes arbitrary options from the lists.

The option lists in the potentials dictate the possible READ steps threads can take: for a thread τ to read v from x, an option o with $\mathtt{val}(o) = v$ and $\mathtt{loc}(o) = x$ must be the first in each of τ's lists. Then, to progress to the next option in the list, the thread may consume these options, and discard the first element from each of its lists.

A WRITE step is more involved, encapsulating the requirements of opSRA. First, since opSRA performs write events following their mo-order, when a thread writes to x, it cannot later read the value of x from a write that was already performed (this would violate read-coherence in terms of SRA). Accordingly, we do not allow a thread to write to x if some read option o with $\mathtt{loc}(o) = x$ appears in its potential. Second, when a thread performs a write of v to x, it allows future reads from this write. That is, new read options o with $\mathtt{loc}(o) = x$ and $\mathtt{val}(o) = v$ may be added to every list of every thread. This makes the write step in loSRA (unlike the one of opSRA) *non-deterministic*—the writer essentially has to "guess" which threads will read from the new write and when.

But, where in the lists should we allow to add such options? The following examples demonstrate two possible cases. We write in them o_x^v for a read option of value v from location x.

Example 5.1. Consider the IRIW program with its (SRA-allowed) outcome in Ex. 3.5. Clearly, the first step may only be a write by T_1 or T_4. Suppose, w.l.o.g., that T_1 begins. Since T_3 reads 0 from x, an option o_{x}^0 should be added in the lists of T_3. Now, before reading 0 from x, T_3 has to read 1 from y. Hence, when T_4 writes 1 to y, an option o_{y}^1 should be placed *before* o_{x}^0 in the lists of T_3.

Example 5.2. Consider the MP program with its outcome in Ex. 3.4. It is forbidden under SRA, and so we need to *avoid* the following scenario: First, T_1 writes 0 to x and adds a corresponding option o_{x}^0 to the (initially empty) list of T_2, and then writes 1 to x without adding any option to any list (no thread reads 1 from x in this program outcome). Then, T_1 further writes 1 to y and adds a corresponding option o_{y}^1 in the list of T_1 placed *before* o_{x}^0. Finally, T_2 may run: read 1 from y (consuming o_{y}^1) and then 0 from x (consuming o_{x}^0).

How can we resolve the tension between the two examples? The restriction we impose on the positions of the added read options is based on the following key observation:[7]

Shared-memory causality principle: *After thread π reads from a certain write executed by thread τ, thread π can perform a sequence of operations only if thread τ could perform the same sequence immediately after it executed the write.*

Indeed, if thread τ has just performed a write w, then after thread π reads from w, it "synchronizes" with τ and it is thus confined by the sequences of reads that τ may perform. (Note that the converse does not hold: thread τ may be able to read values that thread π cannot read anymore, since thread π may be already aware of later writes to other locations.) Hence, to allow the addition of a read option o in certain positions of a list L of some thread π, we require a *justification*: the suffix of L after the first occurrence of o should be a subsequence of an option list of the writing thread τ. This guarantees that after π reads from a write w of τ, it will not be able to read something that τ

[7] A weaker observation, which only considers single reads, was essential for the soundness of OGRA—an Owicki Gries logic for RA introduced in [38].

could not read at the time that it wrote w. (Revisiting Ex. 5.2, the read option o^1_y *cannot* be placed before o^0_x, because T_1 *cannot* have o^0_x in its lists at the point of writing 1 to y.)

Example 5.3. We revisit Ex. 5.1 and show how the weak outcome of the IRIW program (see Ex. 3.5) is obtained in the lossy SRA machine loSRA. One possible way to obtain this outcome is depicted as follows:[8]

$$\{\epsilon\}\,||\,\{\epsilon\}\,||\,\{\epsilon\}\,||\,\{\epsilon\} \xrightarrow[\mathsf{W(x,0)}]{\mathsf{T}_1} \{\epsilon\}\,||\,\{\epsilon\}\,||\,\{o^0_x\}\,||\,\{o^0_x\} \xrightarrow[\mathsf{W(x,1)}]{\mathsf{T}_1} \{\epsilon\}\{o^1_x\}\{o^0_x\}\{o^0_x\} \xrightarrow[\mathsf{W(y,0)}]{\mathsf{T}_4} \{\epsilon\}\{o^1_x o^0_y\}\{o^0_x\}\{o^0_x\} \xrightarrow[\mathsf{W(y,1)}]{\mathsf{T}_4}$$
$$\{\epsilon\}\{o^1_x o^0_y\}\{o^1_y o^0_x\}\{o^0_x\} \xrightarrow[\mathsf{R(x,1)}]{\mathsf{T}_2} \{\epsilon\}\{o^0_y\}\{o^1_y o^0_x\}\{o^0_x\} \xrightarrow[\mathsf{R(y,0)}]{\mathsf{T}_2} \{\epsilon\}\{\epsilon\}\{o^1_y o^0_x\}\{o^0_x\} \xrightarrow[\mathsf{R(y,1)}]{\mathsf{T}_3} \{\epsilon\}\{\epsilon\}\{o^0_x\}\{o^0_x\} \xrightarrow[\mathsf{R(x,0)}]{\mathsf{T}_3} \{\epsilon\}\{\epsilon\}\{\epsilon\}\{o^0_x\}$$

Initially, all potentials are empty. Then, T_1 performs its write $\mathsf{x} := 0$ and adds new options in the list of T_3 and T_4 (the first will be used later, since we want T_3 to read 0 from x, and the second will be needed as a justification for $\mathsf{y} := 1$ by T_4). Then, T_1 completes its code by executing $\mathsf{x} := 1$ and adding a read option in T_2's list. Now, T_4 performs its two writes: $\mathsf{y} := 0$ adds a read option in the end of T_2's list, and $\mathsf{y} := 1$ adds a read option in the *beginning* of T_3's list. The latter adds a read option before existing ones (o^1_y is positioned before o^0_x), and thus requires a justification: T_4 (the writing thread) should itself have the option o^0_x at this stage. Indeed, o^0_x appears in the potential of T_4 (it was added by T_1). Finally, we can execute all reads, each consumes the corresponding read option. In the end, o^0_x is left in the potential of T_4, which has no effect (and it is possible to remove it by taking a non-deterministic step that loses some parts of the potentials).

Now, since the potential of thread τ is used both for (i) dictating future reads of τ, and (ii) justifying placement of read options that are generated by τ's write steps, we may need more than one option list for each thread. We also allow to discard existing lists in silent moves of the memory system. This is demonstrated in the following example.

Example 5.4. Consider the following program, whose annotated outcome is allowed under SRA:

$\mathsf{x} := 0$ $\mathsf{x} := 1$ $\mathsf{a}_1 := \mathsf{z}\ /\!/\ 1$ $\mathsf{a}_2 := \mathsf{y}\ /\!/\ 0$	$\mathsf{y} := 0$ $\mathsf{y} := 1$ $\mathsf{b}_1 := \mathsf{x}\ /\!/\ 1$ $\mathsf{b}_2 := \mathsf{z}\ /\!/\ 0$	$\mathsf{z} := 0$ $\mathsf{z} := 1$ $\mathsf{c}_1 := \mathsf{y}\ /\!/\ 1$ $\mathsf{c}_2 := \mathsf{x}\ /\!/\ 0$	$\mathsf{d}_1 := \mathsf{x}\ /\!/\ 1$ $\mathsf{d}_2 := \mathsf{y}\ /\!/\ 1$ $\mathsf{d}_3 := \mathsf{z}\ /\!/\ 0$	$\mathsf{e}_1 := \mathsf{y}\ /\!/\ 1$ $\mathsf{e}_2 := \mathsf{z}\ /\!/\ 1$ $\mathsf{e}_3 := \mathsf{x}\ /\!/\ 0$	$\mathsf{f}_1 := \mathsf{z}\ /\!/\ 1$ $\mathsf{f}_2 := \mathsf{x}\ /\!/\ 1$ $\mathsf{f}_3 := \mathsf{y}\ /\!/\ 0$

✓ SRA

Suppose that it can be obtained by the memory system outlined above when it is restricted to *one* option list per thread (i.e., singleton potentials). Suppose, w.l.o.g., that $\mathsf{z} := 1$ is the last write performed in the execution. Later, T_3 has to read 1 from y and 0 from x. Hence, its option list must include o^1_y and o^0_x in this order. In addition, a read option o^1_z should be placed in T_6's list before $o^1_x \cdot o^0_y$. The justification for it requires $o^1_x \cdot o^0_y$ to be a subsequence of T_3's list. This implies that T_3's list should contain some interleaving of $o^1_y \cdot o^0_x$ and $o^1_x \cdot o^0_y$. But, no such interleaving is a possible future for T_3 (and thus cannot be generated by loSRA): reading o^1_y does not allow T_3 to read o^0_y later; and reading o^1_x does not allow T_3 to read o^0_x later. By allowing more than one option list per thread, we can have $o^1_y \cdot o^0_x$ and $o^1_x \cdot o^0_y$ as two *separate* lists in the potential of T_3—both are possible continuations for it after $\mathsf{z} := 1$. Then, after executing $\mathsf{z} := 1$ and placing a new read option o^1_z before $o^1_x \cdot o^0_y$ in the potential of T_6 using the list $o^1_x \cdot o^0_y$ as a justification, T_3 may "lose" the justifying list $o^1_x \cdot o^0_y$, and choose to continue with $o^1_y \cdot o^0_x$ for its own reads.

Another complication arises due to the fact that read options do not uniquely identify write events in the execution graph (this is unavoidable—for the decision procedure, we need the alphabet of read options to be finite):

[8] We adopt the $\cdot \,||\, \ldots \,||\, \cdot$ notation to denote the states of the lossy system. For example, $B_1\,||\,B_2\,||\,B_3\,||\,B_4$ denotes a mapping from Tid that assigns B_i to T_i for $1 \le i \le 4$ and $\{\epsilon\}$ to all other threads in Tid.

Example 5.5. Consider the following program:

$$\begin{array}{|l||l||l||l|}\hline \mathsf{x} := 0 & \mathsf{y} := 0 & \mathsf{a} := \mathsf{z} \;/\!/\; 1 & \mathsf{c} := \mathsf{w} \;/\!/\; 1 \\ \mathsf{x} := 1 & \mathsf{y} := 1 & \mathsf{w} := 1 & \mathsf{d} := \mathsf{y} \;/\!/\; 0 \\ \mathsf{z} := 1 & \mathsf{z} := 1 & \mathsf{b} := \mathsf{x} \;/\!/\; 0 & \\ \hline \end{array} \qquad \text{✗ SRA} \tag{2MP}$$

Its annotated outcome is disallowed under SRA. Indeed, since T_3 reads $\mathsf{x} = 0$ after $\mathsf{z} = 1$, the read of z must read from the write of T_2. But then, after reading $\mathsf{w} = 1$ (from T_3) T_4 cannot read $\mathsf{y} = 0$. However, the semantics described so far allows this outcome as in the following snippet:

$$\{\epsilon\} \,||\, \{\epsilon\} \,||\, \{\epsilon\} \,||\, \{\epsilon\} \xrightarrow[\mathtt{W(x,0)}]{\mathsf{T}_1} \xrightarrow[\mathtt{W(x,1)}]{\mathsf{T}_1} \xrightarrow[\mathtt{W(y,0)}]{\mathsf{T}_2} \xrightarrow[\mathtt{W(y,1)}]{\mathsf{T}_2} \xrightarrow[\mathtt{W(z,1)}]{\mathsf{T}_1} \{o^0_{\mathsf{y}}\} \,||\, \{o^0_{\mathsf{x}}\} \,||\, \{o^0_{\mathsf{x}}, o^1_{\mathsf{z}} o^0_{\mathsf{y}}\} \,||\, \{o^0_{\mathsf{y}}\} \xrightarrow[\mathtt{W(z,1)}]{\mathsf{T}_2}$$

$$\{o^0_{\mathsf{y}}\} \,||\, \{o^0_{\mathsf{x}}\} \,||\, \{o^1_{\mathsf{z}} o^0_{\mathsf{x}}, o^1_{\mathsf{z}} o^0_{\mathsf{y}}\} \,||\, \{o^0_{\mathsf{y}}\} \xrightarrow[\mathtt{R(z,1)}]{\mathsf{T}_3} \{o^0_{\mathsf{y}}\} \,||\, \{o^0_{\mathsf{x}}\} \,||\, \{o^0_{\mathsf{x}}, o^0_{\mathsf{y}}\} \,||\, \{o^0_{\mathsf{y}}\} \xrightarrow[\mathtt{W(w,1)}]{\mathsf{T}_3} \{o^0_{\mathsf{y}}\} \,||\, \{o^0_{\mathsf{x}}\} \,||\, \{o^0_{\mathsf{x}}, o^0_{\mathsf{y}}\} \,||\, \{o^1_{\mathsf{w}} o^0_{\mathsf{y}}\} \ldots$$

What went wrong? The problem arises when T_3 reads 1 from z. At this point it has two possible futures, $o^1_{\mathsf{z}} o^0_{\mathsf{x}}$ and $o^1_{\mathsf{z}} o^0_{\mathsf{y}}$. Since read options, consisting of location and value, do not uniquely identify writes, it may read 1 from z, and remain with both o^0_{x} and o^0_{y}. Now, it uses one of these options to justify the position of o^1_{w} in the list of T_4, and the other for its own read. However, in a single run of opSRA, when reading 1 from z, T_3 must pick which write event to read from, and then, either it cannot read $\mathsf{x} = 0$ or it cannot read $\mathsf{y} = 0$.

To remedy this problem, we make read options to be more informative. Together with location and value, read options also include the thread identifier that performed the write. When a thread writes, it adds options with its own thread identifier in the different lists. For a thread τ to read v from x, a read option o with $\mathtt{val}(o) = v$ and $\mathtt{loc}(o) = x$ and some *unique* writing thread identifier must be the first in every option list of τ. In this example, the two o^1_{z} options will carry different thread identifiers, which forces T_3 to discard one of its lists before reading.

Even with thread identifiers, read options do not uniquely identify write events. Nevertheless, as our proof shows, the ambiguity inside the writing thread does not harm the adequacy of the semantics. Roughly speaking, it can be resolved by picking the po-earliest write event, as reading from it enforces the weakest constraints for the rest of the run.

Finally, RMWs behave like an atomic combination of a read and a write, with a slight adaptation of the above semantics. Recall that in opSRA, an RMW may only read from the mo-maximal write to the relevant location. To achieve this in loSRA, we include an additional flag in read options whose value is either R or RMW. Intuitively, an RMW value means that the read option is set to read from the mo-maximal write. Accordingly, an RMW step may only consume read options marked as RMW. Since WRITE steps to x replace the mo-maximal write to x in the execution graph, they may choose to mark any of the added read options as RMW, but they can only execute when no existing read option (of any thread) from location x is marked as an RMW.

Next, we turn to the formal definitions.

Notation 5.6 (Sequences). We use ϵ to denote the empty sequence. The length of a sequence s is denoted by $|s|$ (in particular $|\epsilon| = 0$). We often identify a sequence s over Σ with its underlying function in $\{1, \ldots, |s|\} \to \Sigma$, and write $s(k)$ for the symbol at position $1 \le k \le |s|$ in s. We write $\sigma \in s$ if the symbol σ appears in s, that is if $s(k) = \sigma$ for some $1 \le k \le |s|$. We use "$\cdot$" for the concatenation of sequences, and lift it to concatenation of sets S_1 and S_2 of sequences in the obvious way ($S_1 \cdot S_2 \triangleq \{s_1 \cdot s_2 \mid s_1 \in S_1, s_2 \in S_2\}$). We identify symbols with sequences of length 1 or their singletons when needed (e.g., in expressions like $\sigma \cdot S$ for $\sigma \in \Sigma$ and a set S of sequences over Σ).

Definition 5.7. *Read options*, *option lists* and *potentials* are defined as follows:

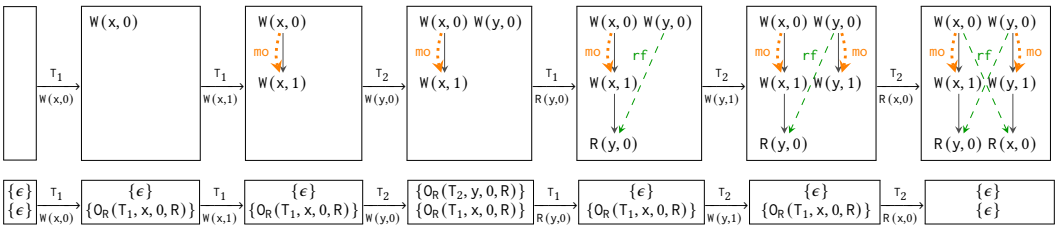


Fig. 4. Illustration of runs of opSRA (top) and loSRA (bottom) for the SB program (Ex. 3.3). In opSRA's states (execution graphs), events of $\mathtt{T}_1$ are on the left and of $\mathtt{T}_2$ on the right. In loSRA's states (a potential for each thread), the potential of $\mathtt{T}_1$ is at the top and of $\mathtt{T}_2$ at the bottom. In this simple example, all option lists consist of at most one option and all potentials are singletons.

(1) A *read option* is a quadruple $o = \mathtt{O}_\mathsf{R}(\tau, x, v, u)$, where $\tau \in \mathsf{Tid}$, $x \in \mathsf{Loc}$, $v \in \mathsf{Val}$ and $u \in \{\mathsf{R}, \mathtt{RMW}\}$. The functions `tid`, `loc`, `val` and `rmw` return the thread identifier (τ), location (x), value (v), and RMW flag (u) of a given read option.
(2) An *option list* L is a sequence of read options.
(3) A *potential* B is a finite non-empty set of option lists.

We define an ordering on option lists, which extends to potentials and to mappings of potentials to threads.

Definition 5.8. The (overloaded) relation $\sqsubseteq$ is defined by:

(1) on option lists: $L \sqsubseteq L'$ if L is a (not necessarily contiguous) subsequence of L';
(2) on potentials: $B \sqsubseteq B'$ if $\forall L \in B.\ \exists L' \in B'.\ L \sqsubseteq L'$ (a.k.a. "Hoare ordering");
(3) on functions from Tid to the set of potentials: $\mathcal{B} \sqsubseteq \mathcal{B}'$ if $\mathcal{B}(\tau) \sqsubseteq \mathcal{B}'(\tau)$ for every $\tau \in \mathsf{Tid}$.

The loSRA memory system is formally defined (in the setting of Def. 4.1) as follows. Figure 4 illustrates a run of loSRA for the SB program (Ex. 3.3) alongside with a corresponding run of opSRA.

Definition 5.9. loSRA is defined by: loSRA.$\mathtt{Q}$ is the set of functions $\mathcal{B}$ assigning a potential to every $\tau \in \mathsf{Tid}$; loSRA.$\mathtt{Q}_0 = \{\lambda\tau \in \mathsf{Tid}.\ \{\epsilon\}\}$;[9] and the transitions are as follows:

$$\textsc{write}\quad \frac{\begin{array}{l}\forall \pi \in \mathsf{Tid}, L' \in \mathcal{B}'(\pi).\ \exists n \geq 0, u_1, \ldots, u_n, L_0, \ldots, L_n.\\ \quad L' = L_0 \cdot \mathtt{O}_\mathsf{R}(\tau, x, v_\mathsf{W}, u_1) \cdot L_1 \cdot \ldots \cdot \mathtt{O}_\mathsf{R}(\tau, x, v_\mathsf{W}, u_n) \cdot L_n\\ \wedge\ L_0 \cdot \ldots \cdot L_n \in \mathcal{B}(\pi)\ \wedge\ L_1 \cdot \ldots \cdot L_n \in \mathcal{B}(\tau)\\ \wedge\ \forall o \in L_1 \cdot \ldots \cdot L_n.\ \mathtt{loc}(o) \neq x\\ \wedge\ \forall o \in L_0.\ \mathtt{loc}(o) = x \implies \pi \neq \tau \wedge \mathtt{rmw}(o) = \mathsf{R}\end{array}}{\mathcal{B} \xrightarrow{\tau, \mathsf{W}(x, v_\mathsf{W})}_{\text{loSRA}} \mathcal{B}'}$$

$$\textsc{rmw}\quad \frac{\begin{array}{c}\mathtt{loc}(o) = x \qquad \mathtt{val}(o) = v_\mathsf{R}\\ \mathtt{rmw}(o) = \mathtt{RMW}\\ \mathcal{B} = \mathcal{B}_{\text{mid}}[\tau \mapsto o \cdot \mathcal{B}_{\text{mid}}(\tau)]\\ \mathcal{B}_{\text{mid}} \xrightarrow{\tau, \mathsf{W}(x, v_\mathsf{W})}_{\text{loSRA}} \mathcal{B}'\end{array}}{\mathcal{B} \xrightarrow{\tau, \mathsf{RMW}(x, v_\mathsf{R}, v_\mathsf{W})}_{\text{loSRA}} \mathcal{B}'}$$

$$\textsc{read}\quad \frac{\mathtt{loc}(o) = x \qquad \mathtt{val}(o) = v_\mathsf{R} \qquad \mathcal{B} = \mathcal{B}'[\tau \mapsto o \cdot \mathcal{B}'(\tau)]}{\mathcal{B} \xrightarrow{\tau, \mathsf{R}(x, v_\mathsf{R})}_{\text{loSRA}} \mathcal{B}'}$$

$$\textsc{lower}\quad \frac{\mathcal{B}' \sqsubseteq \mathcal{B}}{\mathcal{B} \xrightarrow{\varepsilon}_{\text{loSRA}} \mathcal{B}'}$$

The definition of the WRITE step generally follows the intuitive explanation above. (See an illustration in Fig. 5.) Every option list of thread π ($L' \in \mathcal{B}'(\pi)$) after a WRITE transition by thread τ is obtained by adding $n \geq 0$ read options ($\mathtt{O}_\mathsf{R}(\tau, x, v_\mathsf{W}, u_1), \ldots, \mathtt{O}_\mathsf{R}(\tau, x, v_\mathsf{W}, u_n)$) of the current write

[9] To achieve implicit initialization of all locations to 0, one should take loSRA.$\mathtt{Q}_0$ to consist of all functions assigning to each thread sequences consisting of read options of the form $\mathtt{O}_\mathsf{R}(\mathtt{T}_0, x, 0, u)$ where $\mathtt{T}_0$ is a distinguished thread identifier that is not used in programs (corresponds to the initializing thread, see Remark 1).

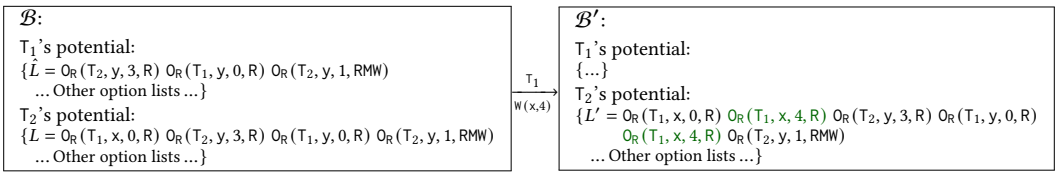


Fig. 5. Illustration of loSRA's WRITE step, as defined in Def. 5.9. Two read options $O_R(T_1, x, 4, R)$ are added to the option list L' of thread T_2 in $\mathcal{B}'$, constructed from the list L of T_2 and justified by the list $\hat{L}$ of T_1 in $\mathcal{B}$. Observe that $\hat{L} = L_1 \cdot L_2$, $L = L_0 \cdot L_1 \cdot L_2$ and $L' = L_0 \cdot O_R(T_1, x, 4, R) \cdot L_1 \cdot O_R(T_1, x, 4, R) \cdot L_2$, where $L_0 = O_R(T_1, x, 0, R)$, $L_1 = O_R(T_2, y, 3, R)\ O_R(T_1, y, 0, R)$ and $L_2 = O_R(T_2, y, 1, RMW)$.

to an existing list L of thread π ($L = L_0 \cdot \ldots \cdot L_n \in \mathcal{B}(\pi)$), provided that: (i) the suffix of the existing list right after the position of the first added option is an option list of the writing thread ($L_1 \cdot \ldots \cdot L_n \in \mathcal{B}(\tau)$); (ii) the list L' cannot have other read options from location x after the first added read option ($\forall o \in L_1 \cdot \ldots \cdot L_n.\ \mathtt{loc}(o) \neq x$); (iii) before the first added read option (i.e., in L_0) thread τ should not have other read options from x (that is, if $\pi = \tau$, then the list L' cannot have any read options from x besides of the newly added ones), and other threads may have read options from x, but these options cannot be RMW options ($\forall o \in L_0.\ \mathtt{loc}(o) = x \implies \pi \neq \tau \wedge \mathtt{rmw}(o) = \mathsf{R}$). When $n = 0$ (no new options are added to some list), we assume that the conditions involving $L_1, \ldots, L_n$ vacuously hold, and thus we only require $L' = L_0 \in \mathcal{B}(\pi)$ (the list is left intact) and $\forall o \in L_0.\ \mathtt{loc}(o) = x \implies \pi \neq \tau \wedge \mathtt{rmw}(o) = \mathsf{R}$. Note that since the universal quantification is on lists of the new state, the step allows to "duplicate" lists before modifying them, as well as to "discard" complete lists (as often useful when a certain list is needed only as a justification for positioning a read option; see, e.g., Ex. 5.5). We also note that several RMW options can be added, but only one of them may be later fulfilled.

Remark 4. Our formal WRITE step insists on having a justification in the form of a complete option list of the writing thread ($L_1 \cdot \ldots \cdot L_n \in \mathcal{B}(\tau)$). It suffices, however, for the suffix after the first added read option to be a *subsequence* of some list of the writing thread ($\{L_1 \cdot \ldots \cdot L_n\} \sqsubseteq \mathcal{B}(\tau)$). Indeed, this less restrictive step is derivable by combining a LOWER step and a WRITE step. For $\pi = \tau$ (adding read options in the lists of the thread that performed the write), this means that no justification is needed (since $L_0 \cdot \ldots \cdot L_n \in \mathcal{B}(\tau)$ implies $\{L_1 \cdot \ldots \cdot L_n\} \sqsubseteq \mathcal{B}(\tau)$). Similarly, no justification is required for placing read options in the end of existing lists (since $\{\epsilon\} \sqsubseteq \mathcal{B}(\tau)$ always holds).

The READ step requires that the first option in all lists in the executing thread's potential are the same read option o, and allows the thread to read the value of o from the location of o, while consuming o from all these lists. Note that, by definition, the potential $\mathcal{B}'(\tau)$ is non-empty, and so the set $\mathcal{B}(\tau)$ as defined in the step is non-empty. When all options are consumed, τ's potential consists of a single empty list.

Remark 5. Our formal READ step always discards the first option from the lists, which was used to justify the read. An alternative semantics that keeps the lists unchanged in read steps (allowing to discard the first option using the LOWER step) would be equivalent. Indeed, the write step that added the consumed option could always add multiple identical consecutive read options.

The RMW step is an atomic sequencing of READ and WRITE to the same location. The READ part can only be performed provided that the first option in all lists is marked with RMW.

The LOWER transition allows to remove read options, as well as full option lists, at any point. It also allows to add new lists, provided that each new list is "at most as powerful" as some existing

list (as used in Remark 4). Intuitively, LOWER can only reduce the possible traces, while it allows us to show that loSRA is a well-structured transition system.

Example 5.10. Consider the 2+2W program with its (SRA-disallowed) outcome in Ex. 3.6. To see that this outcome cannot be obtained by loSRA, consider the last write executed in a run of this program. Suppose, w.l.o.g., that it is $\mathsf{y} := 2$ by T_1. After executing this write, T_1 cannot have any *other* read options of location y in its lists. Hence, a read option of the form $\mathsf{O}_\mathsf{R}(\_, \mathsf{y}, 1, \_)$ should be added to T_1's potential *after* T_1 executed $\mathsf{y} := 2$. This contradicts our assumption that $\mathsf{y} := 2$ was the last executed write.

Example 5.11. Consider the 2RMW program with its (SRA-disallowed) outcome in Ex. 3.8. To try to obtain this outcome in loSRA, the $\mathsf{x} := 0$ by T_1 must add a read option $\mathsf{O}_\mathsf{R}(\mathsf{T}_1, \mathsf{x}, 0, \mathtt{RMW})$ in both its own list and in a list of T_2. But, the execution of the first RMW, which consumes one of these options, cannot proceed if there is another option marked with RMW. Hence, the second RMW cannot read 0, and this outcome cannot be obtained by loSRA.

We conclude with the equivalence of opSRA and loSRA. We postpone its proof to §7, after we introduce a similar system for WRA.

THEOREM 5.12. *For every program P, the set of program states that are reachable under* opSRA *coincides with the set of program states that are reachable under* loSRA.

6 MAKING WEAK RELEASE/ACQUIRE LOSSY: THE loWRA MEMORY SYSTEM

As we did for SRA, we introduce an alternative memory system, which we call loWRA (for "lossy-WRA"), that is equivalent to opWRA. Like loSRA, the loWRA memory system is based on thread potentials, where machine states record information on what can be done from now on, rather than on what was done until now, as in opSRA and opWRA. The causality constraints are maintained by adhering to the "shared-memory causality principle" (see §5), thus requiring appropriate justifications for the positioning of added read options in other threads' lists. However, as we explain below loWRA requires a key change w.r.t. loSRA that has to do with what thread potentials consist of.

The first observation about loWRA is that it must *allow* existing read options from location x to appear in the potential of thread τ after τ writes to x. Indeed, this is necessary for allowing certain outcomes that loSRA forbids (e.g., Examples 3.6 and 3.7). Intuitively speaking, the fact that thread τ writes to x should not restrict the thread from later reading from a write that was executed before τ's write, as long as τ is not already aware of the other write at the point of writing. This is in contrast to SRA, where writes can be executed following their mo order, and thus writing to some location makes the thread aware of the latest write.

We further observe that simply allowing read options from x in write steps to x as mentioned above would make the semantics overly weak. First, if τ writes to x and adds read options $o_1, \ldots, o_n$ in one of its own lists, it should not write again to x before consuming (or discarding) each of $o_1, \ldots, o_n$. Indeed, if τ writes to x again, then the second write is aware (via $\mathsf{po} \subseteq \mathsf{hb}$) of the first one, and reading from the first one after executing the second would violate weak-read-coherence. Second, if $o_1, \ldots, o_n$ are added in a list of another thread π, then after consuming o_1 but before consuming o_n thread π should not write to x. Indeed, performing the read specified by o_1 will make thread π aware of the write w associated with $o_2, \ldots, o_n$. When π writes to x, its write will be hb-after w, and reading again from w will violate weak-read-coherence.

Thus, we need to put certain limitations on the ability to write to a location x that are related to the read options from x in the potentials. The key idea is that such restrictions can be supported by setting the potentials of loWRA to include *write options* in addition to read options. Write options take the form $\mathsf{O}_\mathsf{W}(x)$ where $x \in \mathsf{Loc}$. In the initial states, all lists consist solely of write options (to

some locations), which reflect the initial possible continuations of each thread. Then, when τ writes to x, it (1) has to discard all of its lists that do not begin with $\mathtt{O_W}(x)$, and consume the $\mathtt{O_W}(x)$ option from the head of each of its remaining lists; (2) cannot place read options in its own lists after some $\mathtt{O_W}(x)$ option; and (3) cannot place new read options in other threads' lists in a way that will make some $\mathtt{O_W}(x)$ option appear between two of the added read options. The "shared-memory causality principle" now applies not only to read options, but also to write options: if τ has just performed a write w, then after π reads from w, it "synchronizes" with τ, and so its continuations (sequences of both reads and *writes*) should all be possible continuations of τ. In fact, as our correspondence proofs show, enforcing the "shared-memory causality principle" and conditions (1)-(3) above suffices to precisely capture WRA.

Example 6.1. The annotated outcome of the WW program in Ex. 3.7 can be obtained with the following run (using subscripts and superscripts for locations and values while eliding the other components of read options):

$$\{\mathtt{O_W(x)}\} \,||\, \{\mathtt{O_W(x)}\} \xrightarrow[\mathtt{W(x,1)}]{\mathtt{T_1}} \{\epsilon\} \,||\, \{\mathtt{O_W(x)} \cdot o^1_{\mathtt{x}}\} \xrightarrow[\mathtt{W(x,2)}]{\mathtt{T_2}} \{o^2_{\mathtt{x}}\} \,||\, \{o^1_{\mathtt{x}}\} \xrightarrow[\mathtt{R(x,2)}]{\mathtt{T_1}} \{\epsilon\} \,||\, \{o^1_{\mathtt{x}}\} \xrightarrow[\mathtt{R(x,1)}]{\mathtt{T_2}} \{\epsilon\} \,||\, \{\epsilon\}$$

We start with an $\mathtt{O_W(x)}$ option for both threads. Then, $\mathtt{T_1}$ executes its write: consumes its $\mathtt{O_W(x)}$, and adds a read option in the end of $\mathtt{T_2}$'s list. Now, $\mathtt{T_2}$ executes its write: consumes its $\mathtt{O_W(x)}$ and adds a read option in the end of $\mathtt{T_1}$'s list. Finally, both threads perform their reads by consuming the read options from their potentials. Justifications for the writes trivially exist (essentially, no justification is needed for placing a read option in the *end* of some list). Note that in order to obtain this behavior it is crucial to weaken the condition of loSRA: some thread ($\mathtt{T_2}$ in this example) has to write to location $\mathtt{x}$ while it has already an option to read from $\mathtt{x}$ in its potential.

Example 6.2. The annotated outcome of the oscillating program in Ex. 3.7 can be obtained with the following (prefix of) run (using subscripts and superscripts as above):

$$\{\mathtt{O_W(x)}\} \,||\, \{\epsilon\} \,||\, \{\mathtt{O_W(x)}\} \xrightarrow[\mathtt{W(x,1)}]{\mathtt{T_1}} \{\epsilon\} \,||\, \{o^1_{\mathtt{x}} \cdot o^1_{\mathtt{x}}\} \,||\, \{\mathtt{O_W(x)} \cdot o^1_{\mathtt{x}}\} \xrightarrow[\mathtt{W(x,2)}]{\mathtt{T_3}} \{\epsilon\} \,||\, \{o^1_{\mathtt{x}} \cdot o^2_{\mathtt{x}} \cdot o^1_{\mathtt{x}}\} \,||\, \{o^1_{\mathtt{x}}\} \xrightarrow[\mathtt{R(x,1)}]{\mathtt{T_2}} \ldots$$

We start with an $\mathtt{O_W(x)}$ option for $\mathtt{T_1}$ and $\mathtt{T_3}$. Then, $\mathtt{T_1}$ executes its write: consumes its $\mathtt{O_W(x)}$, and adds two read options to $\mathtt{T_2}$ and one to $\mathtt{T_3}$. Now, $\mathtt{T_3}$ executes its write: consumes its $\mathtt{O_W(x)}$ and adds in the list of $\mathtt{T_2}$ a read option *in between* the two options that were added by $\mathtt{T_1}$. This is justified since (after consuming $\mathtt{O_W(x)}$) $\mathtt{T_3}$ has $o^1_{\mathtt{x}}$ in its list. Finally, $\mathtt{T_2}$ can run and perform the three reads.

Example 6.3. We demonstrate why loWRA disallows the annotated outcome of the MP-trans program in Ex. 3.4. The first executed operation must be $\mathtt{x} := 0$ by $\mathtt{T_1}$. Since $\mathtt{T_3}$ reads 0 from $\mathtt{x}$, a corresponding read option $o^0_{\mathtt{x}}$ has to be added to lists of $\mathtt{T_3}$. Then, since $\mathtt{T_3}$ will read 1 from $\mathtt{x}$ (which is written by $\mathtt{T_2}$) before it reads 0, when $\mathtt{T_2}$ executes $\mathtt{x} := 1$, a read option $o^1_{\mathtt{x}}$ has to be added to lists of $\mathtt{T_3}$ and be placed *before* $o^0_{\mathtt{x}}$. The semantics of loWRA requires a justification for placing $o^1_{\mathtt{x}}$ before $o^0_{\mathtt{x}}$: a list of $\mathtt{T_2}$ that contains $\mathtt{O_W(x)}$ and somewhere after it $o^0_{\mathtt{x}}$. Hence, when $\mathtt{T_1}$ executes $\mathtt{x} := 0$, the read option $o^0_{\mathtt{x}}$ should also be added to the lists of $\mathtt{T_2}$ after $\mathtt{O_W(x)}$. Now, since $\mathtt{T_2}$ reads 1 from $\mathtt{y}$ before it executes $\mathtt{x} := 1$, when $\mathtt{T_1}$ executes $\mathtt{y} := 1$, a read option $o^1_{\mathtt{y}}$ has to be added to lists of $\mathtt{T_2}$, and be placed *before* $\mathtt{O_W(x)}$ (which precedes $o^0_{\mathtt{x}}$). In turn, this requires a justification in the form of a list of $\mathtt{T_1}$ that contains $\mathtt{O_W(x)}$ that precedes $o^0_{\mathtt{x}}$. Therefore, when $\mathtt{T_1}$ executes $\mathtt{x} := 0$, the read option $o^0_{\mathtt{x}}$ should also be added to the lists of $\mathtt{T_1}$, somewhere *after* $\mathtt{O_W(x)}$, which is disallowed by loWRA.

Finally, RMWs in loWRA are handled differently than in loSRA. Indeed, all we have in WRA is that two RMWs never read from the same event, and thus we cannot require, as required in loSRA, that after executing a write, no RMW will read from a write that was executed earlier. Naively, WRA's weak-atomicity constraint could be supported by adding at most one option marked with

RMW when performing a write. This is in contrast, however, with the "shared-memory causality principle": if we decide to give thread π an RMW-option, then later when it reads from a write of thread τ, it may still be able to perform an RMW, while thread τ never had such option. To resolve this mismatch, we utilize the observation in Remark 3, and slightly modify loWRA's read options. Instead of marking read options with RMW flags, we instrument them with *RMW thread identifiers*, denoting the (unique) thread that may consume this option when executing an RMW. When a thread writes, it picks an arbitrary but unique thread identifier to include in this field of its added options; reads ignore this field; and RMWs by thread τ can only consume read options whose RMW thread identifier is τ. Now, in the above scenario, instead of saying that π has some option that τ hasn't, we will have that both threads have the same option, which is a conditional option to perform an RMW if their identifier matches the RMW thread identifier of the option. This allows us to maintain the "shared-memory causality principle".

We turn to the formal definitions. Some notions (e.g., read optionsg) overlap with these of §5. To improve readability, we use the same terms, and the ambiguity is resolved by the context.

Definition 6.4. An *option* o is either $\mathtt{O_R}(\tau, x, v, \pi_{\mathtt{RMW}})$ (*read option*) or $\mathtt{O_W}(x)$ (*write option*), where $\tau, \pi_{\mathtt{RMW}} \in \mathsf{Tid}$, $x \in \mathsf{Loc}$ and $v \in \mathsf{Val}$. The functions $\mathtt{typ}$, $\mathtt{tid}$, $\mathtt{loc}$, $\mathtt{val}$ and $\mathtt{rmw\text{-}tid}$ return (when applicable) the type ($\mathtt{R}/\mathtt{W}$), thread identifier (τ), location (x), value (v), and RMW thread identifier ($\pi_{\mathtt{RMW}}$) of a given option.

Option lists (which now include both read and write options) and potentials, as well as the $\sqsubseteq$ ordering, are defined as in Definitions 5.7 and 5.8 (using Def. 6.4 instead of loSRA's read options).

Definition 6.5. The memory system loWRA is defined by: loWRA.$\mathtt{Q}$ is the set of functions $\mathcal{B}$ assigning a potential to every $\tau \in \mathsf{Tid}$; loWRA.$\mathtt{Q}_0 = \{\mathcal{B} \mid \forall \tau \in \mathsf{Tid}, L \in \mathcal{B}(\tau), o \in L.\ \mathtt{typ}(o) = \mathtt{W}\}$; and the transitions are as follows:

WRITE

$$\frac{\begin{array}{c} o = \mathtt{O_R}(\tau, x, v, \pi_{\mathtt{RMW}}) \\ \forall \pi \in \mathsf{Tid}, L' \in \mathcal{B}'(\pi).\ \exists n \geq 0, L_0, \ldots, L_n. \\ L' = L_0 \cdot o \cdot L_1 \cdot \ldots \cdot o \cdot L_n \ \wedge\ \mathtt{O_W}(x) \cdot L_1 \cdot \ldots \cdot L_n \in \mathcal{B}(\tau) \\ \wedge\ (\pi = \tau \implies \mathtt{O_W}(x) \cdot L_0 \cdot \ldots \cdot L_n \in \mathcal{B}(\tau) \wedge \mathtt{O_W}(x) \notin L_0 \cdot \ldots \cdot L_{n-1}) \\ \wedge\ (\pi \neq \tau \implies L_0 \cdot \ldots \cdot L_n \in \mathcal{B}(\pi) \wedge \mathtt{O_W}(x) \notin L_1 \cdot \ldots \cdot L_{n-1}) \end{array}}{\mathcal{B} \xrightarrow{\tau, \mathtt{W}(x, v_{\mathtt{W}})}_{\text{loWRA}} \mathcal{B}'}$$

RMW

$$\frac{\begin{array}{c} \mathtt{loc}(o) = x \qquad \mathtt{val}(o) = v_{\mathtt{R}} \\ \mathtt{rmw\text{-}tid}(o) = \tau \\ \mathcal{B} = \mathcal{B}_{\text{mid}}[\tau \mapsto o \cdot \mathcal{B}_{\text{mid}}(\tau)] \\ \mathcal{B}_{\text{mid}} \xrightarrow{\tau, \mathtt{W}(x, v_{\mathtt{W}})}_{\text{loWRA}} \mathcal{B}' \end{array}}{\mathcal{B} \xrightarrow{\tau, \mathtt{RMW}(x, v_{\mathtt{R}}, v_{\mathtt{W}})}_{\text{loWRA}} \mathcal{B}'}$$

READ

$$\frac{\mathtt{loc}(o) = x \qquad \mathtt{val}(o) = v_{\mathtt{R}} \qquad \mathcal{B} = \mathcal{B}'[\tau \mapsto o \cdot \mathcal{B}'(\tau)]}{\mathcal{B} \xrightarrow{\tau, \mathtt{R}(x, v_{\mathtt{R}})}_{\text{loWRA}} \mathcal{B}'}$$

LOWER

$$\frac{\mathcal{B}' \sqsubseteq \mathcal{B}}{\mathcal{B} \xrightarrow{\varepsilon}_{\text{loWRA}} \mathcal{B}'}$$

The READ, RMW and LOWER steps are as in loSRA (except for the precondition $\mathtt{rmw\text{-}tid}(o) = \tau$ instead of $\mathtt{rmw}(o) = \mathtt{RMW}$ in the RMW step).

The WRITE step follows the intuitive explanation above. Keeping in mind that the writing thread consumes a write option to the written location, every option list after the WRITE transition is obtained from some previous list ($\mathtt{O_W}(x) \cdot L_0 \cdot \ldots \cdot L_n \in \mathcal{B}(\tau)$ for the writing thread and $L_0 \cdot \ldots \cdot L_n \in \mathcal{B}(\pi)$ for other threads), with the addition of $n \geq 0$ read options of the current write (all with the same RMW thread identifier), provided that: (1) the suffix of the existing list right after the position of the first added read option is an option list (after consuming the first write option) of the writing thread ($\mathtt{O_W}(x) \cdot L_1 \cdot \ldots \cdot L_n \in \mathcal{B}(\tau)$); (2) for the writing thread, the prefix of the existing list (after consuming the first write option) before the position of the last added read option cannot have options to write to x ($\mathtt{O_W}(x) \notin L_0 \cdot \ldots \cdot L_{n-1}$); and (3) for the other threads, the part of the existing list between the first and the last positions of the added read options cannot have options to write to x

$(\mathtt{O_W}(x) \notin L_1 \cdot \ldots \cdot L_{n-1})$. When $n = 0$ for some $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$ (no new options are added to some list), we only require $\mathtt{O_W}(x) \cdot L' \in \mathcal{B}(\tau)$ if $\pi = \tau$, and $L' \in \mathcal{B}(\pi)$ otherwise.

We conclude with the equivalence of opWRA and loWRA. The proof is given in the next section, together with the proof of the corresponding theorem for SRA (Thm. 5.12).

Theorem 6.6. *For every program P, the set of program states that are reachable under* opWRA *coincides with the set of program states that are reachable under* loWRA.

7 EQUIVALENCE OF loSRA AND opSRA AND OF loWRA AND opWRA

In this section, we establish the equivalence of loSRA and opSRA (Thm. 5.12) and of loWRA and opWRA (Thm. 6.6). We use the same approach for both SRA and WRA, while having some different technical arguments for each. Here, we provide the approach and proof sketch, while detailing the full proofs in Appendix A. Whenever possible, we speak of opXRA and loXRA, standing for both opSRA and opWRA, and for both loSRA and loWRA, respectively.

To establish the equivalence of loXRA and opXRA, we define a simulation $\curlyvee \subseteq \text{loXRA.Q} \times \text{opXRA.Q}$, formalizing the intuitive relation between loXRA's potentials and opXRA's execution graphs. For defining $\curlyvee$, we first define a "write list" linking the read options in an option list L to write events in an execution graph G. For loWRA, a write list also has write options that need to be identical to the write options in L; and we also assume a mapping $tid_{\mathtt{RMW}} : \mathsf{W} \to \mathsf{Tid}$ relating every write event to the unique thread that may read from it in an RMW event.

Definition 7.1. A *write list* is a sequence of write events and write options. Let G be an execution graph, L an option list and $tid_{\mathtt{RMW}} : \mathsf{W} \to \mathsf{Tid}$. A write list W is a $\langle G, L\rangle$*-write-list* (for SRA) or a $\langle G, L, tid_{\mathtt{RMW}}\rangle$*-write-list* (for WRA) if $|L| = |W|$ and the following hold for every $1 \le k \le |W|$:

- If $\mathtt{typ}(L(k)) = \mathtt{R}$ (i.e., $L(k)$ is a read option), then the following hold:
  - $W(k) \in G.\mathsf{W}$.
  - $\mathtt{tid}(W(k)) = \mathtt{tid}(L(k))$, $\mathtt{loc}(W(k)) = \mathtt{loc}(L(k))$ and $\mathtt{val_W}(W(k)) = \mathtt{val}(L(k))$.
  - for SRA: if $\mathtt{rmw}(L(k)) = \mathtt{RMW}$, then $W(k) \notin dom(G.\mathtt{mo})$.
  - for WRA: $tid_{\mathtt{RMW}}(W(k)) = \mathtt{rmw\text{-}tid}(L(k))$.
- If $\mathtt{typ}(L(k)) = \mathtt{W}$ (i.e., $L(k)$ is a write option), then $W(k) = L(k)$ (relevant only for WRA).

The following notion of $\langle G, \tau\rangle$*-consistency* of a write list W intuitively means that XRA-consistency is satisfied by the extension of the execution graph G with a sequence of reads and writes of thread τ obtained by following W: For an element $w \in W$ that is in $G.\mathsf{W}$, the corresponding extension of G is a read event reading from w, and for an element of W of the form $\mathtt{O_W}(x)$, the extension of G is a write event to x (writing an arbitrary value).

Ensuring this consistency depends on the constraints of XRA and is thus different for SRA and WRA. For SRA, we should ensure that τ is not already aware of some write that is $\mathtt{mo}$-later than some write of W, and that after reading from a write w_1 of W, thread τ will not become aware of some write that is $\mathtt{mo}$-later than some write w_2 that appears after w_1 in W. Formally:

Definition 7.2. A write list W is $\langle G, \tau\rangle$*-consistent for* SRA if for every $1 \le k \le |W|$, we have $W(k) \notin dom(G.\mathtt{mo} \,;\, G.\mathtt{hb}^? \,;\, [\mathsf{E}^\tau \cup \{W(j) \mid 1 \le j < k\}])$.

For WRA, we should ensure that (1) τ is not already aware of some write that is $\mathtt{hb}|_{\mathtt{loc}}$-later than some write of W; (2) after reading from a write w_1 of W, τ will not become aware of some write that is $\mathtt{hb}|_{\mathtt{loc}}$-later than some write w_2 that appears after w_1 in W; (3) if τ is already aware of some write w to x, then it cannot write to x and then read from w; and (4) if τ is becoming aware of some write w to x by reading from a write (not necessarily to x), it cannot later write to x and then read from w. In the following definition, the first two properties are covered by condition C1, and the third and the fourth by conditions C2 and C3, respectively:

Definition 7.3. A write list W is $\langle G, \tau\rangle$-*consistent for* WRA if for every $1 \le k \le |W|$ with $W(k) \in \mathsf{E}$:

C1 $W(k) \notin dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau \cup \{W(j) \mid 1 \le j < k\}])$.
C2 If $W(i) = \mathtt{O_W}(\mathtt{loc}(W(k)))$ for some $i < k$, then $W(k) \notin dom(G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$.
C3 For every $j < k$, if $W(i) = \mathtt{O_W}(\mathtt{loc}(W(k)))$ for some $j < i < k$, then $\langle W(k), W(j)\rangle \notin G.\mathsf{hb}^?$.

Now, $\curlyvee$ relates an loXRA state $\mathcal{B}$ with an execution graph G if each option list in $\mathcal{B}$ has an appropriate write list. For loWRA, we require in addition the existence of a mapping $tid_{\mathsf{RMW}} : \mathsf{W} \to$ Tid relating every write event to the unique thread that may read from it in an RMW event, and enforce that tid_{RMW} matches the execution graph G (the last requirement in the following definition).

Definition 7.4. A state $\mathcal{B} \in$ loXRA.Q *matches* an execution graph G, denoted by $\mathcal{B} \curlyvee G$, if:

- For SRA: for every $\tau \in$ Tid and $L \in \mathcal{B}(\tau)$, there exists a $\langle G, \tau\rangle$-consistent $\langle G, L\rangle$-write-list.
- For WRA: there exists a function $tid_{\mathsf{RMW}} : \mathsf{W} \to$ Tid, such that the following hold:
  - For every $\tau \in$ Tid and $L \in \mathcal{B}(\tau)$, there exists a $\langle G, \tau\rangle$-consistent $\langle G, L, tid_{\mathsf{RMW}}\rangle$-write-list.
  - For every $\langle w, e\rangle \in G.\mathsf{rf} ; [\mathsf{RMW}]$, we have $\mathtt{tid}(e) = tid_{\mathsf{RMW}}(w)$.

Equipped with these definitions, we show that every trace of opXRA is a trace of loXRA, and vice versa. In one direction we will show a forward simulation from loXRA to opXRA and for the other direction a backward simulation. Notice that ε-transitions (using LOWER) do not affect reachability of program states, and thus the trace equivalence ignores ε-transitions.

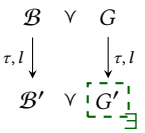
Definition 7.5. Two traces are *equivalent* if their restrictions to non ε-transitions are equal.

The next theorem encompasses the right-to-left direction of both Thm. 5.12 and Thm. 6.6.

LEMMA 7.6. *For every trace of* loXRA *there is an equivalent trace of* opXRA.

PROOF SKETCH. We show that $\curlyvee$ constitutes a forward simulation relation from loXRA to opXRA. First, the initial states clearly match: For SRA: we clearly have $(\lambda\tau \in \mathrm{Tid}.\ \{\epsilon\}) \curlyvee G_\emptyset$. For WRA: for every $\mathcal{B} \in \mathrm{loWRA.Q}_0$ we have $\mathcal{B} \curlyvee G_\emptyset$, since (using any function $tid_{\mathsf{RMW}} : \mathsf{W} \to$ Tid) for every $\tau \in$ Tid and $L \in \mathcal{B}(\tau)$, L itself, having only write options, is a $\langle G, \tau\rangle$-consistent $\langle G, L, tid_{\mathsf{RMW}}\rangle$-write-list, regardless of what G is.

Now, suppose that $\mathcal{B} \curlyvee G$ and $\mathcal{B} \xrightarrow{\tau,l}_{\mathrm{loXRA}} \mathcal{B}'$ for some $\tau \in$ Tid and $l \in$ Lab. We show that there exists G' such that $\mathcal{B}' \curlyvee G'$ and $G \xrightarrow{\tau,l}_{\mathrm{opXRA}} G'$ (as depicted on the right).



Unfolding Def. 7.4:

- For SRA: For every $\pi \in$ Tid and $L \in \mathcal{B}(\pi)$, let $W_{\langle\pi,L\rangle}$ be a $\langle G, \pi\rangle$-consistent $\langle G, L\rangle$-write-list.
- For WRA: Let $tid_{\mathsf{RMW}} : \mathsf{W} \to$ Tid be a function satisfying the conditions of Def. 7.4, and for every $\pi \in$ Tid and $L \in \mathcal{B}(\pi)$, let $W_{\langle\pi,L\rangle}$ be a $\langle G, \pi\rangle$-consistent $\langle G, L, tid_{\mathsf{RMW}}\rangle$-write-list.

Consider the possible cases:

- WRITE step, $l = \mathtt{W}(x, v_\mathsf{W})$: We obtain G' by extending G with the appropriate write event w. Then, using the write lists that exist for G, we construct the required write lists for G'. For this matter, let $\pi \in$ Tid and $L' \in \mathcal{B}'(\pi)$. Let L be the option list in $\mathcal{B}(\pi)$ from which L' is constructed by adding new read options; and L^{justify} be the option list in $\mathcal{B}(\tau)$ that justifies the positioning of the new read options in L'. A write list W' for L' is constructed as follows (we informally identify read options with their indices in the corresponding list, but the intention should be clear):
  - The new read options $\mathtt{O_R}(\tau, x, v_\mathsf{W}, \_)$ in L' (added by loXRA's write step) are all mapped by W' to w (the new write event in G').
  - Every read option o in L' that appears *before* the first added read option $\mathtt{O_R}(\tau, x, v_\mathsf{W}, \_)$ is mapped by W' to the same write it is mapped to by $W_{\langle\pi,L\rangle}$.

– Every other read option o in L' is mapped by W' to the po-maximal write between the write event that o is mapped to by $W_{\langle \pi, L\rangle}$ and the write event that o is mapped to by $W_{\langle \tau, L^{\text{justify}}\rangle}$. Here, we use the fact that both $W_{\langle \pi, L\rangle}$ and $W_{\langle \tau, L^{\text{justify}}\rangle}$ map o to a write event in G of the same thread (which is $\texttt{tid}(o)$). Hence, the two mentioned writes are ordered by po. This construction ensures that both corresponding writes in $W_{\langle \pi, L\rangle}$ and in $W_{\langle \tau, L^{\text{justify}}\rangle}$ have mo (in SRA) or $\mathsf{hb}|_{\mathtt{loc}}$ (in WRA) to the write event that W' maps o to.

Now, it is possible to show that any violation of consistency of W' (Definitions 7.2 and 7.3) entails a violation of consistency of $W_{\langle \pi, L\rangle}$ or of $W_{\langle \tau, L^{\text{justify}}\rangle}$. (For WRA, we use the function $tid_{\mathsf{RMW}}[w \mapsto \pi_{\mathsf{RMW}}]$ where π_{RMW} is the RMW thread identifier of the read options added in the step.) This establishes the required simulation invariant and shows that $\mathcal{B}' \triangledown G'$.

- READ step, $l = \mathsf{R}(x, v_{\mathsf{R}})$: We obtain the graph G' by extending G with the appropriate read event r. As r's reads-from source in G', we have multiple candidates: each option list L in τ's potential in $\mathcal{B}$ has to start with $\mathsf{O}_{\mathsf{R}}(\eta, x, v_{\mathsf{R}}, \_)$ for some (unique) $\eta \in \mathsf{Tid}$ which is mapped by the write list $W_{\langle \tau, L\rangle}$ to some write event w_L of thread η writing v_{R} to x. Among all these writes, we pick the po-minimal one as the reads-from source of r in G'. Using the consistency of the write lists for G (in particular, condition C1 for WRA), we show that opXRA can indeed take the read step from G to G'. In turn, write lists for G' are obtained from those for G in the straightforward way (since no read options are added in this step). Their consistency is again derived from the consistency of the $W_{\langle \tau, L\rangle}$ lists for G. Intuitively speaking, picking the po-minimal write as the reads-from source of r in G', imposes on τ the weakest constraints in G', and allows us to prove the consistency of the new write lists.
- RMW step, $l = \mathsf{RMW}(x, v_{\mathsf{R}}, v_{\mathsf{W}})$: This case is handled by carefully combining the WRITE and READ steps. We note that for SRA, each option list L in τ's potential has to start with $\mathsf{O}_{\mathsf{R}}(\_, x, v_{\mathsf{R}}, \mathsf{RMW})$, and thus, all these options are mapped by the corresponding write lists to the mo-maximal write event to x in G. This ensures that opSRA can take the RMW step.

  In turn, for WRA, we have to show that the reads-from source of the new RMW event in G' is not already read by another RMW. Here, we use the fact that each option list L in τ's potential has to start with $\mathsf{O}_{\mathsf{R}}(\_, x, v_{\mathsf{R}}, \tau)$. Thus, the first write event in each of the corresponding write lists is mapped by tid_{RMW} to τ, which means that it is read by RMWs only by thread τ. Together with C1, this implies that none of these write events is read by an RMW event, and so, opWRA can take the RMW step (see also Remark 3).

Finally, for handling the LOWER step, suppose that $\mathcal{B} \triangledown G$ and $\mathcal{B} \xrightarrow{\varepsilon}_{\text{loXRA}} \mathcal{B}'$. To see that $\mathcal{B}' \triangledown G$, we adapt the write lists that exist for $\mathcal{B}$ to "skip" on all indices that were removed by the lower transition. That is, if f is an increasing function such that $L'(k) = L(f(k))$ where $L' \in \mathcal{B}'(\tau)$ and $L \in \mathcal{B}(\tau)$, we derive a write list W' for L' from the write list $W_{\langle \tau, L\rangle}$ by setting: $W' = \lambda k.\ W_{\langle \tau, L\rangle}(f(k))$. Then, the required properties of W' follow from the corresponding properties of W. □

For the converses (left-to-right direction of both Thm. 5.12 and Thm. 6.6), we favor *backward* simulation, since loXRA requires to "guess" the future, and without knowing the target state, we cannot construct the next step.

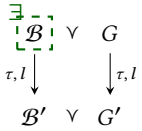
LEMMA 7.7. *For every trace of* opXRA *there is an equivalent trace of* loXRA.

PROOF SKETCH. We show that $\triangledown^{-1}$ constitutes a backward simulation from opXRA to loXRA.[10] The two first requirements of a backward simulation clearly hold for $\triangledown$: (1) $\triangledown^{-1}$ is total, as for every state G of opXRA, we have $(\lambda\tau \in \mathsf{Tid}.\ \{\epsilon\}) \triangledown G$. (2) Consider a state $\mathcal{B}$ of loXRA, such that $\mathcal{B} \triangledown G_{\emptyset}$.

[10]Recall that a backward simulation from an LTS A to an LTS B is a relation $R \subseteq A.\mathsf{Q} \times B.\mathsf{Q}$ such that (1) R is total (for every $q \in A.\mathsf{Q}$, we have $\langle q, p\rangle \in R$ for some $p \in B.\mathsf{Q}$); (2) if $\langle q, p\rangle \in R$ and $q \in A.\mathsf{Q}_0$, then $p \in B.\mathsf{Q}_0$; and (3) if $q \xrightarrow{\sigma}_A q'$ and $\langle q', p'\rangle \in R$, then there exists $p \in B.\mathsf{Q}$ such that $p \xrightarrow{\sigma}_B p'$ and $\langle q, p\rangle \in R$.

By the definition of $\vartriangleright$, it should be possible to link every read option of $\mathcal{B}$ to some write event of $G_\emptyset$. Since there are no events in $G_\emptyset$, there cannot be read options in $\mathcal{B}$, implying that $\mathcal{B} \in \mathsf{loXRA}.\mathtt{Q}_\emptyset$.

We move to the third requirement. Suppose that $G \xrightarrow{\tau,l}_{\mathsf{opXRA}} G'$ and $\mathcal{B}' \vartriangleright G'$. We construct a state $\mathcal{B}$ such that $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loXRA}} \mathcal{B}'$ and $\mathcal{B} \vartriangleright G$ (depicted on the right).



Unfolding Def. 7.4:

- For SRA: For every $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$, let $W'_{\langle \pi, L' \rangle}$ be a $\langle G', \pi \rangle$-consistent $\langle G, L' \rangle$-write-list.
- For WRA: Let $tid_{\mathtt{RMW}} : \mathsf{W} \to \mathsf{Tid}$ be a function satisfying the conditions of Def. 7.4, and for every $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$, let $W'_{\langle \pi, L' \rangle}$ be a $\langle G', \pi \rangle$-consistent $\langle G', L', tid_{\mathtt{RMW}} \rangle$-write-list.

Consider the possible cases:

- WRITE step, $l = \mathtt{W}(x, v_{\mathtt{W}})$: Let w be the write event that is added in thread τ when opXRA moves from G to G'. We obtain $\mathcal{B}$ by:
  - Removing from every option list $L' \in \mathcal{B}'(\pi)$ of every thread π the read options that correspond to the write event w in $W'_{\langle \pi, L' \rangle}$.
  - Adding to $\mathcal{B}(\tau)$ "justifying lists": For every list $L' \in \mathcal{B}'(\pi)$ of every thread π, we add to $\mathcal{B}(\tau)$ a list L that is obtained from L' by taking its suffix from the first read option that corresponds to w in $W'_{\langle \pi, L' \rangle}$, and removing from L all read options that correspond to w in $W'_{\langle \pi, L' \rangle}$.
  - For WRA: we also add a write option $\mathtt{O}_{\mathtt{W}}(x)$ in the beginning of every option list of τ.

  Using the correlation between every option list L' and the write list $W'_{\langle \pi, L' \rangle}$, we are able to show that $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loXRA}} \mathcal{B}'$. It remains to show that $\mathcal{B} \vartriangleright G$, namely that for every thread π and option list $L \in \mathcal{B}(\pi)$, there exists an appropriate write-list W. Since every list $L \in \mathcal{B}(\pi)$ is obtained from some list $L' \in \mathcal{B}'(\pi)$, as described above, we can construct W, by removing from $W'_{\langle \pi, L' \rangle}$ the write events that correspond to the read options that where removed from L' in the process of generating L. (For WRA, we also add a write option $\mathtt{O}_{\mathtt{W}}(x)$ in the beginning of W, in case that $\pi = \tau$.)
- READ step, $l = \mathtt{R}(x, v_{\mathtt{R}})$: Let r be the read event that is added in thread τ when opXRA moves from G to G', and w be the write event from which r reads (i.e., $G'.\mathtt{rf} = G.\mathtt{rf} \cup \{\langle w, r \rangle\}$). Let o be the read option given by $o \triangleq \mathtt{O}_{\mathtt{R}}(\mathtt{tid}(w), x, v_{\mathtt{R}}, \mathtt{R})$ for SRA and $o \triangleq \mathtt{O}_{\mathtt{R}}(\mathtt{tid}(w), x, v_{\mathtt{R}}, tid_{\mathtt{RMW}}(w))$ for WRA. We obtain $\mathcal{B}$ by adding o in the beginning of every option list $L' \in \mathcal{B}'(\tau)$ of the reading thread τ. By definition, we have $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loXRA}} \mathcal{B}'$.

  We show that $\mathcal{B} \vartriangleright G$. For every thread π that is not the reading thread τ, we have that the option lists in $\mathcal{B}(\pi)$ are exactly the same as the lists in $\mathcal{B}'(\pi)$, and we can thus use for them the same write lists as we have for $\mathcal{B}'(\pi)$. Now, every option list $L \in \mathcal{B}(\tau)$ of the reading thread τ is obtained from some option list $L' \in \mathcal{B}'(\tau)$. We define an appropriate write list W, by adding w at the beginning of the write list $W'_{\langle \tau, L' \rangle}$. Following the preconditions of the read step in opXRA, we are able to show that these write lists are indeed consistent.
- RMW step, $l = \mathtt{RMW}(x, v_{\mathtt{R}}, v_{\mathtt{W}})$:

  This case combines the proofs given for the read and write cases. In particular, since we consider backward simulation, we obtain $\mathcal{B}$ by first manipulating $\mathcal{B}'$ into an intermediate state $\mathcal{B}''$ according to the write case, and then manipulating $\mathcal{B}''$ into $\mathcal{B}$ according to the read step. Observe, however, that the condition for performing the RMW step requires from $\mathcal{B}$ more than the condition for performing the READ step: the first read option in all option lists of $\mathcal{B}$ should have the RMW flag for SRA and the same RMW-thread identifier for WRA. We generate $\mathcal{B}$ to meet these requirements by changing the added read option o, as defined above for the READ

step, to have the RMW flag for SRA and the thread τ as the RMW-thread identifier for WRA (since we add the same read option to all lists of $\mathcal{B}$, they share the same RMW-thread identifier). The write lists that witness $\mathcal{B} \vee G$ are also defined by first generating, as in the write case, the write lists that are consistent with respect to $\mathcal{B}''$ and from them, as in the read case, the write lists that are consistent with respect to $\mathcal{B}$. □

8 DECIDABILITY OF THE REACHABILITY PROBLEMS UNDER SRA AND WRA

In this section, we solve the reachability problems under SRA and WRA using the framework of well-structured transition systems. As in §7, whenever possible we speak of XRA, standing for both SRA and WRA.

Given the equivalence between XRA and opXRA (Theorems 4.6 and 4.9) and Prop. 4.10, the reachability problem under the declarative XRA model is reduced to reachability under opXRA. In turn, given the equivalence between opXRA and loXRA (Theorems 5.12 and 6.6), to show the decidability of the reachability problem under XRA, it suffices to establish the decidability of reachability under loXRA. That is, for a concurrent program P and a "bad state" $\overline{p} \in P.\mathtt{Q}$, we need to check whether $\overline{p}$ is reachable (see Def. 4.4) under the memory system loXRA.

To show the decidability of this problem, we use the framework of well-structured transition systems. More precisely, we reduce reachability under loXRA to coverability in a well-structured transition system that meets the conditions ensuring that coverability is decidable.

We start in §8.1 with preliminaries on well-structured transition systems, continue in §8.2 with reformulation of the WRITE step, which will be useful in §8.3, where we conclude with showing that loXRA is indeed a well-structured transition system that admits the required properties.

8.1 Preliminaries on well-structured transition systems

We recall the relevant definitions and propositions about well-structured transition systems. We refer the reader to [8, 25, 51] for a more detailed exposition.

A *well-quasi-ordering* (wqo) on a set S is a reflexive and transitive relation $\preceq$ on S such that for every infinite sequence $s_1, s_2, \ldots$ of elements of S, we have $s_i \preceq s_j$ for some $i < j$. In a context of a set S and a wqo $\preceq$ on S, the *upward closure* of a set $U \subseteq S$, denoted by $\uparrow U$, is given by $\{s \in S \mid \exists u \in U.\ u \preceq s\}$; a set $U \subseteq S$ is called *upward closed* if $U = \uparrow U$; and a set $B \subseteq U$ is called a *basis* of U if $U = \uparrow B$. The properties of a wqo ensure that every upward closed set has a *finite* basis.

A *well-structured transition system* (WSTS) is an LTS A equipped with a wqo $\preceq$ on $A.\mathtt{Q}$ that is *compatible* with A, that is: if $q_1 \rightarrow_A q_2$ and $q_1 \preceq q_3$, then there exists $q_4 \in A.\mathtt{Q}$ such that $q_3 \rightarrow_A^* q_4$ and $q_2 \preceq q_4$. The *coverability problem* for $\langle A, \preceq \rangle$ asks whether an input state $q \in A.\mathtt{Q}$ is coverable, namely: is some state q' with $q \preceq q'$ reachable in A?

Coverability is decidable (see, e.g., [8, 25]) for a WSTS $\langle A, \preceq \rangle$ provided that $\preceq$ is decidable and the following hold:

(i) *effective initialization*: there exists an algorithm that accepts a state $q \in A.\mathtt{Q}$ and decides whether $\uparrow\{q\} \cap A.\mathtt{Q}_0 = \emptyset$.
(ii) *effective pred-basis*: there exists an algorithm that accepts a state $q \in A.\mathtt{Q}$ and returns a finite basis of $\uparrow \mathrm{pred}_A(\uparrow\{q\})$.

Roughly speaking, these conditions ensure that (1) backward reachability analysis from q will converge to a fixed point; (2) each step in its calculation is effective; and (3) we can check whether the fixed point contains an initial state.

8.2 Backwards formulation of the WRITE step

The following alternative formulation of the WRITE step is convenient to use in our proofs. This formulation "works backwards"—choosing read options to omit from the target state for reaching the source state. Each such possibility is an "index choice":

Definition 8.1. An *index choice* for a state $\mathcal{B}' \in \text{loXRA.Q}$ is a function $\mathcal{P}$ assigning a set $\mathcal{P}(\pi, L') \subseteq \{1, \ldots, |L'|\}$ to every $\pi \in \text{Tid}$ and $L' \in \mathcal{B}'(\pi)$. An index choice $\mathcal{P}$ for $\mathcal{B}'$ *supports a* $\langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$*-step*, denoted by $\mathcal{P} \models_{\text{XRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$, if the following hold for some (unique) $\pi_{\mathtt{RMW}} \in \text{Tid}$ (in the case of WRA) and every $\pi \in \text{Tid}$ and $L' \in \mathcal{B}'(\pi)$:

- For every $k \in \mathcal{P}(\pi, L')$:
  - For SRA: $L'(k) \in \{\mathtt{O_R}(\tau, x, v_{\mathtt{W}}, \mathtt{R}), \mathtt{O_R}(\tau, x, v_{\mathtt{W}}, \mathtt{RMW})\}$.
  - For WRA: $L'(k) = \mathtt{O_R}(\tau, x, v_{\mathtt{W}}, \pi_{\mathtt{RMW}})$.
- For every $k \in \{1, \ldots, |L'|\} \setminus \mathcal{P}(\pi, L')$:
  - For SRA: $\mathtt{loc}(L'(k)) \neq x$ whenever at least one of the following hold:
    - $k > p$ for some $p \in \mathcal{P}(\pi, L')$.
    - $\pi = \tau$.
    - $\mathtt{rmw}(L'(k)) = \mathtt{RMW}$.
  - For WRA: $L'(k) \neq \mathtt{O_W}(x)$ whenever at least one of the following hold:
    - $p_1 < k < p_2$ for some $p_1, p_2 \in \mathcal{P}(\pi, L')$.
    - $\pi = \tau$ and $k < p$ for some $p \in \mathcal{P}(\pi, L')$.

The first condition requires that each read option included in the index choice corresponds to a write by thread τ to location x of value $v_{\mathtt{W}}$, and for WRA all the read options should also share the same RMW thread identifier. The second condition requires that, besides the positions in the index choice (i.e., for options that existed before the write step): For SRA—the location x does not appear in any list after the first position in $\mathcal{P}(\pi, L')$; the location x does not appear at all in lists of thread τ; and options to read from x are not RMW options. For WRA—there are no write options to x between two positions in the index choice, and for lists of thread τ there are also no write options to x before the first position in the index choice.

To formulate the justification requirement, we use the following notations:

Notation 8.2 (List filters). For a list L and a set $P \subseteq \{1, \ldots, |L|\}$ of positions in L, we define:

- $L \setminus P$ is the list derived from L by removing from it the positions in P. The mapping of the positions of L that are not in P to their matching positions in $L \setminus P$ is denoted by $\mathsf{Map}_{\langle L,P\rangle}$ (formally, $\mathsf{Map}_{\langle L,P\rangle} \triangleq \lambda k \in \{1, \ldots, |L|\} \setminus P.\ k - |\{j \in P \mid j < k\}|$).
- $L \setminus\!\setminus P$ further removes from L the positions before the first position in P, namely returns the list $L\setminus(P \cup \{1, \ldots, \min(P) - 1\})$ (undefined if $P = \emptyset$). The mapping of the positions of L that are not in P and not before the first position in P to their matching positions in $L \setminus\!\setminus P$ is denoted by $\mathsf{MMap}_{\langle L,P\rangle}$ (formally, $\mathsf{MMap}_{\langle L,P\rangle} \triangleq \lambda k \in \{\min(P), \ldots, |L|\} \setminus P.\ \mathsf{Map}_{\langle L,P\rangle}(k) - \min(P) + 1$).

For example, for the option list of loSRA (used in Fig. 5)

$$L' = \mathtt{O_R}(\mathtt{T_1}, \mathtt{x}, 0, \mathtt{R})\ \mathtt{O_R}(\mathtt{T_1}, \mathtt{x}, 4, \mathtt{R})\ \mathtt{O_R}(\mathtt{T_2}, \mathtt{y}, 3, \mathtt{R})\ \mathtt{O_R}(\mathtt{T_1}, \mathtt{y}, 0, \mathtt{R})\ \mathtt{O_R}(\mathtt{T_1}, \mathtt{x}, 4, \mathtt{R})\ \mathtt{O_R}(\mathtt{T_2}, \mathtt{y}, 1, \mathtt{RMW})$$

and $P = \{2, 5\}$, we have:

- $L' \setminus P = \mathtt{O_R}(\mathtt{T_1}, \mathtt{x}, 0, \mathtt{R})\ \mathtt{O_R}(\mathtt{T_2}, \mathtt{y}, 3, \mathtt{R})\ \mathtt{O_R}(\mathtt{T_1}, \mathtt{y}, 0, \mathtt{R})\ \mathtt{O_R}(\mathtt{T_2}, \mathtt{y}, 1, \mathtt{RMW})$
- $\mathsf{Map}_{\langle L',P\rangle} = [1 \mapsto 1; 3 \mapsto 2; 4 \mapsto 3; 6 \mapsto 4]$
- $L' \setminus\!\setminus P = \mathtt{O_R}(\mathtt{T_2}, \mathtt{y}, 3, \mathtt{R})\ \mathtt{O_R}(\mathtt{T_1}, \mathtt{y}, 0, \mathtt{R})\ \mathtt{O_R}(\mathtt{T_2}, \mathtt{y}, 1, \mathtt{RMW})$
- $\mathsf{MMap}_{\langle L',P\rangle} = [3 \mapsto 1; 4 \mapsto 2; 6 \mapsto 3]$

Definition 8.3. The *source* of $\mathcal{B}'$ w.r.t. a thread τ and an index choice $\mathcal{P}$ for $\mathcal{B}'$, denoted by $\mathrm{src}(\mathcal{B}', \tau, \mathcal{P})$, is given by:

$$\mathrm{src}(\mathcal{B}', \tau, \mathcal{P}) \triangleq \lambda\pi \in \mathsf{Tid}. \begin{cases} \{L' \setminus \mathcal{P}(\pi, L') \mid L' \in \mathcal{B}'(\pi)\} & \pi \neq \tau \\ \{L' \setminus \mathcal{P}(\tau, L') \mid L' \in \mathcal{B}'(\tau)\} \cup & \pi = \tau \\ \{L' \setminus\!\setminus \mathcal{P}(\eta, L') \mid \eta \in \mathsf{Tid}, L' \in \mathcal{B}'(\eta) \text{ such that } \mathcal{P}(\eta, L') \neq \emptyset\} \end{cases}$$

The following proposition follow directly from our definitions.

Proposition 8.4. $\mathcal{B} \xrightarrow{\tau, \mathtt{W}(x, v_\mathtt{W})}_{\mathsf{loXRA}} \mathcal{B}'$ *iff the following hold for some index choice* $\mathcal{P}$ *for* $\mathcal{B}'$*:*

- $\mathcal{P} \models_{\mathsf{XRA}} \langle \tau, \mathtt{W}(x, v_\mathtt{W}) \rangle$.
- $\mathrm{src}(\mathcal{B}', \tau, \mathcal{P})(\pi) \subseteq \mathcal{B}(\pi)$ *for every* $\pi \in \mathsf{Tid} \setminus \{\tau\}$.
- *For* SRA: $\mathrm{src}(\mathcal{B}', \tau, \mathcal{P})(\tau) \subseteq \mathcal{B}(\tau)$.
- *For* WRA: $\mathtt{O}_\mathtt{W}(x) \cdot \mathrm{src}(\mathcal{B}', \tau, \mathcal{P})(\tau) \subseteq \mathcal{B}(\tau)$.

8.3 loXRA as a WSTS

We continue with showing that concurrent systems with loXRA serving as the memory system are well-structured transition systems that satisfy the above requirements.

The $\sqsubseteq$ ordering on the states of loXRA (see Def. 5.8) is clearly decidable and also forms a wqo. Indeed, by Higman's lemma [26], $\sqsubseteq$ is a wqo on the set of all option lists. In turn, its lifting to potentials (which are finite by definition) is a wqo on the set of all potentials (see [51]). Finally, by Dickson's lemma [22], the pointwise lifting of $\sqsubseteq$ to functions assigning a potential to every $\tau \in \mathsf{Tid}$ (i.e., states of loXRA) is also a wqo.

Now, let P be a program. The $\sqsubseteq$ ordering is naturally lifted to states of the concurrent system $P \parallel \mathsf{loXRA}$ (that is, pairs $\langle \overline{p}, \mathcal{B} \rangle \in P.\mathtt{Q} \times \mathsf{loXRA}.\mathtt{Q}$; see Def. 4.3) by defining $\langle \overline{p}, \mathcal{B} \rangle \sqsubseteq \langle \overline{p}', \mathcal{B}' \rangle$ iff $\overline{p} = \overline{p}'$ and $\mathcal{B} \sqsubseteq \mathcal{B}'$. We show next that $P \parallel \mathsf{loXRA}$ equipped with $\sqsubseteq$ is indeed a WSTS that admits the required conditions for having a decidable coverability problem.

Lemma 8.5. $\langle P \parallel \mathsf{loXRA}, \sqsubseteq \rangle$ *is a WSTS that admits effective initialization and effective pred-basis.*

Proof.

- $\sqsubseteq$ is compatible with $P \parallel \mathsf{loXRA}$: First, since $P.\mathtt{Q}$ is (by definition) finite and $\sqsubseteq$ is a wqo on $\mathsf{loXRA}.\mathtt{Q}$, we have that $\sqsubseteq$ is a wqo on $(P \parallel \mathsf{loXRA}).\mathtt{Q}$. Second, since LOWER is explicitly included in loXRA, $\sqsubseteq$ is trivially compatible with $P \parallel \mathsf{loXRA}$. Indeed, given $q_1 = \langle \overline{p}_1, \mathcal{B}_1 \rangle$, $q_2 = \langle \overline{p}_2, \mathcal{B}_2 \rangle$ and $q_3 = \langle \overline{p}_3, \mathcal{B}_3 \rangle$ such that $q_1 \rightarrow_{P \parallel \mathsf{loXRA}} q_2$ and $q_1 \sqsubseteq q_3$ (so $\overline{p}_1 = \overline{p}_3$), for $q_4 = q_2$, we have $q_3 \rightarrow^*_{P \parallel \mathsf{loXRA}} q_4$ (since $\mathcal{B}_3 \xrightarrow{\varepsilon}_{\mathsf{loXRA}} \mathcal{B}_1$ using the LOWER step) and $q_2 \sqsubseteq q_4$.
- Effective initialization: $P \parallel \mathsf{loXRA}$ trivially admits effective initialization. Indeed, the states $\langle \overline{p}, \mathcal{B} \rangle$ for which $\uparrow\{\langle \overline{p}, \mathcal{B} \rangle\} \cap (P \parallel \mathsf{loXRA}).\mathtt{Q}_0 \neq \emptyset$ are exactly the initial states themselves—$P.\mathtt{Q}_0 \times \mathsf{loXRA}.\mathtt{Q}_0$.
- Effective pred-basis: To prove that $P \parallel \mathsf{loXRA}$ has effective pred-basis, it suffices to show how to calculate a finite basis Q^α of $\uparrow \mathrm{pred}^\alpha_{\mathsf{loXRA}}(\uparrow\{\mathcal{B}'\})$ for each α of the form $\langle \tau, \mathtt{W}(x, v_\mathtt{W}) \rangle$, $\langle \tau, \mathtt{R}(x, v_\mathtt{R}) \rangle$, $\langle \tau, \mathtt{RMW}(x, v_\mathtt{R}, v_\mathtt{W}) \rangle$ or ε. Then, a finite basis of $\uparrow \mathrm{pred}^\alpha_{P \parallel \mathsf{loXRA}}(\uparrow\{\langle \overline{p}', \mathcal{B}' \rangle\})$ is given by $\mathrm{pred}^\alpha_P(\{\overline{p}'\}) \times Q^\alpha$ for $\alpha \neq \varepsilon$; and by $\{\overline{p}'\} \times Q^\alpha$ for $\alpha = \varepsilon$ (silent memory step). In addition, for a silent program step, a finite basis of $\uparrow \mathrm{pred}^{\langle \tau, \varepsilon \rangle}_{P \parallel \mathsf{loXRA}}(\uparrow\{\langle \overline{p}', \mathcal{B}' \rangle\})$ is given by $\mathrm{pred}^{\langle \tau, \varepsilon \rangle}_P(\{\overline{p}'\}) \times \{\mathcal{B}'\}$.

  Silent memory step The set of predecessors of $\mathcal{B}'$ with respect to a silent memory step (i.e., using LOWER) is very simple—it contains any state $\mathcal{B}$ such that $\mathcal{B}' \sqsubseteq \mathcal{B}$. Thus, $\{\mathcal{B}'\}$ is a finite basis of $\uparrow \mathrm{pred}^\varepsilon_{\mathsf{loXRA}}(\{\mathcal{B}'\})$.

  Read We split the handling of loSRA and loWRA.

– For loSRA: A predecessor $\mathcal{B}$ of $\mathcal{B}'$ with respect to a READ step $\mathcal{B}$ is similar to $\mathcal{B}'$, except for having in each option list of τ an additional first read option o with $\mathtt{loc}(o) = x$ and $\mathtt{val}(o) = v_{\mathsf{R}}$. Hence, for $\alpha = \langle \tau, \mathsf{R}(x, v_{\mathsf{R}}) \rangle$, the set $\{\mathcal{B}'[\tau \mapsto \mathtt{O}_{\mathsf{R}}(\tau_{\mathsf{W}}, x, v_{\mathsf{R}}, u) \cdot \mathcal{B}'(\tau)] \mid \tau_{\mathsf{W}} \in \mathsf{Tid}, u \in \{\mathsf{R}, \mathsf{RMW}\}\}$ is a finite basis of $\uparrow \mathsf{pred}^{\alpha}_{\mathsf{loSRA}}(\{\mathcal{B}'\})$. It is also a basis of $\uparrow \mathsf{pred}^{\alpha}_{\mathsf{loSRA}}(\uparrow\{\mathcal{B}'\})$: For a state $\mathcal{B}''$ with $\mathcal{B}' \sqsubseteq \mathcal{B}''$, a corresponding read option $\mathtt{O}_{\mathsf{R}}(\tau_{\mathsf{W}}, x, v_{\mathsf{R}}, u)$ appears in the lists of τ in $\mathsf{pred}^{\alpha}_{\mathsf{loSRA}}(\{\mathcal{B}''\})$ before some additional read options, ensuring that $\mathsf{pred}^{\alpha}_{\mathsf{loSRA}}(\{\mathcal{B}'\}) \sqsubseteq \mathsf{pred}^{\alpha}_{\mathsf{loSRA}}(\{\mathcal{B}''\})$.
– For loWRA: The calculation is almost the same as for loSRA, with the only difference that for $\alpha = \langle \tau, \mathsf{R}(x, v_{\mathsf{R}}) \rangle$, the set $\{\mathcal{B}'[\tau \mapsto \mathtt{O}_{\mathsf{R}}(\tau_{\mathsf{W}}, x, v_{\mathsf{R}}, \pi_{\mathsf{RMW}}) \cdot \mathcal{B}'(\tau)] \mid \tau_{\mathsf{W}}, \pi_{\mathsf{RMW}} \in \mathsf{Tid}\}$ is a finite basis of $\uparrow \mathsf{pred}^{\langle \tau, \mathsf{R}(x, v_{\mathsf{R}}) \rangle}_{\mathsf{loWRA}}(\{\mathcal{B}'\})$.

Write We construct the basis of the predecessors w.r.t. a WRITE step by considering all (finitely many) possibilities of omitting read options from lists of $\mathcal{B}'$, using Prop. 8.4 and the following technical lemma, which shows that if $\mathcal{B}' \sqsubseteq \mathcal{B}''$ then for every source state $\mathsf{src}(\mathcal{B}'', \tau, \mathcal{P}'')$ of $\mathcal{B}''$ there exists a source state $\mathsf{src}(\mathcal{B}', \tau, \mathcal{P}')$ of $\mathcal{B}'$, such that $\mathsf{src}(\mathcal{B}', \tau, \mathcal{P}') \sqsubseteq \mathsf{src}(\mathcal{B}'', \tau, \mathcal{P}'')$.

LEMMA 8.6. *Let $\mathcal{P}''$ be an index choice for $\mathcal{B}'' \in \mathsf{loXRA.Q}$ such that $\mathcal{P}'' \models_{\mathsf{XRA}} \langle \tau, \mathsf{W}(x, v_{\mathsf{W}}) \rangle$. If $\mathcal{B}' \sqsubseteq \mathcal{B}''$, then $\mathsf{src}(\mathcal{B}', \tau, \mathcal{P}') \sqsubseteq \mathsf{src}(\mathcal{B}'', \tau, \mathcal{P}'')$ for some index choice $\mathcal{P}'$ for $\mathcal{B}'$ such that $\mathcal{P}' \models_{\mathsf{XRA}} \langle \tau, \mathsf{W}(x, v_{\mathsf{W}}) \rangle$.*

PROOF. Since $\mathcal{B}' \sqsubseteq \mathcal{B}''$, for every $\pi \in \mathsf{Tid}$, there exists a function $F_\pi : \mathcal{B}'(\pi) \to \mathcal{B}''(\pi)$ such that for every $L' \in \mathcal{B}'(\pi)$, we have $L' \sqsubseteq F_\pi(L')$, witnessed by a strictly increasing function $f_{\langle \pi, L' \rangle} : \{1, \ldots, |L'|\} \to \{1, \ldots, |F_\pi(L')|\}$, such that $L'(k) = (F_\pi(L'))(f_{\langle \pi, L' \rangle}(k))$ for every $k \in \{1, \ldots, |L'|\}$.
We define $\mathcal{P}'$ to be the positions in $\mathcal{P}''$ that originated in $\mathcal{B}'$, according to the $f_{\langle \pi, L' \rangle}$ functions. That is,

$$\mathcal{P}' \triangleq \lambda \pi \in \mathsf{Tid}, L' \in \mathcal{B}'(\pi). \{k \in \{1, \ldots, |L'|\} \mid f_{\langle \pi, L' \rangle}(k) \in \mathcal{P}''(\pi, F_\pi(L'))\}.$$

It is easy to verify that $\mathcal{P}'$ supports a $\langle \tau, \mathsf{W}(x, v_{\mathsf{W}}) \rangle$-step. Let $\mathcal{B}'_0 = \mathsf{src}(\mathcal{B}', \tau, \mathcal{P}')$. We show that $\mathcal{B}'_0 \sqsubseteq \mathsf{src}(\mathcal{B}'', \tau, \mathcal{P}'')$.
Recall that for every thread $\pi \in \mathsf{Tid}$, we have that every list $L'_0 \in \mathcal{B}'_0(\pi)$ is equal to $L' \setminus \mathcal{P}'(\pi, L')$ (or resp. to $L' \setminus\!\setminus \mathcal{P}'(\eta, L')$) for some list L' of $\mathcal{B}'(\pi)$ (resp. for some list L' of $\mathcal{B}'(\eta)$ for some $\eta \in \mathsf{Tid}$ with $\mathcal{P}'(\eta, L') \neq \emptyset$). Hence, we can define a function $H_\pi : \mathcal{B}'_0(\pi) \to \mathsf{src}(\mathcal{B}'', \tau, \mathcal{P}'')(\pi)$, by setting $H_\pi(L'_0) = F_\pi(L') \setminus \mathcal{P}''(\pi, F_\pi(L'))$. Observe that for every $L'_0 \in \mathcal{B}'_0(\pi)$, we have $L'_0 \sqsubseteq H_\pi(L'_0)$, witnessed by the function $h_{\langle \pi, L'_0 \rangle} : \{1, \ldots, |L'_0|\} \to \{1, \ldots, |H_\pi(L'_0)|\}$ that adapts $f_{\langle \pi, L' \rangle}$ to the positions of L'_0 that originated from L'. Namely, for every $k \in \{1, \ldots, |L'_0|\}$, the value of $h_{\langle \pi, L'_0 \rangle}(k)$ is the position in $H_\pi(L'_0)$ that corresponds (according to $\mathcal{P}''$) to the position in $F_\pi(L')$ that is the value of $f_{\langle \pi, L' \rangle}$ on the position in L' that corresponds (according to $\mathcal{P}'$) to k. Formally,

$$h_{\langle \pi, L'_0 \rangle}(k) \triangleq \mathsf{Map}_{\langle F_\pi(L'), \mathcal{P}''(\pi, F_\pi(L')) \rangle}(f_{\langle \pi, L' \rangle}(\mathsf{Map}^{-1}_{\langle L', \mathcal{P}'(\pi, L') \rangle}(k))).$$

(Respectively, we define $H_\pi(L'_0) = F_\eta(L') \setminus\!\setminus \mathcal{P}''(\eta, F_\eta(L')))$, witnessed analogously.) □

By Prop. 8.4 and Lemma 8.6, we get a finite basis of $\uparrow \mathsf{pred}^{\langle \tau, \mathsf{W}(x, v_{\mathsf{W}}) \rangle}_{\mathsf{loXRA}}(\uparrow\{\mathcal{B}'\})$, given by:
– For SRA: $\{\mathsf{src}(\mathcal{B}', \tau, \mathcal{P}) \mid \mathcal{P} \in S_{\mathsf{SRA}}(\mathcal{B}', \tau, x, v_{\mathsf{W}})\}$
– For WRA: $\{\mathsf{src}(\mathcal{B}', \tau, \mathcal{P})[\tau \mapsto \mathtt{O}_{\mathsf{W}}(x) \cdot \mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\tau)] \mid \mathcal{P} \in S_{\mathsf{WRA}}(\mathcal{B}', \tau, x, v_{\mathsf{W}})\}$
where: $S_{\mathsf{XRA}}(\mathcal{B}', \tau, x, v_{\mathsf{W}}) = \{\mathcal{P} \mid \mathcal{P}$ is an index choice for $\mathcal{B}'$ such that $\mathcal{P} \models_{\mathsf{XRA}} \langle \tau, \mathsf{W}(x, v_{\mathsf{W}}) \rangle\}$.
Indeed, Prop. 8.4 provides the direct correspondence between the source states and predecessor states of $\mathcal{B}'$; the left upward closure of $\uparrow \mathsf{pred}^{\langle \tau, \mathsf{W}(x, v_{\mathsf{W}}) \rangle}_{\mathsf{loXRA}}(\uparrow\{\mathcal{B}'\})$ preserves the equivalence, since a finite basis refers by definition to an upward closed set; and Lemma 8.6 shows that the equivalence holds also with the right upward closure: if $\mathcal{B}' \sqsubseteq \mathcal{B}''$ then for every

source state $\mathsf{src}(\mathcal{B}'', \tau, \mathcal{P}'')$ of $\mathcal{B}''$ there exists a source state $\mathsf{src}(\mathcal{B}', \tau, \mathcal{P}')$ of $\mathcal{B}'$, such that $\mathsf{src}(\mathcal{B}', \tau, \mathcal{P}') \sqsubseteq \mathsf{src}(\mathcal{B}'', \tau, \mathcal{P}'')$.

RMW The predecessor with respect to an RMW step intuitively combines the predecessors with respect to the read and write steps. By Prop. 8.4 and Lemma 8.6, we get that the following is a finite basis of $\uparrow\mathsf{pred}^{\langle \tau, \mathtt{RMW}(x, v_R, v_W)\rangle}_{\mathsf{loXRA}}(\uparrow\{\mathcal{B}'\})$:

– For SRA: $\{\mathsf{src}(\mathcal{B}', \tau, \mathcal{P})[\tau \mapsto \mathsf{O}_R(\tau_W, x, v_R, \mathtt{RMW}) \cdot \mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\tau)] \mid \mathcal{P} \in S_{\mathsf{SRA}}(\mathcal{B}', \tau, x, v_W)\}$

– For WRA: $\{\mathsf{src}(\mathcal{B}', \tau, \mathcal{P})[\tau \mapsto \mathsf{O}_R(\tau_W, x, v_R, \tau) \cdot \mathsf{O}_W(x) \cdot \mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\tau)] \mid \mathcal{P} \in S_{\mathsf{WRA}}(\mathcal{B}', \tau, x, v_W)\}$

where: $S_{\mathsf{XRA}}(\mathcal{B}', \tau, x, v_W) = \{\mathcal{P} \mid \mathcal{P}$ is an index choice for $\mathcal{B}'$ such that $\mathcal{P} \models_{\mathsf{XRA}} \langle \tau, \mathtt{W}(x, v_W)\rangle\}$. □

It is now easy to establish the decidability of reachability under loSRA and under loWRA.

Theorem 8.7 (loSRA and loWRA reachability). *Given a program P and a state $\overline{p} \in P.\mathsf{Q}$, it is decidable to check whether $\overline{p}$ is reachable (see Def. 4.4) under the memory systems* loSRA *and* loWRA.

Proof. Since the first component (the program state) in $\sqsubseteq$-ordered pairs of $P \parallel$ loXRA's states is equal, reachability under loXRA is reduced to coverability in $P \parallel$ loXRA w.r.t. $\sqsubseteq$ (a.k.a. *control-state reachability*), which is decidable by Lemma 8.5 and the results of [8]. □

Corollary 8.8. *The* SRA *and* WRA *reachability problems are decidable.*

Proof. Directly follows from Theorems 4.9 and 5.12 (for SRA) or Theorems 4.6 and 6.6 (for WRA), as well as Prop. 4.10 and Thm. 8.7. □

Corollary 8.9 (RA race-free reachability). *Given a program P such that every* SRA*-consistent execution graph that is generated by P is write/write-race free (see Def. 3.11), and a state $\overline{p} \in P.\mathsf{Q}$, it is decidable to check whether $\overline{p}$ is reachable under* RA.

Proof. Directly follows from Thm. 3.12 and Corollary 8.8. □

9 RELATED WORK

Decidability results. The reachability problem was previously investigated for TSO—the "total store ordering" model of x86 multiprocessors. TSO is a multi-copy-atomic model stronger than any of the models studied here (in particular it disallows the weak behavior of the IRIW program in Ex. 3.5). Atig et al. [12, 13] established the decidability of the problem (and a non-primitive recursive lower bound) by reducing it to (and from) reachability in lossy channel systems. Since causal consistency models are not multi-copy atomic and they lack any notion of a global mapping from locations to values, the idea behind their reduction to reachability in lossy channel system cannot be applied for the models studied here. Notably, unlike TSO and other (less realistic) models studied in [13], the models studied in the current paper *cannot* be fully explained by program transformations (instruction reordering and merging) [39]. On the other hand, the reduction of [12] *from* reachability in lossy channel systems to reachability under TSO, which establishes the non-primitive recursive lower bound, applies as is to the causal models.

More recently, Abdulla et al. [4] greatly simplified the previous proofs for TSO (and demonstrated much better practical running times on certain benchmarks) by developing and utilizing a "load-buffer" semantics for TSO. Load-buffers are roughly similar to our potential lists, but while load buffers are FIFO queues, our lists necessarily allow the insertion of future reads at different positions, subject to certain (novel) conditions ensuring that causal consistency is not violated. In addition, while the "load-buffer" semantics for TSO includes a global machine memory, our causal consistency semantics are, roughly speaking, based on point-to-point communication, allowing our "shared-memory causality principle" to govern the interactions between threads. Finally, our semantics

employs more than one option list per thread, while the "load-buffer" semantics for TSO has exactly one buffer of reads per thread.

Undecidability results. Abdulla et al. [2] proved the undecidability of safety verification under RA using a reduction from Post correspondence problem. More recently, in [5] the reachability problem was shown to be undecidable for the relaxed fragment of PS 2.0 (a version of the promising semantics) [41], and in [3] undecidability was established for the full POWER model as well.

Causal consistency and its related verification problems. Different causally consistent shared-memory models, their verification problems and approaches to address these problems were recently outlined in [34], where the problems we resolve here were left open. Operational "message-passing" semantics for SRA was developed in [36], but it is inadequate for our purposes since making it "lossy" would affect its allowed outcomes. Verification of programs under causal consistency (especially under RA) has received considerable amount of attention in recent years. The different approaches include (non-automated) program logics [23, 29, 38, 55, 56], (bounded) model checking [2, 6, 31, 42] and robustness verification [18, 37, 47]. The latter reduces the verification problem to the verification under sequential consistency and the verification of the program's robustness against causal consistency. Thus, this approach cannot work for programs that meet their safety specification but still exhibit non-sequentially-consistent behaviors. Finally, the problem asking whether a given implementation provides causal consistency guarantees was studied in [17]. It is, however, completely independent from verification of *client* programs assuming causal consistency, as we study here.

10 CONCLUSION AND FUTURE WORK

We have established the decidability of reachability under two main causal consistency models, SRA and WRA. To do so, we developed novel operational semantics for the two models that are based on the notion of thread potentials and meet the requirements for decidability of the framework of well-structured transition systems. Besides the theoretical interest, Abdulla et al. [4] demonstrate that similar verification procedures (also of non-primitive recursive complexity) may be actually practical for challenging (even though naturally quite small) algorithms and synchronization mechanisms. We plan to explore this in the future.

In contrast to our results, reachability is undecidable under RA, the C/C++11's causal consistency model [2]. Intuitively, this stems from the fact that RA requires one to maintain $\mathtt{mo}$ separately from the execution order, while SRA allows the execution of writes following $\mathtt{hb} \cup \mathtt{mo}$, and WRA does not use $\mathtt{mo}$ at all. More concretely, to support RA, the condition of loSRA that ensures that writes to each location x cannot execute when there are options to read x in the thread's potential has to be weakened (see Ex. 5.10). In turn, the conditions of loWRA are too weak, as they, in particular, do not ensure that all threads observe the writes to each location x in a way that is consistent across all threads (see Ex. 6.1). Finding alternative conditions on the write steps that will capture conditions closer to those of RA (either from above, like SRA, or from below, like WRA) is rather delicate and left to future work. In particular, we note that while the undecidability of RA implies that no similar WSTS can be developed for RA, the existing reduction that shows undecidability crucially relies on RMWs, and the decidability of RA without RMW operations is (to the best of our knowledge) still open.

We note that since SRA, RA and WRA coincide on write/write-race-free programs, and write/write-race freedom can be checked under SRA (Thm. 3.12), our result allows the verification of safety properties under RA for this class of programs. Concurrent separation logics [29, 55, 56], designed for verification under RA, are also essentially limited to reason only about write/write-race-free programs and stateless model checking is significantly simpler with this assumption (see [31,

§5 and Remark 3]). We also note that it is straightforward to support C/C++11's non-atomics, with "catch-fire" semantics (i.e., data races are errors) in addition to release/acquire accesses and sequentially consistent fences (which are modeled as RMWs as in Ex. 3.9). Indeed, as demonstrated in [29], it suffices to check for data races assuming RA semantics. Supporting other features of C/C++11, such as relaxed and sequentially consistent accesses, is left to future work.

We believe that the potential-based semantics—both specifically for SRA and WRA and as a general idea for operational semantics—may be of independent interest in the development of verification techniques for programs running under weak consistency, including program logics and model-checking techniques. In particular, we are interested in developing abstraction techniques, as was done for TSO and similar buffer-based models (see, e.g., [33, 53]). Other directions for future work include handling other variants of causally consistent shared-memory (see, e.g., [17]), supporting transactions (to enable, e.g., full verification of client programs under PSI, see §3.1) and studying verification of parametrized programs under causal consistency (which is decidable for TSO [4, 7]).

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful feedback, and Jean Pichon-Pharabod and Christopher Pulte for their comments on a previous version of this paper. This research was supported by the Israel Science Foundation (grant 1566/18) and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 851811). The first author was also supported by the Alon Young Faculty Fellowship.

REFERENCES

[1] Parosh Aziz Abdulla. 2010. Well (and better) quasi-ordered transition systems. *The Bulletin of Symbolic Logic* 16, 4 (2010), 457–515. http://www.jstor.org/stable/40961367

[2] Parosh Aziz Abdulla, Jatin Arora, Mohamed Faouzi Atig, and Shankaranarayanan Krishna. 2019. Verification of programs under the release-acquire semantics. In *PLDI*. ACM, New York, NY, USA, 1117–1132. https://doi.org/10.1145/3314221.3314649

[3] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmed Bouajjani, Egor Derevenetc, Carl Leonardsson, and Roland Meyer. 2021. On the state reachability problem for concurrent programs under Power. In *NETYS*. Springer International Publishing, Cham, 47–59.

[4] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmed Bouajjani, and Tuan Phong Ngo. 2018. A load-buffer semantics for total store ordering. *Logical Methods in Computer Science* Volume 14, Issue 1 (Jan. 2018). https://doi.org/10.23638/LMCS-14(1:9)2018

[5] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Adwait Godbole, S. Krishna, and Viktor Vafeiadis. 2021. The decidability of verification under PS 2.0. In *ESOP*. Springer International Publishing, Cham, 1–29.

[6] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bengt Jonsson, and Tuan Phong Ngo. 2018. Optimal stateless model checking under the release-acquire semantics. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 135 (Oct. 2018), 29 pages. https://doi.org/10.1145/3276505

[7] Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Rojin Rezvan. 2019. Parameterized verification under TSO is PSPACE-complete. *Proc. ACM Program. Lang.* 4, POPL, Article 26 (Dec. 2019), 29 pages. https://doi.org/10.1145/3371094

[8] Parosh Aziz Abdulla, Kārlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. 2000. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation* 160, 1 (2000), 109 – 127. https://doi.org/10.1006/inco.1999.2843

[9] Sarita V. Adve and Mark D. Hill. 1990. Weak ordering—a new definition. In *ISCA*. ACM, New York, NY, USA, 2–14. https://doi.org/10.1145/325164.325100

[10] Jade Alglave, Luc Maranget, and Michael Tautschnig. 2014. Herding cats: modelling, simulation, testing, and data mining for weak memory. *ACM Trans. Program. Lang. Syst.* 36, 2, Article 7 (July 2014), 74 pages. https://doi.org/10.1145/2627752

[11] Masoud Saeida Ardekani, Pierre Sutra, and Marc Shapiro. 2013. Non-monotonic snapshot isolation: Scalable and strong consistency for geo-replicated transactional systems. In *SRDS*. IEEE Computer Society, Washington, DC, USA, 163–172. https://doi.org/10.1109/SRDS.2013.25

[12] Mohamed Faouzi Atig, Ahmed Bouajjani, Sebastian Burckhardt, and Madanlal Musuvathi. 2010. On the verification problem for weak memory models. In *POPL*. ACM, New York, NY, USA, 7–18. https://doi.org/10.1145/1706299.1706303

[13] Mohamed Faouzi Atig, Ahmed Bouajjani, Sebastian Burckhardt, and Madanlal Musuvathi. 2012. What's decidable about weak memory models?. In *ESOP*. Springer-Verlag, Berlin, Heidelberg, 26–46. https://doi.org/10.1007/978-3-642-28869-2_2

[14] Mark Batty, Kayvan Memarian, Kyndylan Nienhuis, Jean Pichon-Pharabod, and Peter Sewell. 2015. The problem of programming language concurrency semantics. In *ESOP*. Springer, Berlin, Heidelberg, 283–307. https://doi.org/10.1007/978-3-662-46669-8_12

[15] Mark Batty, Scott Owens, Susmit Sarkar, Peter Sewell, and Tjark Weber. 2011. Mathematizing C++ concurrency. In *POPL*. ACM, New York, NY, USA, 55–66. https://doi.org/10.1145/1925844.1926394

[16] Giovanni Bernardi and Alexey Gotsman. 2016. Robustness against consistency models with atomic visibility. In *CONCUR*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 7:1–7:15. https://doi.org/10.4230/LIPIcs.CONCUR.2016.7

[17] Ahmed Bouajjani, Constantin Enea, Rachid Guerraoui, and Jad Hamza. 2017. On verifying causal consistency. In *POPL*. ACM, New York, NY, USA, 626–638. https://doi.org/10.1145/3009837.3009888

[18] Lucas Brutschy, Dimitar Dimitrov, Peter Müller, and Martin Vechev. 2018. Static serializability analysis for causal consistency. In *PLDI*. ACM, New York, NY, USA, 90–104. https://doi.org/10.1145/3192366.3192415

[19] Sebastian Burckhardt. 2014. Principles of eventual consistency. *Found. Trends Program. Lang.* 1, 1-2 (Oct. 2014), 1–150. https://doi.org/10.1561/2500000011

[20] Andrea Cerone, Alexey Gotsman, and Hongseok Yang. 2015. Transaction chopping for parallel snapshot isolation. In *DISC*. Springer-Verlag, Berlin, Heidelberg, 388–404. https://doi.org/10.1007/978-3-662-48653-5_26

[21] Andrea Cerone, Alexey Gotsman, and Hongseok Yang. 2017. Algebraic laws for weak consistency. In *CONCUR*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 26:1–26:18. https://doi.org/10.4230/LIPIcs.CONCUR.2017.26

[22] Leonard Eugene Dickson. 1913. Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors. *American Journal of Mathematics* 35, 4 (1913), 413–422. http://www.jstor.org/stable/2370405

[23] Simon Doherty, Brijesh Dongol, Heike Wehrheim, and John Derrick. 2019. Verifying C11 programs operationally. In *PPoPP*. ACM, New York, NY, USA, 355–365. https://doi.org/10.1145/3293883.3295702

[24] Stephen Dolan, KC Sivaramakrishnan, and Anil Madhavapeddy. 2018. Bounding data races in space and time. In *PLDI*. ACM, New York, NY, USA, 242–255. https://doi.org/10.1145/3192366.3192421

[25] Alain Finkel and Philippe Schnoebelen. 2001. Well-structured transition systems everywhere! *Theoretical Computer Science* 256, 1 (2001), 63 – 92. https://doi.org/10.1016/S0304-3975(00)00102-X

[26] Graham Higman. 1952. Ordering by Divisibility in Abstract Algebras. *Proceedings of the London Mathematical Society* s3-2, 1 (1952), 326–336. https://doi.org/10.1112/plms/s3-2.1.326

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

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

[29] Jan-Oliver Kaiser, Hoang-Hai Dang, Derek Dreyer, Ori Lahav, and Viktor Vafeiadis. 2017. Strong logic for weak memory: Reasoning about release-acquire consistency in Iris. In *ECOOP*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 17:1–17:29. https://doi.org/10.4230/LIPIcs.ECOOP.2017.17

[30] Jeehoon Kang, Chung-Kil Hur, Ori Lahav, Viktor Vafeiadis, and Derek Dreyer. 2017. A Promising Semantics for Relaxed-Memory Concurrency. In *POPL*. ACM, New York, NY, USA, 175–189. https://doi.org/10.1145/3009837.3009850

[31] Michalis Kokologiannakis, Ori Lahav, Konstantinos Sagonas, and Viktor Vafeiadis. 2017. Effective stateless model checking for C/C++ concurrency. *Proc. ACM Program. Lang.* 2, POPL, Article 17 (Dec. 2017), 32 pages. https://doi.org/10.1145/3158105

[32] Dexter Kozen. 1977. Lower bounds for natural proof systems. In *SFCS*. IEEE Computer Society, Washington, 254–266. https://doi.org/10.1109/SFCS.1977.16

[33] Michael Kuperstein, Martin Vechev, and Eran Yahav. 2011. Partial-coherence abstractions for relaxed memory models. In *PLDI*. ACM, New York, NY, USA, 187–198. https://doi.org/10.1145/1993498.1993521

[34] Ori Lahav. 2019. Verification under causally consistent shared memory. *ACM SIGLOG News* 6, 2 (April 2019), 43–56. https://doi.org/10.1145/3326938.3326942

[35] Ori Lahav and Udi Boker. 2020. Decidable verification under a causally consistent shared memory. In *PLDI*. ACM, New York, NY, USA, 211–226. https://doi.org/10.1145/3385412.3385966

[36] Ori Lahav, Nick Giannarakis, and Viktor Vafeiadis. 2016. Taming release-acquire consistency. In *POPL*. ACM, New York, NY, USA, 649–662. https://doi.org/10.1145/2837614.2837643

[37] Ori Lahav and Roy Margalit. 2019. Robustness against release/acquire semantics. In *PLDI*. ACM, New York, NY, USA, 126–141. https://doi.org/10.1145/3314221.3314604

[38] Ori Lahav and Viktor Vafeiadis. 2015. Owicki-Gries reasoning for weak memory models. In *ICALP*. Springer-Verlag, Berlin, Heidelberg, 311–323. https://doi.org/10.1007/978-3-662-47666-6_25

[39] Ori Lahav and Viktor Vafeiadis. 2016. Explaining relaxed memory models with program transformations. In *FM*. Springer, Cham, 479–495. https://doi.org/10.1007/978-3-319-48989-6_29
[40] Ori Lahav, Viktor Vafeiadis, Jeehoon Kang, Chung-Kil Hur, and Derek Dreyer. 2017. Repairing sequential consistency in C/C++11. In *PLDI*. ACM, New York, NY, USA, 618–632. https://doi.org/10.1145/3062341.3062352
[41] Sung-Hwan Lee, Minki Cho, Anton Podkopaev, Soham Chakraborty, Chung-Kil Hur, Ori Lahav, and Viktor Vafeiadis. 2020. Promising 2.0: global optimizations in relaxed memory concurrency. In *PLDI*. ACM, New York, NY, USA, 362–376. https://doi.org/10.1145/3385412.3386010
[42] Mohsen Lesani, Christian J. Bell, and Adam Chlipala. 2016. Chapar: Certified causally consistent distributed key-value stores. In *POPL*. ACM, New York, NY, USA, 357–370. https://doi.org/10.1145/2837614.2837622
[43] Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues. 2012. Making geo-replicated systems fast as possible, consistent when necessary. In *OSDI*. USENIX Association, Berkeley, CA, USA, 265–278. http://dl.acm.org/citation.cfm?id=2387880.2387906
[44] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. 2011. Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS. In *SOSP*. ACM, New York, NY, USA, 401–416. https://doi.org/10.1145/2043556.2043593
[45] Mapping 2019. C/C++11 mappings to processors. Retrieved July 3, 2019 from http://www.cl.cam.ac.uk/~pes20/cpp/cpp0xmappings.html
[46] Luc Maranget, Susmit Sarkar, and Peter Sewell. 2012. A tutorial introduction to the ARM and POWER relaxed memory models. http://www.cl.cam.ac.uk/~pes20/ppc-supplemental/test7.pdf.
[47] Kartik Nagar and Suresh Jagannathan. 2018. Automated detection of serializability violations under weak consistency. In *CONCUR*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 41:1–41:18. https://doi.org/10.4230/LIPIcs.CONCUR.2018.41
[48] Scott Owens, Susmit Sarkar, and Peter Sewell. 2009. A better x86 memory model: x86-TSO. In *TPHOLs*. Springer, Heidelberg, 391–407. https://doi.org/10.1007/978-3-642-03359-9_27
[49] Azalea Raad, Ori Lahav, and Viktor Vafeiadis. 2018. On parallel snapshot isolation and release/acquire consistency. In *ESOP*. Springer, Berlin, Heidelberg, 940–967. https://doi.org/10.1007/978-3-319-89884-1_33
[50] Susmit Sarkar, Peter Sewell, Jade Alglave, Luc Maranget, and Derek Williams. 2011. Understanding POWER multiprocessors. In *PLDI*. ACM, New York, NY, USA, 175–186. https://doi.org/10.1145/1993498.1993520
[51] Sylvain Schmitz and Philippe Schnoebelen. 2012. Algorithmic aspects of WQO theory. (Aug. 2012). https://cel.archives-ouvertes.fr/cel-00727025 Lecture notes.
[52] Yair Sovran, Russell Power, Marcos K. Aguilera, and Jinyang Li. 2011. Transactional storage for geo-replicated systems. In *SOSP*. ACM, New York, NY, USA, 385–400. https://doi.org/10.1145/2043556.2043592
[53] Thibault Suzanne and Antoine Miné. 2016. From array domains to abstract interpretation under store-buffer-based memory models. In *SAS*. Springer, Berlin, Heidelberg, 469–488.
[54] Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike Spreitzer, Marvin Theimer, and Brent W. Welch. 1994. Session guarantees for weakly consistent replicated data. In *PDIS*. IEEE Computer Society, Washington, DC, USA, 140–149. http://dl.acm.org/citation.cfm?id=645792.668302
[55] Aaron Turon, Viktor Vafeiadis, and Derek Dreyer. 2014. GPS: Navigating weak memory with ghosts, protocols, and separation. In *OOPSLA*. ACM, New York, NY, USA, 691–707. https://doi.org/10.1145/2660193.2660243
[56] Viktor Vafeiadis and Chinmay Narayan. 2013. Relaxed separation logic: A program logic for C11 concurrency. In *OOPSLA*. ACM, New York, NY, USA, 867–884. https://doi.org/10.1145/2509136.2509532
[57] Paolo Viotti and Marko Vukolić. 2016. Consistency in non-transactional distributed storage systems. *ACM Comput. Surv.* 49, 1, Article 19 (June 2016), 34 pages. https://doi.org/10.1145/2926965
[58] John Wickerson, Mark Batty, Tyler Sorensen, and George A. Constantinides. 2017. Automatically comparing memory consistency models. In *POPL*. ACM, New York, NY, USA, 190–204. https://doi.org/10.1145/3009837.3009838

A FULL EQUIVALENCE PROOFS

In this appendix, we provide full proofs of Lemmas 7.6 and 7.7, first for SRA and then for WRA. Our proofs assume the alternative definition of loXRA's write steps that is given in Prop. 8.4.

A.1 Equivalence of loSRA and opSRA

Lemma A.1. *For every trace of* loSRA *there is an equivalent trace of* opSRA.

Proof. As described in §7, we show that $\curlyvee$ constitutes a forward simulation relation from loSRA to opSRA. We detail here the simulation step. Suppose that $\mathcal{B} \curlyvee G$ and $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loSRA}} \mathcal{B}'$ for some

$\tau \in \mathsf{Tid}$ and $l \in \mathsf{Lab}$. We show that there exists G' such that $\mathcal{B}' \triangleright G'$ and $G \xrightarrow{\tau,l}_{\mathsf{opSRA}} G'$. Consider the possible cases:

- WRITE step, $l = \mathtt{W}(x, v_{\mathtt{W}})$:
  Let $w = \mathsf{NextEvent}(G.\mathsf{E}, \tau, l)$. Let G' be the execution graph defined by $G'.\mathsf{E} = G.\mathsf{E} \cup \{w\}$, $G'.\mathsf{rf} = G.\mathsf{rf}$ and $G'.\mathsf{mo} = G.\mathsf{mo} \cup (G.\mathtt{W}_x \times \{w\})$. By definition, we have $G \xrightarrow{\tau,l}_{\mathsf{opSRA}} G'$. We show that $\mathcal{B}' \triangleright G'$. By Prop. 8.4, since $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loSRA}} \mathcal{B}'$, there exists an index choice $\mathcal{P}$ for $\mathcal{B}'$ such that $\mathcal{P} \models_{\mathsf{SRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$. and $\mathrm{src}(\mathcal{B}', \tau, \mathcal{P})(\pi) \subseteq \mathcal{B}(\pi)$ for every $\pi \in \mathsf{Tid}$. Let $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$. We construct a $\langle G', \pi\rangle$-consistent $\langle G', L'\rangle$-write-list W'. Let $P \triangleq \mathcal{P}(\pi, L')$, $L \triangleq L' \setminus P$, $f \triangleq \mathsf{Map}_{\langle L', P\rangle}$, $L_\tau \triangleq L' \setminus\!\setminus P$ and $f_\tau \triangleq \mathsf{MMap}_{\langle L', P\rangle}$ (the last two are only defined if $P \neq \emptyset$).
  Since $\mathcal{B} \triangleright G$, there exist a $\langle G, \pi\rangle$-consistent $\langle G, L\rangle$-write-list W, and a $\langle G, \tau\rangle$-consistent $\langle G, L_\tau\rangle$-write-list W_τ. We define W' as follows:

$$W' \triangleq \lambda k \in \{1, \ldots, |L'|\}. \begin{cases} w & k \in P \\ W(f(k)) & k < \min(P) \\ \max_{G.\mathsf{mo}}\{W(f(k)), W_\tau(f_\tau(k))\} & \text{otherwise} \end{cases}$$

  It is easy to see that W' is a $\langle G', L'\rangle$-write-list. In particular, to show that $\mathtt{rmw}(L'(k)) = \mathtt{RMW}$ implies $W'(k) \notin \mathit{dom}(G'.\mathsf{mo})$, we use the fact that $\mathcal{P} \models_{\mathsf{SRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$, and so for every $k \in \{1, \ldots, |L'|\} \setminus P$, we have that $\mathtt{rmw}(L'(k)) = \mathtt{RMW}$ implies $\mathtt{loc}(L'(k)) \neq x$.
  We show that W' is $\langle G', \pi\rangle$-consistent.
  Let $1 \leq k \leq |L'|$. We prove that $W'(k) \notin \mathit{dom}(G'.\mathsf{mo} ; G'.\mathsf{hb}^? ; [\mathsf{E}^\pi \cup \{W'(j) \mid 1 \leq j < k\}])$. Suppose otherwise. First, note that we cannot have $k \in P$, since w is a maximal element in $G'.\mathsf{mo}$. Let $w_\pi = W(f(k))$ and $w_\tau = W_\tau(f_\tau(k))$ (the latter is only defined if $k > \min(P)$). Consider the two possible cases:
  - $W'(k) \in \mathit{dom}(G'.\mathsf{mo} ; G'.\mathsf{hb}^? ; [\mathsf{E}^\pi])$: The definition of W' ensures that $\langle w_\pi, W'(k)\rangle \in G'.\mathsf{mo}^?$, and so $w_\pi \in \mathit{dom}(G'.\mathsf{mo} ; G'.\mathsf{hb}^? ; [\mathsf{E}^\pi])$. From the $\langle G, \pi\rangle$-consistency of W, we know that $w_\pi \notin \mathit{dom}(G.\mathsf{mo} ; G.\mathsf{hb}^? ; [\mathsf{E}^\pi])$, and therefore it must be the case that $\pi = \tau$ and $\langle w_\pi, w\rangle \in G'.\mathsf{mo}$. Hence, $\mathtt{loc}(w_\pi) = x$, and so $\mathtt{loc}(L'(k)) = x$, which contradicts the fact that $\mathcal{P} \models_{\mathsf{SRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$.
  - $\langle W'(k), W'(j)\rangle \in G'.\mathsf{mo} ; G'.\mathsf{hb}^?$ for some $1 \leq j < k$. Consider the two possible cases:
    * $W'(j) = w$: In this case we must have $k > \min(P)$, and so $W'(k) = \max_{G.\mathsf{mo}}\{w_\pi, w_\tau\}$. Hence, we have $\langle w_\tau, W'(k)\rangle \in G.\mathsf{mo}^?$, and so $\langle w_\tau, w\rangle \in G'.\mathsf{mo};G'.\mathsf{hb}^?$. Now, if $\langle w_\tau, w\rangle \in G'.\mathsf{mo};G'.\mathsf{hb}$, then we also have $w_\tau \in \mathit{dom}(G.\mathsf{mo} ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$, which contradicts the fact that W_τ is $\langle G, \tau\rangle$-consistent. Therefore, we have $\langle w_\tau, w\rangle \in G'.\mathsf{mo}$. Hence, $\mathtt{loc}(w_\tau) = x$, and so $\mathtt{loc}(L'(k)) = x$, which contradicts the fact that $\mathcal{P} \models_{\mathsf{SRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$.
    * $W'(j) \neq w$: In this case, we must have $\langle W'(k), W'(j)\rangle \in G.\mathsf{mo} ; G.\mathsf{hb}^?$. The definition of W' ensures that $\langle w_\pi, W'(k)\rangle \in G.\mathsf{mo}^?$, and so $\langle w_\pi, W'(j)\rangle \in G.\mathsf{mo};G.\mathsf{hb}^?$. Now, since W is $\langle G, \pi\rangle$-consistent, we cannot have $W'(j) = W(f(j))$. Hence, $j > \min(P)$ and $W'(j) = W_\tau(f_\tau(j))$. Let $w'_\tau = W_\tau(f_\tau(j))$. It follows that $k > \min(P)$, and so $\langle w_\tau, W'(k)\rangle \in G.\mathsf{mo}^?$. Hence, we have $\langle w_\tau, w'_\tau\rangle \in G.\mathsf{mo} ; G.\mathsf{hb}^?$. This contradicts the fact that W_τ is $\langle G, \tau\rangle$-consistent.
- READ step, $l = \mathtt{R}(x, v_{\mathtt{R}})$:
  By definition, since $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loSRA}} \mathcal{B}'$, there exists a read option o with $\mathtt{loc}(o) = x$ and $\mathtt{val}(o) = v_{\mathtt{R}}$ such that $\mathcal{B}(\tau) = o \cdot \mathcal{B}'(\tau)$. Since $\mathcal{B} \triangleright G$, for every $L \in \mathcal{B}(\tau)$ there exists a $\langle G, \tau\rangle$-consistent $\langle G, L\rangle$-write-list W_L. Let $A = \{W_L(1) \mid L \in \mathcal{B}(\tau)\}$. Since $\mathcal{B}(\tau)$ is non-empty, we know that A is not empty. Since each W_L is a $\langle G, L\rangle$-write-list, we have that $\mathtt{tid}(w) = \mathtt{tid}(o)$ for every $w \in A$. Hence, $G.\mathsf{po}$ totally orders A. Let $w = \min_{G.\mathsf{po}} A$ and let $L_{\min} \in \mathcal{B}(\tau)$ such that $w = W_{L_{\min}}(1)$.

Let $r = \mathsf{NextEvent}(G.\mathsf{E}, \tau, l)$ and let G' be the execution graph given by $G'.\mathsf{E} = G.\mathsf{E} \cup \{r\}$, $G'.\mathsf{rf} = G.\mathsf{rf} \cup \{\langle w, r\rangle\}$ and $G'.\mathsf{mo} = G.\mathsf{mo}$.

We show that $G \xrightarrow{\tau,l}_{\mathsf{opSRA}} G'$. By definition, it suffices to show the following:

- $w \in G.\mathbb{W}_x$ and $\mathtt{val}_{\mathbb{W}}(w) = v_{\mathsf{R}}$: We have $w = W_{L_{\min}}(1)$, and since $W_{L_{\min}}$ is a $\langle G, L_{\min}\rangle$-write-list, we have that $w \in G.\mathbb{W}$, $\mathtt{loc}(w) = \mathtt{loc}(W_{L_{\min}}(1)) = \mathtt{loc}(L_{\min}(1)) = \mathtt{loc}(o) = x$ and $\mathtt{val}_{\mathbb{W}}(w) = \mathtt{val}_{\mathbb{W}}(W_{L_{\min}}(1)) = \mathtt{val}(L_{\min}(1)) = \mathtt{val}(o) = v_{\mathsf{R}}$.
- $w \notin dom(G.\mathsf{mo}\,;G.\mathsf{hb}^?\,;[\mathsf{E}^\tau])$: Since $W_{L_{\min}}$ is $\langle G, \tau\rangle$-consistent and $w = W_{L_{\min}}(1)$, we cannot have $w \in dom(G.\mathsf{mo}\,;G.\mathsf{hb}^?\,;[\mathsf{E}^\tau])$.

It remains to show that $\mathcal{B}' \triangleright G'$. Let $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$. We define a $\langle G', \pi\rangle$-consistent $\langle G', L'\rangle$-write-list. Consider two cases:

- $\pi \neq \tau$: By definition, since $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loSRA}} \mathcal{B}'$, we have $L' \in \mathcal{B}(\pi)$. Since $\mathcal{B} \triangleright G$, there exists a $\langle G, \pi\rangle$-consistent $\langle G, L'\rangle$-write-list W. It is easy to see that W is a $\langle G', L'\rangle$-write-list. We show that W is also $\langle G', \pi\rangle$-consistent. Let $1 \leq k \leq |L'|$.
  Suppose by contradiction that $W(k) \in dom(G'.\mathsf{mo}\,;G'.\mathsf{hb}^?\,;[\mathsf{E}^\pi \cup \{W(j) \mid 1 \leq j < k\}])$. It follows that $W(k) \in dom(G.\mathsf{mo}\,;G.\mathsf{hb}^?\,;[\mathsf{E}^\pi \cup \{W(j) \mid 1 \leq j < k\}])$. This contradicts the fact that W is $\langle G, \pi\rangle$-consistent.
- $\pi = \tau$: Let $L = o \cdot L'$. Then, $L \in \mathcal{B}(\tau)$. Let $W' = \lambda k \in \{1, \ldots, |L'|\}.\ W_L(1+k)$. It is easy to see that W' is a $\langle G', L'\rangle$-write-list. We show that W' is $\langle G', \tau\rangle$-consistent. Suppose by contradiction that $W'(k) \in dom(G'.\mathsf{mo}\,;G'.\mathsf{hb}^?\,;[\mathsf{E}^\tau \cup \{W'(j) \mid 1 \leq j < k\}])$.
  Now, if $W'(k) \in dom(G.\mathsf{mo}\,;G.\mathsf{hb}^?\,;[\mathsf{E}^\tau \cup \{W'(j) \mid 1 \leq j < k\}])$, it follows that

$$W_L(1+k) \in dom(G.\mathsf{mo}\,;G.\mathsf{hb}^?\,;[\mathsf{E}^\tau \cup \{W_L(1+j) \mid 1 \leq j < k\}]),$$

  which contradicts the fact that W_L is $\langle G, \tau\rangle$-consistent. Hence, we must have $\langle W'(k), w\rangle \in G.\mathsf{mo};G.\mathsf{hb}^?$. Since $L(1) = o$, the definition of w ensures that $\langle w, W_L(1)\rangle \in G.\mathsf{po}^?$. It follows that $\langle W_L(1+k), W_L(1)\rangle \in G.\mathsf{mo};G.\mathsf{hb}^?$, which again contradicts the fact that W_L is $\langle G, \tau\rangle$-consistent.

- RMW step, $l = \mathsf{RMW}(x, v_{\mathsf{R}}, v_{\mathsf{W}})$:
  This case is handled by carefully combining the WRITE and READ steps. By definition, since $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loSRA}} \mathcal{B}'$, there exists a read option o with $\mathtt{loc}(o) = x$, $\mathtt{val}(o) = v_{\mathsf{R}}$ and $\mathtt{rmw}(o) = \mathsf{RMW}$ such that $L(1) = o$ for every $L \in \mathcal{B}(\tau)$. Since $\mathcal{B} \triangleright G$, for every $L \in \mathcal{B}(\tau)$ there exists a $\langle G, \tau\rangle$-consistent $\langle G, L\rangle$-write-list W_L. Moreover, since $\mathtt{rmw}(o) = \mathsf{RMW}$, we have $W_L(1) = \max_{G.\mathsf{mo}} G.\mathbb{W}_x$ for every $L \in \mathcal{B}(\tau)$.
  Let $w = \max_{G.\mathsf{mo}} G.\mathbb{W}_x$, $e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, l)$ and G' be the execution graph given by $G'.\mathsf{E} = G.\mathsf{E} \cup \{e\}$, $G'.\mathsf{rf} = G.\mathsf{rf} \cup \{\langle w, e\rangle\}$ and $G'.\mathsf{mo} = G.\mathsf{mo} \cup (G.\mathbb{W}_x \times \{e\})$.
  For showing that $G \xrightarrow{\tau,l}_{\mathsf{opSRA}} G'$, it suffices, by definition, to show that $\mathtt{val}_{\mathbb{W}}(w) = v_{\mathsf{R}}$. Indeed, since $\mathcal{B}(\tau)$ is (by definition) non-empty, we can take some $L \in \mathcal{B}(\tau)$. We have $w = W_L(1)$, and since W_L is a $\langle G, L\rangle$-write-list, we have that $\mathtt{val}_{\mathbb{W}}(w) = \mathtt{val}_{\mathbb{W}}(W_L(1)) = \mathtt{val}(L(1)) = \mathtt{val}(o) = v_{\mathsf{R}}$.
  It remains to show that $\mathcal{B}' \triangleright G'$. Using Prop. 8.4, since $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loSRA}} \mathcal{B}'$, we know that there exists an index choice $\mathcal{P}$ for $\mathcal{B}'$ such that $\mathcal{P} \models_{\mathsf{SRA}} \langle \tau, \mathbb{W}(x, v_{\mathsf{W}})\rangle$, $\mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\pi) \subseteq \mathcal{B}(\pi)$ for every $\pi \in \mathsf{Tid} \setminus \{\tau\}$ and $o \cdot \mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\tau) \subseteq \mathcal{B}(\tau)$.
  Let $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$. We construct a $\langle G', \pi\rangle$-consistent $\langle G', L'\rangle$-write-list W'. Let $P \triangleq \mathcal{P}(\pi, L')$ and (L_τ and f_τ and are only defined if $P \neq \emptyset$):

$$L \triangleq \begin{cases} L' \setminus P & \pi \neq \tau \\ o \cdot (L' \setminus P) & \pi = \tau \end{cases} \qquad f \triangleq \begin{cases} \mathsf{Map}_{\langle L', P\rangle} & \pi \neq \tau \\ \lambda k \in \{1, \ldots, |L'|\} \setminus P.\ \mathsf{Map}_{\langle L', P\rangle}(k) + 1 & \pi = \tau \end{cases}$$

$$L_\tau \triangleq o \cdot L' \setminus\setminus P \qquad f_\tau \triangleq \lambda k \in \{\min(P), \ldots, |L'|\} \setminus P.\ \mathsf{MMap}_{\langle L', P\rangle}(k) + 1$$

Since $\mathcal{B} \triangleright G$, there exist a $\langle G, \pi\rangle$-consistent $\langle G, L\rangle$-write-list W, and a $\langle G, \tau\rangle$-consistent $\langle G, L_\tau\rangle$-write-list W_τ. We define W' as follows:

$$W' \triangleq \lambda k \in \{1, \ldots, |L'|\}. \begin{cases} e & k \in P \\ W(f(k)) & k < \min(P) \\ \max_{G.\mathtt{mo}}\{W(f(k)), W_\tau(f_\tau(k))\} & \text{otherwise} \end{cases}$$

It is easy to see that W' is a $\langle G', L'\rangle$-write-list. In particular, to show that $\mathtt{rmw}(L'(k)) = \mathtt{RMW}$ implies $W'(k) \notin dom(G'.\mathtt{mo})$, we use the fact that $\mathcal{P} \models_{\mathsf{SRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$, and so for every $k \in \{1, \ldots, |L'|\} \setminus P$, we have that $\mathtt{rmw}(L'(k)) = \mathtt{RMW}$ implies $\mathtt{loc}(L'(k)) \neq x$.
We show that W' is $\langle G', \pi\rangle$-consistent.
Let $1 \leq k \leq |L'|$. We prove that $W'(k) \notin dom(G'.\mathtt{mo}\,;G'.\mathtt{hb}^?\,;[\mathtt{E}^\pi \cup \{W'(j) \mid 1 \leq j < k\}])$. Suppose otherwise. First, note that we cannot have $k \in P$, since e is a maximal element in $G'.\mathtt{mo}$. Let $w_\pi = W(f(k))$ and $w_\tau = W_\tau(f_\tau(k))$ (the latter is only defined if $k > \min(P)$). Consider the two possible cases:

- $W'(k) \in dom(G'.\mathtt{mo}\,;G'.\mathtt{hb}^?\,;[\mathtt{E}^\pi])$: The definition of W' ensures that $\langle w_\pi, W'(k)\rangle \in G'.\mathtt{mo}^?$, and so $w_\pi \in dom(G'.\mathtt{mo}\,;G'.\mathtt{hb}^?\,;[\mathtt{E}^\pi])$. From the $\langle G, \pi\rangle$-consistency of W, we know that $w_\pi \notin dom(G.\mathtt{mo}\,;G.\mathtt{hb}^?\,;[\mathtt{E}^\pi])$, and therefore it must be the case that $\langle w_\pi, e\rangle \in G'.\mathtt{mo};(G.\mathtt{hb};G'.\mathtt{rf})^?$ and $\pi = \tau$. Since $\mathcal{P} \models_{\mathsf{SRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$, we have $\mathtt{loc}(L'(k)) \neq x$, and so $\mathtt{loc}(w_\pi) \neq x$. Hence, $\langle w_\pi, e\rangle \notin G'.\mathtt{mo}$, and so we have $\langle w_\pi, e\rangle \in G.\mathtt{mo}\,;G.\mathtt{hb}\,;G'.\mathtt{rf}$, namely $\langle w_\pi, w\rangle \in G.\mathtt{mo}\,;G.\mathtt{hb}$. However, $W(1) = w$, contradicting the $\langle G, \pi\rangle$-consistency of W.
- $\langle W'(k), W'(j)\rangle \in G'.\mathtt{mo}\,;G'.\mathtt{hb}^?$ for some $1 \leq j < k$. Consider the two possible cases:
  - $W'(j) = e$: In this case we must have $k > \min(P)$, and so $W'(k) = \max_{G.\mathtt{mo}}\{w_\pi, w_\tau\}$. There are three possibilities:
    - $W'(k) = w$: Then $\mathtt{loc}(w_\tau) = \mathtt{loc}(L'(k)) = x$, which contradicts the fact that $\mathcal{P} \models_{\mathsf{SRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$.
    - $\langle W'(k), w\rangle \in G'.\mathtt{mo}\,;G'.\mathtt{hb}^?$: This contradicts the $\langle G, \tau\rangle$-consistency of W_τ, as $W_\tau(1) = w$ and $\langle w_\tau, W'(k)\rangle \in G'.\mathtt{mo}^?$, implying that $\langle w_\tau, W_\tau(1)\rangle \in G.\mathtt{mo}\,;G.\mathtt{hb}^?$.
    - $\langle W'(k), e\rangle \in G'.\mathtt{mo}\,;G'.\mathtt{hb}^?\,;G'.\mathtt{po}$: This also contradicts the $\langle G, \tau\rangle$-consistency of W_τ, as we get that $w_\tau \in dom(G.\mathtt{mo}\,;G.\mathtt{hb}^?\,;[\mathtt{E}^\tau])$.
  - $W'(j) \neq e$: In this case, we must have $\langle W'(k), W'(j)\rangle \in G.\mathtt{mo}\,;G.\mathtt{hb}^?$. The definition of W' ensures that $\langle w_\pi, W'(k)\rangle \in G.\mathtt{mo}^?$, and so $\langle w_\pi, W'(j)\rangle \in G.\mathtt{mo};G.\mathtt{hb}^?$. Now, since W is $\langle G, \pi\rangle$-consistent, we cannot have $W'(j) = W(f(j))$. Let $w'_\tau = W_\tau(f_\tau(j))$. Hence, $j > \min(P)$ and $W'(j) = w'_\tau$. It follows that $k > \min(P)$, and so $\langle w_\tau, W'(k)\rangle \in G.\mathtt{mo}^?$. Hence, we have $\langle w_\tau, w'_\tau\rangle \in G.\mathtt{mo}\,;G.\mathtt{hb}^?$. This contradicts the fact that W_τ is $\langle G, \tau\rangle$-consistent.

Finally, for handling the LOWER step, suppose that $\mathcal{B} \triangleright G$ and $\mathcal{B} \xrightarrow{\varepsilon}_{\mathsf{loSRA}} \mathcal{B}'$. We show that $\mathcal{B}' \triangleright G$. Let $\tau \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\tau)$. We define a $\langle G, \tau\rangle$-consistent $\langle G, L'\rangle$-write-list W'. By definition, since $\mathcal{B} \xrightarrow{\varepsilon}_{\mathsf{loSRA}} \mathcal{B}'$, there exists $L \in \mathcal{B}(\tau)$ such that $L' \sqsubseteq L$. Let $f : \{1, \ldots, |L'|\} \to \mathbb{N}$ be an increasing function such that $L'(k) = L(f(k))$ for every $k \in dom(f)$. Since $\mathcal{B} \triangleright G$, there exists a $\langle G, \tau\rangle$-consistent $\langle G, L\rangle$-write-list W. Let $W' = \lambda k \in \{1, \ldots, |L'|\}.\ W(f(k))$. It is easy to see that W' is a $\langle G, L'\rangle$-write-list. We show that W' is $\langle G, \tau\rangle$-consistent. Let $1 \leq k \leq |L'|$. Suppose by contradiction that $W'(k) \in dom(G.\mathtt{mo}\,;G.\mathtt{hb}^?\,;[\mathtt{E}^\tau \cup \{W'(j) \mid 1 \leq j < k\}])$. It follows that $W(f(k)) \in dom(G.\mathtt{mo}\,;G.\mathtt{hb}^?\,;[\mathtt{E}^\tau \cup \{W(f(j)) \mid 1 \leq j < k\}])$. This contradicts the fact that W is $\langle G, \tau\rangle$-consistent. □

LEMMA A.2. *For every trace of* opSRA *there is an equivalent trace of* loSRA.

Proof. As described in §7, we show that $\curlyvee^{-1}$ constitutes a backward simulation from opSRA to loSRA. We detail here the simulation step. Suppose that $G \xrightarrow{\tau,l}_{\mathsf{opSRA}} G'$ and $\mathcal{B}' \curlyvee G'$. We construct a state $\mathcal{B}$ such that $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loSRA}} \mathcal{B}'$ and $\mathcal{B} \curlyvee G$ (depicted on the right). Consider the possible cases:

- write step, $l = \mathtt{W}(x, v_{\mathtt{W}})$:

  Let $w = \mathsf{NextEvent}(G.\mathsf{E}, \tau, l)$. Since $G \xrightarrow{\tau,l}_{\mathsf{opSRA}} G'$, we have $G'.\mathsf{E} = G.\mathsf{E} \cup \{w\}$, $G'.\mathtt{rf} = G.\mathtt{rf}$ and $G'.\mathtt{mo} = G.\mathtt{mo} \cup (G.\mathsf{W}_x \times \{w\})$. Since $\mathcal{B}' \curlyvee G'$, for every $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$ there exists a $\langle G', \pi\rangle$-consistent $\langle G', L'\rangle$-write-list $W'_{\langle \pi, L'\rangle}$. Let $\mathcal{P}$ be the index choice for $\mathcal{B}'$ that assigns the set of "new" positions in $\mathcal{B}'$:

  $$\mathcal{P} \triangleq \lambda \pi \in \mathsf{Tid}, L' \in \mathcal{B}'(\pi).\ \{1 \le k \le |L'| \mid W'_{\langle \pi, L'\rangle}(k) = w\}.$$

  Then, we define $\mathcal{B} \triangleq \mathsf{src}(\mathcal{B}', \tau, \mathcal{P})$.

  By Prop. 8.4, to show that $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loSRA}} \mathcal{B}'$, it suffices to prove that $\mathcal{P} \models_{\mathsf{SRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$. Thus, we show that the following hold for every $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$, where $P = \mathcal{P}(\pi, L')$ and $W' = W'_{\langle \pi, L'\rangle}$:
  - Let $k \in P$. To see that $L'(k) \in \{\mathtt{O}_{\mathtt{R}}(\tau, x, v_{\mathtt{W}}, \mathtt{R}), \mathtt{O}_{\mathtt{R}}(\tau, x, v_{\mathtt{W}}, \mathtt{RMW})\}$, note that since $k \in P$, we have $W'(k) = w$, and since W' is a $\langle G', L'\rangle$-write-list, we must have $\tau = \mathtt{tid}(w) = \mathtt{tid}(L'(k))$, $x = \mathtt{loc}(w) = \mathtt{loc}(L'(k)) = x$ and $v_{\mathtt{W}} = \mathtt{val}_{\mathtt{W}}(w) = \mathtt{val}_{\mathtt{W}}(L'(k))$.
  - Let $k \in \{m+1, \ldots, |L'|\} \setminus P$ where $m = \min(P)$. We show that $\mathtt{loc}(L'(k)) \neq x$. Suppose otherwise. Let $w' = W'(k)$. Since $k \notin P$, we have $w' \neq w$. Hence, since $G'.\mathtt{mo} = G.\mathtt{mo} \cup (G.\mathsf{W}_x \times \{w\})$, we have $\langle w', w\rangle \in G'.\mathtt{mo}$. Thus, $\langle w', W'(m)\rangle \in G'.\mathtt{mo}\,;\, G'.\mathtt{hb}^?$. Since $k > m$, this contradicts the fact that W' is $\langle G', \pi\rangle$-consistent.
  - Suppose that $\pi = \tau$ and let $k \in \{1, \ldots, |L'|\} \setminus P$. We show that $\mathtt{loc}(L'(k)) \neq x$. Suppose otherwise. Let $w' = W'(k)$. Since $k \notin P$, we have $w' \neq w$. Hence, since $G'.\mathtt{mo} = G.\mathtt{mo} \cup (G.\mathsf{W}_x \times \{w\})$, we have $\langle w', w\rangle \in G'.\mathtt{mo}$. Thus, we have $w' \in \mathit{dom}(G'.\mathtt{mo}\,;\, G'.\mathtt{hb}^?\,;\, [\mathsf{E}^\tau])$, which contradicts the fact that W' is $\langle G', \tau\rangle$-consistent.
  - Let $k \in \{1, \ldots, |L'|\} \setminus P$, such that $\mathtt{loc}(L'(k)) = x$. We show that $\mathtt{rmw}(L'(k)) = \mathtt{R}$. Let $w' = W'(k)$. Since $k \notin P$, we have $w' \neq w$. Since $G'.\mathtt{mo} = G.\mathtt{mo} \cup (G.\mathsf{W}_x \times \{w\})$, it follows that $\langle w', w\rangle \in G'.\mathtt{mo}$. However, since W' is a $\langle G', L'\rangle$-write-list, if $\mathtt{rmw}(L'(k)) = \mathtt{RMW}$, then we must have $w' = \max_{G'.\mathtt{mo}} G'.\mathsf{W}_x$, reaching a contradiction.

  It remains to show that $\mathcal{B} \curlyvee G$. Let $\pi \in \mathsf{Tid}$ and $L \in \mathcal{B}(\pi)$. We show that there exists a $\langle G, \pi\rangle$-consistent $\langle G, L\rangle$-write-list W. Following the construction of $\mathcal{B}$, one of the following holds:
  - $L = L' \setminus \mathcal{P}(\pi, L')$ for some $L' \in \mathcal{B}'(\pi)$. Let $P = \mathcal{P}(\pi, L')$, $W' = W'_{\langle \pi, L'\rangle}$ and $f = \mathsf{Map}^{-1}_{\langle L', P\rangle}$. We define $W = \lambda k \in \{1, \ldots, |L|\}.\ W'(f(k))$. Using the fact that W' is a $\langle G', L'\rangle$-write-list, it is easy to see that W is a $\langle G, L\rangle$-write-list. (In particular, note that $\mathtt{rmw}(L(k)) \neq \mathtt{RMW}$ whenever $\mathtt{loc}(L(k)) = x$.)

    It remains to show that W is $\langle G, \pi\rangle$-consistent, namely to prove that for every k, we have $W(k) \notin \mathit{dom}(G.\mathtt{mo}\,;\, G.\mathtt{hb}^?\,;\, [\mathsf{E}^\pi \cup \{W(j) \mid 1 \le j < k\}])$. Indeed, in case that we have $W(k) \in \mathit{dom}(G.\mathtt{mo}\,;\, G.\mathtt{hb}^?\,;\, [\mathsf{E}^\pi])$, since $G.\mathtt{mo} \subseteq G'.\mathtt{mo}$ and $G.\mathtt{hb} \subseteq G'.\mathtt{hb}$, it follows that $W'(f(k)) \in \mathit{dom}(G'.\mathtt{mo}\,;\, G'.\mathtt{hb}^?\,;\, [\mathsf{E}^\pi])$, which contradicts the $\langle G', \pi\rangle$-consistency of W'. Analogously, if $W(k) \in \mathit{dom}(G.\mathtt{mo}\,;\, G.\mathtt{hb}^?\,;\, \{W(j) \mid 1 \le j < k\})$ then since f is an increasing function, we have $W'(f(k)) \in \mathit{dom}(G'.\mathtt{mo}\,;\, G'.\mathtt{hb}^?\,;\, \{W'(f(j)) \mid 1 \le j < k\})$, which contradicts the $\langle G', \pi\rangle$-consistency of W'.
  - $\pi = \tau$ and $L = L' \setminus\!\setminus \mathcal{P}(\eta, L')$ for some $\eta \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\eta)$ such that $\mathcal{P}(\eta, L') \neq \emptyset$. Let $P = \mathcal{P}(\eta, L')$, $m = \min(P)$, $W' = W'_{\langle \eta, L'\rangle}$ and $f = \mathsf{MMap}^{-1}_{\langle L', P\rangle}$. We define $W = \lambda k \in \{1, \ldots, |L|\}.\ W'(f(k))$. Using the fact that W' is a $\langle G', L'\rangle$-write-list, it is easy to see that W is a $\langle G, L\rangle$-write-list. (In particular, note that $\mathtt{rmw}(L(k)) \neq \mathtt{RMW}$ whenever $\mathtt{loc}(L(k)) = x$.)

It remains to show that W is $\langle G, \tau\rangle$-consistent, namely that for every k we have $W(k) \notin dom(G.\mathsf{mo}\,;G.\mathsf{hb}^?\,;[\mathsf{E}^\tau \cup \{W(j) \mid 1 \le j < k\}])$. Indeed, since f is increasing, if we have $W(k) \in dom(G.\mathsf{mo}\,;G.\mathsf{hb}^?\,;[\{W(j) \mid 1 \le j < k\}])$ for some k, then we also have that $W'(f(k)) \in dom(G'.\mathsf{mo}\,;G'.\mathsf{hb}^?\,;[\{W'(j) \mid 1 \le j < f(k)\}])$, which contradicts the $\langle G', \eta\rangle$-consistency of W'. Now, if $W(k) \in dom(G.\mathsf{mo}\,;G.\mathsf{hb}^?\,;[\mathsf{E}^\tau])$, then since $w = \max_{G'.\mathsf{po}} G'.\mathsf{E}^\tau$, we have that $\langle W'(f(k)), w\rangle \in G'.\mathsf{mo}\,;G'.\mathsf{hb}^?$. However, we have $W'(m) = w$ and $f(k) > m$, from which it follows that $W'(f(k)) \in dom(G'.\mathsf{mo}\,;G'.\mathsf{hb}^?\,;[\{W'(j) \mid 1 \le j < f(k)\}])$, which contradicts the $\langle G', \eta\rangle$-consistency of W'.

- READ step, $l = \mathsf{R}(x, v_\mathsf{R})$:

  Let $r = \mathsf{NextEvent}(G.\mathsf{E}, \tau, l)$. Since $G \xrightarrow{\tau,l}_{\mathsf{opSRA}} G'$, we have $G'.\mathsf{E} = G.\mathsf{E} \cup \{r\}$, $G'.\mathsf{rf} = G.\mathsf{rf} \cup \{\langle w, r\rangle\}$ and $G'.\mathsf{mo} = G.\mathsf{mo}$, for some write event $w \in G.\mathsf{W}_x$ such that $\mathtt{val}_\mathsf{W}(w) = v_\mathsf{R}$ and $w \notin dom(G.\mathsf{mo}\,;G.\mathsf{hb}^?\,;[\mathsf{E}^\tau])$.
  Let o be the read option given by $o \triangleq \mathtt{O}_\mathsf{R}(\mathtt{tid}(w), x, v_\mathsf{R}, \mathsf{R})$. We define $\mathcal{B}$ by:

$$\mathcal{B} \triangleq \lambda\pi \in \mathsf{Tid}.\ \begin{cases} o \cdot \mathcal{B}'(\tau) & \pi = \tau \\ \mathcal{B}'(\pi) & \pi \neq \tau \end{cases}$$

  By definition, $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loSRA}} \mathcal{B}'$.
  We show next that $\mathcal{B} \lhd G$. For a thread $\pi \neq \tau$ and an option list $L \in \mathcal{B}(\pi)$, observe that $L \in \mathcal{B}'(\pi)$, and since $\mathcal{B}' \lhd G'$, there is a $\langle G', \pi\rangle$-consistent $\langle G', L\rangle$-write-list W'. Since $G.\mathsf{mo} \subseteq G'.\mathsf{mo}$ and $G.\mathsf{hb} \subseteq G'.\mathsf{hb}$, W' is also $\langle G, \pi\rangle$-consistent $\langle G, L\rangle$-write-list.
  Consider an option list $L \in \mathcal{B}(\tau)$. Let $L' \in \mathcal{B}'(\tau)$ such that $L = o \cdot L'$. Since $\mathcal{B}' \lhd G'$, there is a $\langle G', \tau\rangle$-consistent $\langle G', L'\rangle$-write-list W'. Define $W \triangleq w \cdot W'$. Using the fact that W' is a $\langle G', L'\rangle$-write-list, it is easy to see that W is a $\langle G, L\rangle$-write-list. It is left to show that W is $\langle G, \tau\rangle$-consistent. For this matter, let $1 \le k \le |L|$. We prove that $W(k) \notin dom(G.\mathsf{mo}\,;G.\mathsf{hb}^?\,;[\mathsf{E}^\tau \cup \{W(j) \mid 1 \le j < k\}])$. Suppose otherwise. Consider the two possible cases:
  - $k = 1$. Then $w \in dom(G.\mathsf{mo}\,;G.\mathsf{hb}^?\,;[\mathsf{E}^\tau])$, which contradicts the properties of w as stated above.
  - $k > 1$. Observe that $W(k) = W'(k-1)$. If $W(k) \in dom(G.\mathsf{mo}\,;G.\mathsf{hb}^?\,;[\mathsf{E}^\tau \cup \{W(j) \mid 2 \le j < k\}])$ then $W'(k-1) \in dom(G'.\mathsf{mo}\,;G'.\mathsf{hb}^?\,;[\mathsf{E}^\tau \cup \{W'(j) \mid 1 \le j < k-1\}])$, contradicting the $\langle G', \tau\rangle$-consistency of W'. Thus, $\langle W(k), W(1)\rangle \in G.\mathsf{mo}\,;G.\mathsf{hb}^?$. Yet, $W(1) = w$, $r \in \mathsf{E}^\tau$ and $\langle w, r\rangle \in G'.\mathsf{rf}$. Hence, $W'(k-1) \in dom(G'.\mathsf{mo}\,;G'.\mathsf{hb}^?\,;[\mathsf{E}^\tau])$, contradicting the $\langle G', \tau\rangle$-consistency of W'.
- RMW step, $l = \mathsf{RMW}(x, v_\mathsf{R}, v_\mathsf{W})$:

  This case combines the proofs given for the read and write cases. Let $e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, l)$. Since $G \xrightarrow{\tau,l}_{\mathsf{opSRA}} G'$, we have $G'.\mathsf{E} = G.\mathsf{E} \cup \{e\}$, $G'.\mathsf{mo} = G.\mathsf{mo} \cup (G.\mathsf{W}_x \times \{e\})$, $G'.\mathsf{rf} = G.\mathsf{rf} \cup \{\langle w, e\rangle\}$ and $\mathtt{val}_\mathsf{W}(w) = v_\mathsf{R}$, where $w = \max_{G.\mathsf{mo}} \mathsf{W}_x$. Since $\mathcal{B}' \lhd G'$, for every $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$ there exists a $\langle G', \pi\rangle$-consistent $\langle G', L'\rangle$-write-list $W'_{\langle \pi, L'\rangle}$.
  Let $\mathcal{P}$ be the index choice for $\mathcal{B}'$ that assigns the set of "new" positions in $\mathcal{B}'$:

$$\mathcal{P} \triangleq \lambda\pi \in \mathsf{Tid}, L' \in \mathcal{B}'(\pi).\ \{1 \le k \le |L'| \mid W'_{\langle \pi, L'\rangle}(k) = e\}.$$

  Then, we define:

$$\mathcal{B} \triangleq \lambda\pi \in \mathsf{Tid}.\ \begin{cases} o \cdot \mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\tau) & \pi = \tau \\ \mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\pi) & \pi \neq \tau \end{cases}$$

  where o is the read option given by $o \triangleq \mathtt{O}_\mathsf{R}(\mathtt{tid}(w), x, v_\mathsf{R}, \mathsf{RMW})$.

The arguments for why $\mathcal{B} \xrightarrow{\tau,l}_{\text{loSRA}} \mathcal{B}'$ are analogous to those of the write case. Using Prop. 8.4, to show that $\mathcal{B} \xrightarrow{\tau,l}_{\text{loSRA}} \mathcal{B}'$, it suffices to prove that $\mathcal{P} \models_{\text{SRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}}) \rangle$. This is done exactly as in the write case.

It remains to show that $\mathcal{B} \lhd G$. Let $\pi \in \text{Tid}$ and $L \in \mathcal{B}(\pi)$. We show that there exists a $\langle G, \pi \rangle$-consistent $\langle G, L \rangle$-write-list W. Following the construction of $\mathcal{B}$, one of the following holds:

- $L = L' \setminus \mathcal{P}(\pi, L')$ for some $L' \in \mathcal{B}'(\pi)$. This case is exactly the same as the analogous case in the WRITE step.
- $\pi = \tau$ and $L = o \cdot (L' \setminus \mathcal{P}(\tau, L'))$ for some $L' \in \mathcal{B}'(\tau)$. Let $P = \mathcal{P}(\tau, L')$, $W' = W'_{\langle \tau, L' \rangle}$ and $f = \lambda k \in \{2, \ldots, |L|\}.\ \text{Map}^{-1}_{\langle L', P \rangle}(k-1)$. We define

$$W \triangleq \lambda k \in \{1, \ldots, |L|\}. \begin{cases} w & k = 1 \\ W'(f(k)) & k > 1 \end{cases}$$

  Using the fact that W' is a $\langle G', L' \rangle$-write-list and that $w = \max_{G.\mathtt{mo}} \mathtt{W}_x$, it is easy to see that W is a $\langle G, L \rangle$-write-list. (In particular, note that for $k > 1$, $\mathtt{rmw}(L(k)) \neq \mathtt{RMW}$ whenever $\mathtt{loc}(L(k)) = x$.) It remains to show that W is $\langle G, \tau \rangle$-consistent, namely to prove that for every $k \in \{1, \ldots, |L|\}$, we have $W(k) \notin dom(G.\mathtt{mo}\,;\, G.\mathtt{hb}^?\,;\, [\mathtt{E}^\tau \cup \{W(j) \mid 1 \leq j < k\}])$. For $k = 1$, this is trivial since $W(1) = w = \max_{G.\mathtt{mo}} \mathtt{W}_x$. Let $k \in \{2, \ldots, |L|\}$. If $W(k) \in dom(G.\mathtt{mo}\,;\, G.\mathtt{hb}^?\,;\, [\mathtt{E}^\tau])$ then since $G.\mathtt{mo} \subseteq G'.\mathtt{mo}$ and $G.\mathtt{hb} \subseteq G'.\mathtt{hb}$, we have $W'(f(k)) \in dom(G'.\mathtt{mo}\,;\, G'.\mathtt{hb}^?\,;\, [\mathtt{E}^\tau])$, which contradicts the $\langle G', \tau \rangle$-consistency of W'. Analogously, if $\langle W(k), W(j) \rangle \in G.\mathtt{mo}\,;\, G.\mathtt{hb}^?$ for $2 \leq j < k$ then $\langle W'(f(k)), W'(f(j)) \rangle \in G'.\mathtt{mo}\,;\, G'.\mathtt{hb}^?$, and since f is an increasing function this contradicts the $\langle G', \tau \rangle$-consistency of W'. Now, if $\langle W(k), W(1) \rangle \in G.\mathtt{mo}\,;\, G.\mathtt{hb}^?$, then since $W(1) = w$, $G'.\mathtt{E} = G.\mathtt{E} \cup \{e\}$ and $G'.\mathtt{rf} = G.\mathtt{rf} \cup \{\langle w, e \rangle\}$, we have $W'(f(k)) \in dom(G'.\mathtt{mo}\,;\, G'.\mathtt{hb}^?\,;\, [G'.\mathtt{E}^\tau])$, which contradicts the $\langle G', \tau \rangle$-consistency of W'.
- $\pi = \tau$ and $L = o \cdot (L' \setminus\!\setminus \mathcal{P}(\eta, L'))$ for some $\eta \in \text{Tid}$ and $L' \in \mathcal{B}'(\eta)$. Let $P = \mathcal{P}(\eta, L')$, $m = \min(P)$, $W' = W'_{\langle \eta, L' \rangle}$ and $f = \lambda k \in \{2, \ldots, |L|\}.\ \text{MMap}^{-1}_{\langle L', P \rangle}(k-1)$.

  We define

$$W \triangleq \lambda k \in \{1, \ldots, |L|\}. \begin{cases} w & k = 1 \\ W'(f(k)) & k > 1 \end{cases}$$

  As above, W is a $\langle G, L \rangle$-write-list, and we show that it is $\langle G, \tau \rangle$-consistent. Namely, we prove that $W(k) \notin dom(G.\mathtt{mo}\,;\, G.\mathtt{hb}^?\,;\, [\mathtt{E}^\tau \cup \{W(j) \mid 1 \leq j < k\}])$ for every $k \in \{1, \ldots, |L|\}$. Again, for $k = 1$, this is trivial since $W(1) = w = \max_{G.\mathtt{mo}} \mathtt{W}_x$. Let $k \in \{2, \ldots, |L|\}$. If $\langle W(k), W(j) \rangle \in G.\mathtt{mo}\,;\, G.\mathtt{hb}^?$ for $2 \leq j < k$, then $\langle W'(f(k)), W'(f(j)) \rangle \in G'.\mathtt{mo}\,;\, G'.\mathtt{hb}^?$, and since f is an increasing function this contradicts the $\langle G', \eta \rangle$-consistency of W'. Now, if $\langle W(k), W(1) \rangle \in G.\mathtt{mo}\,;\, G.\mathtt{hb}^?$, then since $W(1) = w$ and $\langle w, e \rangle \in G'.\mathtt{rf}$, we have $\langle W'(f(k)), e \rangle \in G'.\mathtt{mo}\,;\, G'.\mathtt{hb}^?$. However, $W'(m) = e$ and $f(k) > m$ together imply that we have $W'(f(k)) \in dom(G'.\mathtt{mo}\,;\, G'.\mathtt{hb}^?\,;\, [\{W'(j) \mid 1 \leq j < f(k)\}])$, which contradicts the $\langle G', \eta \rangle$-consistency of W'.

  Lastly, if $W(k) \in dom(G.\mathtt{mo}\,;\, G.\mathtt{hb}^?\,;\, [\mathtt{E}^\tau])$, then $\langle W'(f(k)), e \rangle \in G'.\mathtt{mo}\,;\, G'.\mathtt{hb}^?$ (since $e = \max_{G'.\mathtt{po}} G'.\mathtt{E}^\tau$). However, $W'(m) = e$ and $f(k) > m$, implying that

$$W'(f(k)) \in dom(G'.\mathtt{mo}\,;\, G'.\mathtt{hb}^?\,;\, [\{W'(j) \mid 1 \leq j < f(k)\}]),$$

  which contradicts the $\langle G', \eta \rangle$-consistency of W'. □

A.2 Equivalence of loWRA and opWRA

LEMMA A.3. *For every trace of* loWRA *there is an equivalent trace of* opWRA.

Proof. As described in §7, we show that $\curlyvee$ constitutes a forward simulation relation from loWRA to opWRA. We detail here the simulation step. Suppose that $\mathcal{B} \curlyvee G$ and $\mathcal{B} \xrightarrow{\tau,l}_{\text{loWRA}} \mathcal{B}'$. Let $tid_{\mathsf{RMW}} : \mathbb{W} \to \mathsf{Tid}$ that satisfies the conditions of Def. 7.4. We show that there exists G' such that $\mathcal{B}' \curlyvee G'$ and $G \xrightarrow{\tau,l}_{\text{opWRA}} G'$. Consider the possible cases:[11]

- $l = \mathtt{W}(x, v_{\mathtt{W}})$: Let $w = \mathsf{NextEvent}(G.\mathsf{E}, \tau, l)$. Let G' be the execution graph defined by $G'.\mathsf{E} = G.\mathsf{E} \cup \{w\}$ and $G'.\mathtt{rf} = G.\mathtt{rf}$. By definition, we have $G \xrightarrow{\tau,l}_{\text{opWRA}} G'$.

  We show that $\mathcal{B}' \curlyvee G'$. First, since $\mathcal{B} \xrightarrow{\tau,l}_{\text{loWRA}} \mathcal{B}'$, by Prop. 8.4, there exists an index choice $\mathcal{P}$ for $\mathcal{B}'$ such that $\mathcal{P} \models_{\text{WRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$, $\mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\pi) \subseteq \mathcal{B}(\pi)$ for every $\pi \in \mathsf{Tid} \setminus \{\tau\}$ and $\mathtt{O}_{\mathtt{W}}(x) \cdot \mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\tau) \subseteq \mathcal{B}(\tau)$. Since $\mathcal{P} \models_{\text{WRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$, there exists $\pi_{\mathsf{RMW}} \in \mathsf{Tid}$, such that $L'(k) = \mathtt{O}_{\mathtt{R}}(\tau, x, v_{\mathtt{W}}, \pi_{\mathsf{RMW}})$ for every $\pi \in \mathsf{Tid}$, $L' \in \mathcal{B}'(\pi)$ and $k \in \mathcal{P}(\pi, L')$. (If $\mathcal{P}(\pi, L') = \emptyset$ for every $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$, then π_{RMW} is arbitrary.)
  Let $tid'_{\mathsf{RMW}} = tid_{\mathsf{RMW}}[w \mapsto \pi_{\mathsf{RMW}}]$. Since $w \notin G'.\mathtt{rf}$, we vacuously have $\mathtt{tid}(e) = tid'_{\mathsf{RMW}}(w)$ for every $\langle w, e\rangle \in G'.\mathtt{rf}$; [RMW]. It follows that for every $\langle w', e\rangle \in G'.\mathtt{rf}$; [RMW], we have $\mathtt{tid}(e) = tid'_{\mathsf{RMW}}(w)$.
  We show that for every $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$, there exists a $\langle G', \pi\rangle$-consistent $\langle G', L', tid'_{\mathsf{RMW}}\rangle$-write-list. Let $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$. We construct a $\langle G', \pi\rangle$-consistent $\langle G', L', tid'_{\mathsf{RMW}}\rangle$-write-list W'. Let $P \triangleq \mathcal{P}(\pi, L')$ and (L_τ and f_τ are only defined if $P \neq \emptyset$):

$$L \triangleq \begin{cases} L' \setminus P & \pi \neq \tau \\ \mathtt{O}_{\mathtt{W}}(x) \cdot (L' \setminus P) & \pi = \tau \end{cases} \qquad f \triangleq \begin{cases} \mathsf{Map}_{\langle L', P\rangle} & \pi \neq \tau \\ \lambda k \in \{1, \ldots, |L'|\} \setminus P.\ \mathsf{Map}_{\langle L', P\rangle}(k) + 1 & \pi = \tau \end{cases}$$

$$L_\tau \triangleq \mathtt{O}_{\mathtt{W}}(x) \cdot L' \setminus\!\setminus P \qquad f_\tau \triangleq \lambda k \in \{\min(P), \ldots, |L'|\} \setminus P.\ \mathsf{MMap}_{\langle L', P\rangle}(k) + 1$$

  Then, by definition, we have $L \in \mathcal{B}(\pi)$ and $L_\tau \in \mathcal{B}(\tau)$. Let W be a $\langle G, \pi\rangle$-consistent $\langle G, L, tid_{\mathsf{RMW}}\rangle$-write-list, and W_τ be a $\langle G, \tau\rangle$-consistent $\langle G, L_\tau, tid_{\mathsf{RMW}}\rangle$-write-list. Note that for every $k > \min(P)$ with $k \notin P$ and $\mathtt{typ}(L'(k)) = \mathtt{R}$, we have $\mathtt{tid}(W(f(k))) = \mathtt{tid}(L(f(k))) = \mathtt{tid}(L_\tau(f_\tau(k))) = \mathtt{tid}(W_\tau(f_\tau(k)))$, and so $G.\mathsf{hb}$ must order the two write events, $W(f(k))$ and $W_\tau(f_\tau(k))$.
  We define W' as follows:

$$W' \triangleq \lambda k \in \{1, \ldots, |L'|\}.\ \begin{cases} L'(k) & \mathtt{typ}(L'(k)) = \mathtt{W} \\ w & \mathtt{typ}(L'(k)) = \mathtt{R} \wedge k \in P \\ W(f(k)) & \mathtt{typ}(L'(k)) = \mathtt{R} \wedge k < \min(P) \\ \max_{G.\mathsf{hb}}\{W(f(k)), W_\tau(f_\tau(k))\} & \text{otherwise} \end{cases}$$

  It is easy to see that W' is a $\langle G', L', tid'_{\mathsf{RMW}}\rangle$-write-list. We show that W' is $\langle G', \pi\rangle$-consistent. Let $1 \leq k \leq |L'|$ such that $W'(k) \in \mathsf{E}$. Let $y = \mathtt{loc}(W'(k))$, $w_\pi = W(f(k))$ and $w_\tau = W_\tau(f_\tau(k))$ (the latter is only defined if $k > \min(P)$). We prove that each of the conditions in Def. 7.3 holds:

C1) The proof is exactly as for SRA. We note that if $\pi = \tau$, then we cannot have $\langle w_\pi, w\rangle \in G'.\mathsf{hb}|_{\mathtt{loc}}$. Indeed, otherwise, we have $w_\pi \in dom(G.\mathsf{hb}^?\ ;\ [\mathsf{E}^\tau])$, and since $W(1) = L(1) = \mathtt{O}_{\mathtt{W}}(x) = \mathtt{O}_{\mathtt{W}}(\mathtt{loc}(w_\pi))$, this contradicts the fact that W is $\langle G, \tau\rangle$-consistent. Similarly, we cannot have $\langle w_\tau, w\rangle \in G'.\mathsf{hb}|_{\mathtt{loc}}$. Indeed, otherwise, we have $w_\tau \in dom(G.\mathsf{hb}^?\ ;\ [\mathsf{E}^\tau])$, and since $W_\tau(1) = L_\tau(1) = \mathtt{O}_{\mathtt{W}}(x) = \mathtt{O}_{\mathtt{W}}(\mathtt{loc}(w_\tau))$, this contradicts the fact that W_τ is $\langle G, \tau\rangle$-consistent.

C2) Suppose by contradiction that there exists $i < k$ with $W'(i) = \mathtt{O}_{\mathtt{W}}(y)$ but $W'(k) \in dom(G'.\mathsf{hb}^?\ ;\ [\mathsf{E}^\pi])$. Note that the definition of W' ensures that $W'(i) = L'(i) = \mathtt{O}_{\mathtt{W}}(y)$, and since W is a $\langle G, L, tid_{\mathsf{RMW}}\rangle$-write-list, it follows that $W(f(i)) = \mathtt{O}_{\mathtt{W}}(y)$. Consider the two possible cases:

[11] In WRA, the mo-component is immaterial and can be defined arbitrarily, so we ignore this component in this proof. To reduce the amount of duplication, when possible we refer to the corresponding case in the proof for SRA. To do that one should replace mo in the proof for SRA with $[\mathtt{W}]\ ;\ \mathsf{hb}|_{\mathtt{loc}}\ ;\ [\mathtt{W}]$ in the current proof.

* $W'(k) = w$: In this case, we must have $y = x$, $\pi = \tau$ and $i < \max(\mathcal{P}(\tau, L'))$. Since $\mathcal{P} \not\models_{\mathsf{WRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$, we cannot have $L'(i) = \mathtt{O}_{\mathtt{W}}(x)$.
* $W'(k) \neq w$: In this case, the definition of W' ensures that $\langle w_\pi, W'(k)\rangle \in G'.\mathsf{hb}|^?_{\mathsf{loc}}$, and so $w_\pi \in dom(G'.\mathsf{hb}^? ; [\mathtt{E}^\pi])$. Since $w_\pi \neq w$ (as $w_\pi \in G.\mathtt{E}$), it follows that $w_\pi \in dom(G.\mathsf{hb}^? ; [\mathtt{E}^\pi])$. Since $W(f(i)) = \mathtt{O}_{\mathtt{W}}(y)$, this contradicts the fact that W is $\langle G, \pi\rangle$-consistent.

C3) Suppose by contradiction that there exists $j < i < k$ with $W'(i) = \mathtt{O}_{\mathtt{W}}(y)$ but $\langle W'(k), W'(j)\rangle \in G'.\mathsf{hb}^?$. Note that the definition of W' ensures that $W'(i) = L'(i) = \mathtt{O}_{\mathtt{W}}(y)$, and since W is a $\langle G, L, tid_{\mathtt{RMW}}\rangle$-write-list, it follows that $W(f(i)) = \mathtt{O}_{\mathtt{W}}(y)$. In addition, since W_τ is $\langle G, L_\tau, tid_{\mathtt{RMW}}\rangle$-write-list, it follows that $W_\tau(f_\tau(i)) = \mathtt{O}_{\mathtt{W}}(y)$ if $i > \min(P)$. Consider the possible cases:
* $W'(k) = w$: In this case, we must have $y = x$ and $W'(j) = w$. It follows that $k, j \in P$, and since $\mathcal{P} \not\models_{\mathsf{WRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$, we cannot have $L'(i) = \mathtt{O}_{\mathtt{W}}(x)$.
* $W'(k) \neq w$ and $W'(j) = w$: In this case we must have $i, k > \min(P)$, and so $W'(k) = \max_{G.\mathsf{hb}}\{w_\pi, w_\tau\}$ and $W_\tau(f_\tau(i)) = \mathtt{O}_{\mathtt{W}}(y)$. Hence, we have $\langle w_\tau, W'(k)\rangle \in G.\mathsf{hb}|^?_{\mathsf{loc}}$, and so $\langle w_\tau, w\rangle \in G'.\mathsf{hb}^?$. Since $w_\tau \neq w$ (as $w_\tau \in G.\mathtt{E}$), it follows that $w_\tau \in dom(G.\mathsf{hb}^? ; [\mathtt{E}^\tau])$. Since $W_\tau(f_\tau(i)) = \mathtt{O}_{\mathtt{W}}(y)$, this contradicts the fact that W_τ is $\langle G, \tau\rangle$-consistent.
* $W'(k) \neq w$ and $W'(j) \neq w$: In this case, we must have $\langle W'(k), W'(j)\rangle \in G.\mathsf{hb}^?$. Let $w^j_\pi = W(f(j))$ and $w^j_\tau = W_\tau(f_\tau(j))$ (the latter is only defined if $j > \min(P)$). Our construction ensures that one of the following holds:
  · $W'(j) = w^j_\pi$: Since $W'(k) \neq w$, the definition of W' ensures that $\langle w_\pi, W'(k)\rangle \in G'.\mathsf{hb}|^?_{\mathsf{loc}}$, and so $\langle w_\pi, w^j_\pi\rangle \in G.\mathsf{hb}^?$. This contradicts the fact that W is $\langle G, \pi\rangle$-consistent.
  · $W'(j) = w^j_\tau$: In this case we have $j > \min(P)$, and so $k > \min(P)$. Since $W'(k) \neq w$, the definition of W' ensures that $\langle w_\tau, W'(k)\rangle \in G'.\mathsf{hb}|^?_{\mathsf{loc}}$, and so $\langle w_\tau, w^j_\tau\rangle \in G.\mathsf{hb}^?$. This contradicts the fact that W_τ is $\langle G, \tau\rangle$-consistent.

- $l = \mathtt{R}(x, v_{\mathtt{R}})$:

By definition, since $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loWRA}} \mathcal{B}'$, there exists a read option o with $\mathtt{loc}(o) = x$ and $\mathtt{val}(o) = v_{\mathtt{R}}$ such that $\mathcal{B}(\tau) = o \cdot \mathcal{B}'(\tau)$. For every $L \in \mathcal{B}(\tau)$, let W_L be a $\langle G, \tau\rangle$-consistent $\langle G, L, tid_{\mathtt{RMW}}\rangle$-write-list. Let $A = \{W_L(1) \mid L \in \mathcal{B}(\tau)\}$. Since $\mathcal{B}(\tau)$ is non-empty, we know that A is not empty. Since each W_L is a $\langle G, L, tid_{\mathtt{RMW}}\rangle$-write-list, we have that $\mathtt{tid}(w) = \mathtt{tid}(o)$ for every $w \in A$. Hence, $G.\mathsf{po}$ totally orders A. Let $w = \min_{G.\mathsf{po}} A$ and let $L_{\min} \in \mathcal{B}(\tau)$ such that $w = W_{L_{\min}}(1)$. Let $r = \mathsf{NextEvent}(G.\mathtt{E}, \tau, l)$ and let G' be the execution graph given by $G'.\mathtt{E} = G.\mathtt{E} \cup \{r\}$ and $G'.\mathtt{rf} = G.\mathtt{rf} \cup \{\langle w, r\rangle\}$.

Now, $G \xrightarrow{\tau,l}_{\mathsf{opWRA}} G'$ follows exactly as in the proof for SRA. It remains to show that $\mathcal{B}' \vartriangleright G'$. We use the same $tid_{\mathtt{RMW}}$ mapping and show that for every $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$, there exists a $\langle G', \pi\rangle$-consistent $\langle G', L', tid_{\mathtt{RMW}}\rangle$-write-list. Let $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$. We define a $\langle G', \pi\rangle$-consistent $\langle G', L', tid_{\mathtt{RMW}}\rangle$-write-list. Consider two cases:

– $\pi \neq \tau$: By definition, since $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loWRA}} \mathcal{B}'$, we have $L' \in \mathcal{B}(\pi)$. Let W be a $\langle G, \pi\rangle$-consistent $\langle G, L', tid_{\mathtt{RMW}}\rangle$-write-list. It is easy to see that W is also a $\langle G', L', tid_{\mathtt{RMW}}\rangle$-write-list. It remains to show that W is $\langle G', \pi\rangle$-consistent. Condition C1 follows exactly as for SRA. To see that conditions C2 and C3 hold as well, note that if we have $W(k) \in dom(G'.\mathsf{hb}^? ; [\mathtt{E}^\pi])$ or $\langle W(k), W(j)\rangle \in G'.\mathsf{hb}^?$, then the same holds in G. Therefore, the $\langle G', \pi\rangle$-consistency of W directly follows from its $\langle G, \pi\rangle$-consistency.

– $\pi = \tau$: Let $L = o \cdot L'$. Then, $L \in \mathcal{B}(\tau)$. Let $W' = \lambda k \in \{1, \ldots, |L'|\}.\ W_L(1 + k)$. It is easy to see that W' is a $\langle G', L', tid_{\mathtt{RMW}}\rangle$-write-list. We show that W' is $\langle G', \tau\rangle$-consistent. Let $1 \leq k \leq |W'|$ such that $W'(k) \in \mathtt{E}$. Condition C1 follows exactly as for SRA. We prove conditions C2 and C3.

C2) Suppose by contradiction that there exists $i < k$ with $W'(i) = \mathtt{O_W}(\mathtt{loc}(W'(k)))$ (and so, $W_L(1{+}i) = \mathtt{O_W}(\mathtt{loc}(W_L(1+k)))$) but $W'(k) \in dom(G'.\mathsf{hb}^? ; [\mathsf{E}^\tau])$. If $W'(k) \in dom(G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$, then $W_L(1+k) \in dom(G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$, which contradicts the fact that W_L is $\langle G, \tau\rangle$-consistent. Hence, we must have $\langle W'(k), w\rangle \in G.\mathsf{hb}^?$. Since $L(1) = o$, the definition of w ensures that $\langle w, W_L(1)\rangle \in G.\mathsf{po}^?$. It follows that $\langle W_L(1+k), W_L(1)\rangle \in G.\mathsf{hb}$ while $W_L(1+i) = \mathtt{O_W}(\mathtt{loc}(W_L(1+k)))$ where $i < k$. Again, this contradicts the fact that W_L is $\langle G, \tau\rangle$-consistent.

C3) Suppose by contradiction that there exists $j < i < k$ with $W'(i) = \mathtt{O_W}(\mathtt{loc}(W'(k)))$ (and so, $W_L(1{+}i) = \mathtt{O_W}(\mathtt{loc}(W_L(1+k)))$) but $\langle W'(k), W'(j)\rangle \in G'.\mathsf{hb}^?$. In this case, since $W'(j) \in \mathsf{W}$, we must have $\langle W'(k), W'(j)\rangle \in G.\mathsf{hb}^?$. Hence, $\langle W_L(1+k)), W_L(1+j)\rangle \in G.\mathsf{hb}^?$ which contradicts the fact that W_L is $\langle G, \tau\rangle$-consistent.

- $l = \mathtt{RMW}(x, v_\mathsf{R}, v_\mathsf{W})$:

First, $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loWRA}} \mathcal{B}'$ provides us with the following:

(1) There exists a read option o with $\mathtt{loc}(o) = x$, $\mathtt{val}(o) = v_\mathsf{R}$ and $\mathtt{rmw\text{-}tid}(o) = \tau$ such that $L(1) = o$ for every $L \in \mathcal{B}(\tau)$.

(2) By Prop. 8.4, there exists an index choice $\mathcal{P}$ for $\mathcal{B}'$ such that $\mathcal{P} \models_{\mathsf{WRA}} \langle \tau, \mathsf{W}(x, v_\mathsf{W})\rangle, \mathrm{src}(\mathcal{B}', \tau, \mathcal{P})(\pi) \subseteq \mathcal{B}(\pi)$ for every $\pi \in \mathsf{Tid} \setminus \{\tau\}$ and $o \cdot \mathtt{O_W}(x) \cdot \mathrm{src}(\mathcal{B}', \tau, \mathcal{P})(\tau) \subseteq \mathcal{B}(\tau)$.

For every $L \in \mathcal{B}(\tau)$, let W_L be a $\langle G, \tau\rangle$-consistent $\langle G, L, tid_{\mathtt{RMW}}\rangle$-write-list. Let $A = \{W_L(1) \mid L \in \mathcal{B}(\tau)\}$. Since $\mathcal{B}(\tau)$ is non-empty, we know that A is not empty. Since each W_L is a $\langle G, L, tid_{\mathtt{RMW}}\rangle$-write-list, we have that $\mathtt{tid}(w) = \mathtt{tid}(o)$ for every $w \in A$. Hence, $G.\mathsf{po}$ totally orders A. Let $w = \min_{G.\mathsf{po}} A$ and let $L_{\min} \in \mathcal{B}(\tau)$ such that $w = W_{L_{\min}}(1)$. Let $e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, l)$ and let G' be the execution graph given by $G'.\mathsf{E} = G.\mathsf{E} \cup \{e\}$ and $G'.\mathsf{rf} = G.\mathsf{rf} \cup \{\langle w, e\rangle\}$.

Note that $w = W_{L_{\min}}(1)$, and since $W_{L_{\min}}$ is a $\langle G, L_{\min}, tid_{\mathtt{RMW}}\rangle$-write-list, we have that:

- $w \in G.\mathsf{W}$.
- $\mathtt{loc}(w) = \mathtt{loc}(W_{L_{\min}}(1)) = \mathtt{loc}(L_{\min}(1)) = \mathtt{loc}(o) = x$.
- $\mathtt{val_W}(w) = \mathtt{val_W}(W_{L_{\min}}(1)) = \mathtt{val}(L_{\min}(1)) = \mathtt{val}(o) = v_\mathsf{R}$.
- $tid_{\mathtt{RMW}}(w) = tid_{\mathtt{RMW}}(W_{L_{\min}}(1)) = \mathtt{rmw\text{-}tid}(L_{\min}(1)) = \tau$.

Then, to show that $G \xrightarrow{\tau,l}_{\mathsf{opWRA}} G'$, it suffices, by definition, to show the following:

- $w \notin dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$: Since $W_{L_{\min}}$ is $\langle G, \tau\rangle$-consistent and $w = W_{L_{\min}}(1)$, we cannot have $w \in dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$.
- $w \notin dom(G.\mathsf{rf} ; [\mathsf{RMW}])$: Suppose otherwise, and let $e' \in \mathsf{RMW}$ such that $\langle w, e'\rangle \in G.\mathsf{rf}$. Then, since $tid_{\mathtt{RMW}}(w) = \tau$, the second condition for WRA in Def. 7.4 ensures that $\mathtt{tid}(e) = \tau$. Hence, $w \in dom(G.\mathsf{rf} ; [\mathsf{RMW} \cap \mathsf{E}^\tau]) \subseteq dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$, which contradicts the previous item.

It remains to show that $\mathcal{B}' \lhd G'$. Since $\mathcal{P} \models_{\mathsf{WRA}} \langle \tau, \mathsf{W}(x, v_\mathsf{W})\rangle$, there exists $\pi_{\mathtt{RMW}} \in \mathsf{Tid}$, such that $L'(k) = \mathtt{O_R}(\tau, x, v_\mathsf{W}, \pi_{\mathtt{RMW}})$ for every $\pi \in \mathsf{Tid}$, $L' \in \mathcal{B}'(\pi)$ and $k \in \mathcal{P}(\pi, L')$.

Let $tid'_{\mathtt{RMW}} = tid_{\mathtt{RMW}}[w \mapsto \pi_{\mathtt{RMW}}]$. Since $e \notin G'.\mathsf{rf}$, we vacuously have $\mathtt{tid}(e') = tid'_{\mathtt{RMW}}(e)$ for every $\langle e, e'\rangle \in G'.\mathsf{rf} ; [\mathsf{RMW}]$. In addition, we have $\mathtt{tid}(e) = \tau = tid_{\mathtt{RMW}}(w) = tid'_{\mathtt{RMW}}(w)$. Since w is the unique event such that $\langle w, e\rangle \in G'.\mathsf{rf}$, it follows that for every $\langle w', e'\rangle \in G'.\mathsf{rf} ; [\mathsf{RMW}]$, we have $\mathtt{tid}(e') = tid'_{\mathtt{RMW}}(w')$.

We show that for every $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$, there exists a $\langle G', \pi\rangle$-consistent $\langle G', L', tid'_{\mathtt{RMW}}\rangle$-write-list. Let $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$. We construct a $\langle G', \pi\rangle$-consistent $\langle G', L', tid'_{\mathtt{RMW}}\rangle$-write-list W'. Let $P \triangleq \mathcal{P}(\pi, L')$ and (L_τ and f_τ are only defined if $P \neq \emptyset$):

$$L \triangleq \begin{cases} L' \setminus P & \pi \neq \tau \\ o \cdot \mathtt{O_W}(x) \cdot (L' \setminus P) & \pi = \tau \end{cases} \qquad f \triangleq \begin{cases} \mathsf{Map}_{\langle L', P\rangle} & \pi \neq \tau \\ \lambda k \in \{1, \ldots, |L'|\} \setminus P.\ \mathsf{Map}_{\langle L', P\rangle}(k) + 2 & \pi = \tau \end{cases}$$

$$L_\tau \triangleq o \cdot \mathtt{O_W}(x) \cdot L' \setminus\setminus P \qquad f_\tau \triangleq \lambda k \in \{\min(P), \ldots, |L'|\} \setminus P.\ \mathsf{MMap}_{\langle L', P\rangle}(k) + 2$$

Then, by definition, we have $L \in \mathcal{B}(\pi)$ and $L_\tau \in \mathcal{B}(\tau)$. Let W be a $\langle G, \pi\rangle$-consistent $\langle G, L, tid_{\mathtt{RMW}}\rangle$-write-list, and W_τ be a $\langle G, \tau\rangle$-consistent $\langle G, L_\tau, tid_{\mathtt{RMW}}\rangle$-write-list. Note that for every $k > \min(P)$ with $k \notin P$ and $\mathtt{typ}(L'(k)) = \mathtt{R}$, we have $\mathtt{tid}(W(f(k))) = \mathtt{tid}(L(f(k))) = \mathtt{tid}(L_\tau(f_\tau(k))) = \mathtt{tid}(W_\tau(f_\tau(k))$, and so $G.\mathsf{hb}$ must order the two write events, $W(f(k))$ and $W_\tau(f_\tau(k))$. We define W' as follows:

$$W' \triangleq \lambda k \in \{1, \ldots, |L'|\}.\ \begin{cases} L'(k) & \mathtt{typ}(L'(k)) = \mathtt{W} \\ e & \mathtt{typ}(L'(k)) = \mathtt{R} \wedge k \in P \\ W(f(k)) & \mathtt{typ}(L'(k)) = \mathtt{R} \wedge k < \min(P) \\ \max_{G.\mathsf{hb}}\{W(f(k)), W_\tau(f_\tau(k))\} & \text{otherwise} \end{cases}$$

It is easy to see that W' is a $\langle G', L', tid'_{\mathtt{RMW}}\rangle$-write-list. We show that W' is $\langle G', \pi\rangle$-consistent. Let $1 \le k \le |L'|$ such that $W'(k) \in \mathsf{E}$. Let $y = \mathtt{loc}(W'(k))$, $w_\pi = W(f(k))$ and $w_\tau = W_\tau(f_\tau(k))$ (the latter is only defined if $k > \min(P)$). We prove that each of the conditions in Def. 7.3 holds:

C1) We prove that $W'(k) \notin dom(G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G'.\mathsf{hb}^? ; [\mathsf{E}^\pi \cup \{W'(j) \mid 1 \le j < k\}])$. Suppose otherwise. First, note that we cannot have $k \in P$, since e is a maximal element in $G'.\mathsf{hb}$. Consider the two possible cases:
- $W'(k) \in dom(G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G'.\mathsf{hb}^? ; [\mathsf{E}^\pi])$: The definition of W' ensures that we have $\langle w_\pi, W'(k)\rangle \in G'.\mathsf{hb}|^?_{\mathtt{loc}}$, and so $w_\pi \in dom(G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G'.\mathsf{hb}^? ; [\mathsf{E}^\pi])$. Since W is $\langle G, \pi\rangle$-consistent, we have that $w_\pi \notin dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\pi])$, and therefore it must be the case that $\langle w_\pi, e\rangle \in G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; (G.\mathsf{hb} ; G'.\mathsf{rf})^?$ and $\pi = \tau$. Now, if $w_\pi \in dom(G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$, then since $\pi = \tau$, we have $W(2) = L(2) = \mathtt{O_W}(x) = \mathtt{O_W}(\mathtt{loc}(w_\pi))$, and we obtain a contradiction to the fact that W is $\langle G, \tau\rangle$-consistent. Otherwise, we have $\langle w_\pi, w\rangle \in G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}$. Since $\pi = \tau$, we have $L(1) = o$, and the definition of w ensures that $\langle w, W(1)\rangle \in G.\mathsf{po}^?$. It follows that $\langle w_\pi, W(1)\rangle \in G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^?$, which again contradicts the fact that W is $\langle G, \tau\rangle$-consistent.
- $\langle W'(k), W'(j)\rangle \in G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G'.\mathsf{hb}^?$ for some $1 \le j < k$. Consider the two possible cases:
  - $W'(j) = e$: In this case we must have $k > \min(P)$, and so $W'(k) = \max_{G.\mathsf{hb}}\{w_\pi, w_\tau\}$. Hence, we have $\langle w_\tau, W'(k)\rangle \in G.\mathsf{hb}|^?_{\mathtt{loc}}$. There are four possibilities:
    - $W'(k) = w$: In this case we have $\langle w_\tau, w\rangle \in G.\mathsf{hb}|^?_{\mathtt{loc}}$. Since $L_\tau(1) = o$, the definition of w ensures that $\langle w, W_\tau(1)\rangle \in G.\mathsf{po}^?$. Hence, $\langle w_\tau, W_\tau(1)\rangle \in G.\mathsf{hb}^?$. Since $L_\tau(2) = \mathtt{O_W}(x) = \mathtt{O_W}(\mathtt{loc}(W_\tau(f_\tau(k))))$, we obtain a contradiction to the fact that W_τ is $\langle G, \tau\rangle$-consistent.
    - $\langle W'(k), w\rangle \in G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G'.\mathsf{hb}^?$: This contradicts the $\langle G, \tau\rangle$-consistency of W_τ, as $\langle w, W_\tau(1)\rangle \in G.\mathsf{po}^?$ and $\langle w_\tau, W'(k)\rangle \in G'.\mathsf{hb}|^?_{\mathtt{loc}}$, implying that $\langle w_\tau, W_\tau(1)\rangle \in G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^?$.
    - $\langle W'(k), e\rangle \in G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G'.\mathsf{hb}^? ; G'.\mathsf{po}$: This also contradicts the $\langle G, \tau\rangle$-consistency of W_τ, as we get that $w_\tau \in dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$.
    - $y = x$ and $\langle W'(k), e\rangle \in G'.\mathsf{hb} ; G'.\mathsf{po}$: In this case we have $\langle w_\tau, e\rangle \in G'.\mathsf{hb}|^?_{\mathtt{loc}} ; G'.\mathsf{hb} ; G'.\mathsf{po}$, and so $w_\tau \in dom(G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$. But, since $W_\tau(2) = L_\tau(2) = \mathtt{O_W}(x) = \mathtt{O_W}(\mathtt{loc}(w_\tau))$, we obtain a contradiction to the fact that W_τ is $\langle G, \tau\rangle$-consistent.
  - $W'(j) \neq e$: This case is proved exactly as the corresponding case in the proof for SRA.

C2) Suppose by contradiction that there exists $i < k$ with $W'(i) = \mathtt{O_W}(y)$ but $W'(k) \in dom(G'.\mathsf{hb}^? ; [\mathsf{E}^\pi])$. Note that the definition of W' ensures that $W'(i) = L'(i) = \mathtt{O_W}(y)$, and since W is a $\langle G, L, tid_{\mathtt{RMW}}\rangle$-write-list, it follows that $W(f(i)) = \mathtt{O_W}(y)$. Consider the two possible cases:
- $W'(k) = e$: In this case, we must have $y = x$, $\pi = \tau$ and $i < \max(\mathcal{P}(\tau, L'))$. Since $\mathcal{P} \models_{\mathsf{WRA}} \langle \tau, \mathtt{W}(x, v_{\mathtt{W}})\rangle$, we cannot have $L'(i) = \mathtt{O_W}(x)$.

– $W'(k) \neq e$: In this case, the definition of W' ensures that $\langle w_\pi, W'(k)\rangle \in G'.\mathsf{hb}|^?_{\mathsf{loc}}$, and so $w_\pi \in dom(G'.\mathsf{hb}^? ; [\mathsf{E}^\pi])$. Since $w_\pi \neq e$ (as $w_\pi \in G.\mathsf{E}$), it follows that $w_\pi \in dom(G.\mathsf{hb}^? ; [\mathsf{E}^\pi])$. Since $W(f(i)) = \mathsf{O_W}(y)$, this contradicts the fact that W is $\langle G, \pi\rangle$-consistent.

C3) Suppose by contradiction that there exists $j < i < k$ with $W'(i) = \mathsf{O_W}(y)$ but $\langle W'(k), W'(j)\rangle \in G'.\mathsf{hb}^?$. Note that the definition of W' ensures that $W'(i) = L'(i) = \mathsf{O_W}(y)$, and since W is a $\langle G, L, tid_{\mathsf{RMW}}\rangle$-write-list, it follows that $W(f(i)) = \mathsf{O_W}(y)$. In addition, since W_τ is $\langle G, L_\tau, tid_{\mathsf{RMW}}\rangle$-write-list, it follows that $W_\tau(f_\tau(i)) = \mathsf{O_W}(y)$ if $i > \min(P)$. Consider the possible cases:

– $W'(k) = e$: In this case, we must have $y = x$ and $W'(j) = e$. It follows that $k, j \in P$, and since $\mathcal{P} \models_{\mathsf{WRA}} \langle \tau, \mathsf{W}(x, v_\mathsf{W})\rangle$, we cannot have $L'(i) = \mathsf{O_W}(x)$.

– $W'(k) \neq e$ and $W'(j) = e$: In this case we must have $i, k > \min(P)$, and so $W'(k) = \max_{G.\mathsf{hb}}\{w_\pi, w_\tau\}$ and $W_\tau(f_\tau(i)) = \mathsf{O_W}(y)$. Hence, we have $\langle w_\tau, W'(k)\rangle \in G.\mathsf{hb}|^?_{\mathsf{loc}}$, and so $\langle w_\tau, e\rangle \in G'.\mathsf{hb}^?$. Since $w_\tau \neq e$ (as $w_\tau \in G.\mathsf{E}$), it follows that $w_\tau \in dom(G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$. Since $W_\tau(f_\tau(i)) = \mathsf{O_W}(y)$, this contradicts the fact that W_τ is $\langle G, \tau\rangle$-consistent.

– $W'(k) \neq e$ and $W'(j) \neq e$: In this case, we must have $\langle W'(k), W'(j)\rangle \in G.\mathsf{hb}^?$. Let $w^j_\pi = W(f(j))$ and $w^j_\tau = W_\tau(f_\tau(j))$ (the latter is only defined if $j > \min(P)$). Our construction ensures that one of the following holds:

  * $W'(j) = w^j_\pi$: Since $W'(k) \neq e$, the definition of W' ensures that $\langle w_\pi, W'(k)\rangle \in G'.\mathsf{hb}|^?_{\mathsf{loc}}$, and so $\langle w_\pi, w^j_\pi\rangle \in G.\mathsf{hb}^?$. This contradicts the fact that W is $\langle G, \pi\rangle$-consistent.
  * $W'(j) = w^j_\tau$: In this case we have $j > \min(P)$, and so $k > \min(P)$. Since $W'(k) \neq e$, the definition of W' ensures that $\langle w_\tau, W'(k)\rangle \in G'.\mathsf{hb}|^?_{\mathsf{loc}}$, and so $\langle w_\tau, w^j_\tau\rangle \in G.\mathsf{hb}^?$. This contradicts the fact that W_τ is $\langle G, \tau\rangle$-consistent.

Finally, the LOWER step is handled exactly as for SRA. □

LEMMA A.4. *For every trace of* opWRA *there is an equivalent trace of* loWRA.

PROOF. As described in §7, we show that $\curlyvee^{-1}$ constitutes a backward simulation from opWRA to loWRA. We detail here the simulation step. Suppose that $G \xrightarrow{\tau,l}_{\mathsf{opWRA}} G'$ and $\mathcal{B}' \curlyvee G'$. Let $tid_{\mathsf{RMW}} : \mathsf{W} \to \mathsf{Tid}$ be a function satisfying the conditions of Def. 7.4, and for every $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$, we let $W'_{\langle \pi, L'\rangle}$ be a $\langle G', \pi\rangle$-consistent $\langle G', L', tid_{\mathsf{RMW}}\rangle$-write-list. We construct a state $\mathcal{B}$ such that $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loWRA}} \mathcal{B}'$ and $\mathcal{B} \curlyvee G$. Consider the possible cases:

- $l = \mathsf{W}(x, v_\mathsf{W})$:

  Let $w = \mathsf{NextEvent}(G.\mathsf{E}, \tau, l)$. Since $G \xrightarrow{\tau,l}_{\mathsf{opWRA}} G'$, we have $G'.\mathsf{E} = G.\mathsf{E} \cup \{w\}$ and $G'.\mathtt{rf} = G.\mathtt{rf}$.[12] Let $\mathcal{P}$ be the index choice for $\mathcal{B}'$ that assigns the set of "new" positions in $\mathcal{B}'$:

  $$\mathcal{P} \triangleq \lambda\pi \in \mathsf{Tid}, L' \in \mathcal{B}'(\pi).\ \{1 \leq k \leq |L'| \mid W'_{\langle \pi, L'\rangle}(k) = w\}.$$

  Then, we define

  $$\mathcal{B} \triangleq \lambda\pi \in \mathsf{Tid}.\ \begin{cases} \mathsf{O_W}(x) \cdot \mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\tau) & \pi = \tau \\ \mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\pi) & \pi \neq \tau \end{cases}$$

  By Prop. 8.4, to show that $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loWRA}} \mathcal{B}'$, it suffices to prove that $\mathcal{P} \models_{\mathsf{WRA}} \langle \tau, \mathsf{W}(x, v_\mathsf{W})\rangle$. Let $\pi_{\mathsf{RMW}} = tid_{\mathsf{RMW}}(w)$. Thus, we show that the following hold for every $\pi \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\pi)$, where $P = \mathcal{P}(\pi, L')$ and $W' = W'_{\langle \tau, L'\rangle}$:

  – Let $k \in P$. Then, we have $W'(k) = w$, and thus $L'(k) = \mathsf{O_R}(\tau, x, v_\mathsf{W}, \pi_{\mathsf{RMW}})$.

[12] As before, since the mo-component is immaterial in WRA, we ignore mo in this proof.

- Let $k \in \{1,\ldots,|L'|\} \setminus P$ such that $p_1 < k < p_2$ for some $p_1, p_2 \in P$. We show that $L'(k) \neq \mathtt{O_W}(x)$. Since $p_1, p_2 \in P$, we have $W'(p_1) = W'(p_2) = w$, and so $\langle W'(p_1), W'(p_2)\rangle \in G'.\mathsf{hb}^?$. Since W' is $\langle G', \tau\rangle$-consistent (by C3), we *cannot* have $W'(k) = \mathtt{O_W}(\mathtt{loc}(W'(p_2)))$, and so $L'(k) \neq \mathtt{O_W}(x)$.
- Suppose that $\pi = \tau$ and let $k \in \{1,\ldots,|L'|\} \setminus P$ such that $k < p$ for some $p \in P$. We show that $L'(k) \neq \mathtt{O_W}(x)$. Since $p \in P$, we have $W'(p) = w$, and so $W'(p) \in dom(G'.\mathsf{hb}^? ; [\mathtt{E}^\tau])$. Since W' is $\langle G', \tau\rangle$-consistent (by C2), we *cannot* have $W'(k) = \mathtt{O_W}(\mathtt{loc}(W'(p)))$, and so $L'(k) \neq \mathtt{O_W}(x)$.

Next, we prove that $\mathcal{B} \vee G$, by showing that for every $\pi \in \mathsf{Tid}$ and $L \in \mathcal{B}(\pi)$, there exists a $\langle G, \pi\rangle$-consistent $\langle G, L, tid_{\mathtt{RMW}}\rangle$-write-list. (Since $G.\mathtt{rf} \subseteq G'.\mathtt{rf}$, the second condition of $\vee$ for WRA (Def. 7.4), namely that for every $\langle w, e\rangle \in G.\mathtt{rf} ; [\mathsf{RMW}]$, we have $\mathtt{tid}(e) = tid_{\mathtt{RMW}}(w)$, trivially holds.) Let $\pi \in \mathsf{Tid}$ and $L \in \mathcal{B}(\pi)$. Following the construction of $\mathcal{B}$, one of the following holds:

- $\pi \neq \tau$ and $L = L' \setminus \mathcal{P}(\pi, L')$ for some $L' \in \mathcal{B}'(\pi)$. Let $P = \mathcal{P}(\pi, L')$, $W' = W'_{\langle \pi, L'\rangle}$ and $f = \mathsf{Map}^{-1}_{\langle L', P\rangle}$. We define $W \triangleq \lambda k \in \{1,\ldots,|L|\}.\ W'(f(k))$. Using the fact that W' is a $\langle G', L', tid_{\mathtt{RMW}}\rangle$-write-list, it is easy to see that W is a $\langle G, L, tid_{\mathtt{RMW}}\rangle$-write-list.
  It remains to show that W is $\langle G, \pi\rangle$-consistent, namely to prove that for every k, such that $W(k) \in \mathtt{E}$, the conditions of Def. 7.3 hold. Indeed, the construction of W and the fact that $G.\mathsf{hb} \subseteq G'.\mathsf{hb}$ directly ensure that these conditions follows from the $\langle G', \pi\rangle$-consistency of W'.
- $\pi = \tau$ and $L = \mathtt{O_W}(x) \cdot (L' \setminus \mathcal{P}(\tau, L'))$ for some $L' \in \mathcal{B}'(\tau)$. Let $P = \mathcal{P}(\tau, L')$, $W' = W'_{\langle \tau, L'\rangle}$ and $f = \lambda k \in \{2,\ldots,|L|\}.\ \mathsf{Map}^{-1}_{\langle L', P\rangle}(k-1)$. We define:

$$W \triangleq \lambda k \in \{1,\ldots,|L|\}.\ \begin{cases} \mathtt{O_W}(x) & k = 1 \\ W'(f(k)) & k > 1 \end{cases}$$

  By the fact that W' is a $\langle G', L', tid_{\mathtt{RMW}}\rangle$-write-list, we get that W is a $\langle G, L, tid_{\mathtt{RMW}}\rangle$-write-list. It remains to show that it is $\langle G, \tau\rangle$-consistent. Conditions C1 and C3 in Def. 7.3 follow directly from the $\langle G', \tau\rangle$-consistency of W'. Condition C2, however, deserves more attention, as we added $\mathtt{O_W}(x)$ at the start of the list. Assume toward contradiction some k, such that $W(k) \in \mathtt{E}, \mathtt{loc}(W(k)) = x$ and $W(k) \in dom(G.\mathsf{hb}^? ; [\mathtt{E}^\tau])$. Then since $W'(f(k)) = W(k)$, $w = \max_{G'.\mathsf{po}} G'.\mathtt{E}^\tau$ and $\mathtt{loc}(W(k)) = \mathtt{loc}(w)$, we have $\langle W'(f(k)), w\rangle \in G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G'.\mathsf{hb}^?$, contradicting (C1 in) the $\langle G', \tau\rangle$-consistency of W'.
- $\pi = \tau$ and $L = \mathtt{O_W}(x) \cdot (L' \setminus\setminus \mathcal{P}(\eta, L'))$ for some $\eta \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\eta)$.
  Let $P = \mathcal{P}(\eta, L')$, $m = \min(P)$, $W' = W'_{\langle \eta, L'\rangle}$ and $f = \lambda k \in \{2,\ldots,|L|\}.\ \mathsf{MMap}^{-1}_{\langle L', P\rangle}(k-1)$. We define:

$$W \triangleq \lambda k \in \{1,\ldots,|L|\}.\ \begin{cases} \mathtt{O_W}(x) & k = 1 \\ W'(f(k)) & k > 1 \end{cases}$$

  By the fact that W' is a $\langle G', L', tid_{\mathtt{RMW}}\rangle$-write-list, we get that W is a $\langle G, L, tid_{\mathtt{RMW}}\rangle$-write-list. It remains to show that it is $\langle G, \tau\rangle$-consistent. Condition C3 follows directly from the $\langle G', \eta\rangle$-consistency of W'. We prove the other two conditions:

C1) The existence of some k, such that $W(k) \in dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\{W(j) \mid 1 \leq j < k\}])$ directly contradicts the same condition in the $\langle G', \eta\rangle$-consistency of W'. Now, assume toward contradiction some k, such that $W(k) \in dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathtt{E}^\tau])$. Then, since $W'(f(k)) = W(k)$, $f(k) > m$, $W'(m) = w$ and $w = \max_{G'.\mathsf{po}} G'.\mathtt{E}^\tau$, we have $W'(f(k)) \in dom(G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G'.\mathsf{hb}^? ; [\{W'(j) \mid 1 \leq j < f(k)\}])$, contradicting (C1 in) the $\langle G', \eta\rangle$-consistency of W'.

C2) Assume toward contradiction the existence of some $i < k$, such that $W(i) = \mathtt{O_W}(\mathtt{loc}(W(k)))$ and $W(k) \in dom(G.\mathsf{hb}^? ; [\mathtt{E}^\tau])$. First if $i = 1$, then $\mathtt{loc}(W(k)) = x$, and as above, since

$W'(f(k)) = W(k)$, $f(k) > m$, $W'(m) = w$ and $w = \max_{G'.\mathsf{po}} G'.\mathsf{E}^\tau$, we have $W'(f(k)) \in dom(G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G'.\mathsf{hb}^? ; [\{W'(j) \mid 1 \le j < f(k)\}])$, contradicting (C1 in) the $\langle G', \eta\rangle$-consistency of W'. Now, suppose that $i > 1$. Then, again, since $W'(f(k)) = W(k)$, $f(k) > f(i) > m$, $W'(m) = w$ and $w = \max_{G'.\mathsf{po}} G'.\mathsf{E}^\tau$, we have $\langle W'(f(k)), W'(m)\rangle \in G'.\mathsf{hb}^?$, contradicting (C3 in) the $\langle G', \eta\rangle$-consistency of W'.

- $l = \mathsf{R}(x, v_\mathsf{R})$:

  Let $r = \mathsf{NextEvent}(G.\mathsf{E}, \tau, l)$. Since $G \xrightarrow{\tau,l}_{\mathsf{opWRA}} G'$, we have that $G'.\mathsf{E} = G.\mathsf{E} \cup \{r\}$ and $G'.\mathtt{rf} = G.\mathtt{rf} \cup \{\langle w, r\rangle\}$ for some write event $w \in G.\mathsf{W}_x \setminus dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$ with $\mathtt{val}_\mathsf{W}(w) = v_\mathsf{R}$.
  Let $o = \langle \mathtt{tid}(w), x, v_\mathsf{R}, tid_{\mathsf{RMW}}(w)\rangle$. We define $\mathcal{B}$ by:

$$\mathcal{B} \triangleq \lambda\pi \in \mathsf{Tid}. \begin{cases} o \cdot \mathcal{B}'(\tau) & \pi = \tau \\ \mathcal{B}'(\pi) & \pi \neq \tau \end{cases}$$

  By definition, we have $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loWRA}} \mathcal{B}'$. We show that $\mathcal{B} \vartriangleright G$. Note that the second condition of $\vartriangleright$ for WRA (Def. 7.4) trivially holds, and we need to show that for every $\pi \in \mathsf{Tid}$ and $L \in \mathcal{B}(\pi)$, there exists a $\langle G, \pi\rangle$-consistent $\langle G, L, tid_{\mathsf{RMW}}\rangle$-write-list.
  For $\pi \neq \tau$ and $L \in \mathcal{B}(\pi)$, observe that $L \in \mathcal{B}'(\pi)$, and since $G.\mathsf{hb} \subseteq G'.\mathsf{hb}$, we have that $W'_{\langle \pi, L'\rangle}$ is also a $\langle G, \pi\rangle$-consistent $\langle G, L, tid_{\mathsf{RMW}}\rangle$-write-list.
  Consider an option list $L \in \mathcal{B}(\tau)$. Let $L' \in \mathcal{B}'(\tau)$ such that $L = o \cdot L'$. Let $W' = W'_{\langle \tau, L'\rangle}$. We define $W \triangleq w \cdot W'$. By the fact that W' is a $\langle G', L', tid_{\mathsf{RMW}}\rangle$-write-list, we get that W is a $\langle G, L, tid_{\mathsf{RMW}}\rangle$-write-list. It remains to show that it is $\langle G, \tau\rangle$-consistent. Given the $\langle G', \tau\rangle$-consistency of W', for C1, we only need to show that $w \notin dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$, which is guaranteed by the properties of w as stated above (it follows from the preconditions of the READ step in opWRA). Condition C2 directly follows from the W is $\langle G', \tau\rangle$-consistency of W'. For C3, given the $\langle G', \tau\rangle$-consistency of W', it suffices to handle the case that $j = 1$. Thus, assume toward contradiction some $1 < k \le |L|$ and $1 < i < k$, such that $W(i) = \mathsf{O}_\mathsf{W}(\mathtt{loc}(W(k)))$ and $\langle W(k), w\rangle \in G.\mathsf{hb}^?$. Then, since $r \in G'.\mathsf{E}^\tau$ and $\langle w, r\rangle \in G'.\mathtt{rf}$, we get that $W'(k-1) \in dom(G'.\mathsf{hb}^? ; [\mathsf{E}^\tau])$, while $W'(i-1) = \mathsf{O}_\mathsf{W}(\mathtt{loc}(W'(k-1)))$, contradicting (C2 in) the $\langle G', \tau\rangle$-consistency of W'.
- $l = \mathsf{RMW}(x, v_\mathsf{R}, v_\mathsf{W})$:

  Let $e = \mathsf{NextEvent}(G.\mathsf{E}, \tau, l)$. Since $G \xrightarrow{\tau,l}_{\mathsf{opWRA}} G'$, we have $G'.\mathsf{E} = G.\mathsf{E} \cup \{e\}$, $G'.\mathtt{rf} = G.\mathtt{rf} \cup \{\langle w, e\rangle\}$ and $\mathtt{val}_\mathsf{W}(w) = v_\mathsf{R}$, for some $w \in \mathsf{W}_x$, such that $w \notin dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$ and $w \notin dom(G.\mathtt{rf} ; [\mathsf{RMW}])$.
  Let $\mathcal{P}$ be the index choice for $\mathcal{B}'$ that assigns the set of "new" positions in $\mathcal{B}'$:

$$\mathcal{P} \triangleq \lambda\pi \in \mathsf{Tid}, L' \in \mathcal{B}'(\pi). \{1 \le k \le |L'| \mid W'_{\langle \pi, L'\rangle}(k) = e\}.$$

  Then, we define:

$$\mathcal{B} \triangleq \lambda\pi \in \mathsf{Tid}. \begin{cases} o \cdot \mathsf{O}_\mathsf{W}(x) \cdot \mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\tau) & \pi = \tau \\ \mathsf{src}(\mathcal{B}', \tau, \mathcal{P})(\pi) & \pi \neq \tau \end{cases}$$

  where o is the read option given by $o \triangleq \mathsf{O}_\mathsf{R}(\mathtt{tid}(w), x, v_\mathsf{R}, \tau)$.
  Using Prop. 8.4, to show that $\mathcal{B} \xrightarrow{\tau,l}_{\mathsf{loWRA}} \mathcal{B}'$, it suffices to prove that $\mathcal{P} \models_{\mathsf{WRA}} \langle \tau, \mathsf{W}(x, v_\mathsf{W})\rangle$. This is done as in the write case, together with the following observation: Since $e \in G'.\mathsf{E}^\tau$, $e \in \mathsf{RMW}$ and $\langle w, e\rangle \in G'.\mathtt{rf}$, the fact that tid_{RMW} witnesses $\mathcal{B}' \vartriangleright G'$, guarantees that $tid_{\mathsf{RMW}}(w) = \tau$.
  It remains to show that $\mathcal{B} \vartriangleright G$. We show that for every $\pi \in \mathsf{Tid}$ and $L \in \mathcal{B}(\pi)$, there exists a $\langle G, \pi\rangle$-consistent $\langle G, L, tid_{\mathsf{RMW}}\rangle$-write-list. (The second condition of $\vartriangleright$ for WRA (Def. 7.4) trivially holds.) Let $\pi \in \mathsf{Tid}$ and $L \in \mathcal{B}(\pi)$. Following the construction of $\mathcal{B}$, one of the following holds:

- $\pi \neq \tau$ and $L = L' \setminus \mathcal{P}(\pi, L')$ for some $L' \in \mathcal{B}'(\pi)$. This case is exactly the same as the analogous case in the WRITE step.
- $\pi = \tau$ and $L = o \cdot \mathtt{O_W}(x) \cdot (L' \setminus \mathcal{P}(\tau, L'))$ for some $L' \in \mathcal{B}'(\tau)$. Let $P = \mathcal{P}(\tau, L')$, $W' = W'_{\langle \tau, L' \rangle}$ and $f = \lambda k \in \{3, \ldots, |L|\}$. $\mathrm{Map}^{-1}_{\langle L', P \rangle}(k-2)$. We define:

$$W \triangleq \lambda k \in \{1, \ldots, |L|\}. \begin{cases} w & k = 1 \\ \mathtt{O_W}(x) & k = 2 \\ W'(f(k)) & k > 2 \end{cases}$$

By the fact that W' is a $\langle G', L', tid_{\mathtt{RMW}} \rangle$-write-list, we get that W is a $\langle G, L, tid_{\mathtt{RMW}} \rangle$-write-list, and we show that it is $\langle G, \tau \rangle$-consistent:

C1) Observe first that $W(1) = w$ and $w \notin dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$ is guaranteed by the properties of w as stated above (it follows from the preconditions of the RMW step in opWRA). Now, consider some $2 < k \leq |L|$. By the $\langle G', \tau \rangle$-consistency of W', we have $W(k) \notin dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau \cup \{W(j) \mid 3 \leq j < k\}])$. It is left to show that $\langle W(k), w \rangle \notin G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^?$. Indeed, were it not the case, since $\langle w, e \rangle \in G'.\mathtt{rf}$ and $e \in \mathsf{E}^\tau$, we would have had $W'(f(k)) \in dom(G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$, contradicting (C1 in) the $\langle G', \tau \rangle$-consistency of W'.

C2) Due to adding $W(2) = \mathtt{O_W}(x)$, which is not present in W', we should ensure that $W(k) \notin dom(G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$ for every $2 < k \leq |L|$. Indeed, this is guaranteed by (C1 in) the $\langle G', \tau \rangle$-consistency of W', as $e = \max_{G'.\mathrm{po}} G'.\mathsf{E}^\tau$, $\mathtt{loc}(W(k)) = \mathtt{loc}(e)$, $e \in \mathsf{W}$, $W'(f(k)) = W(k)$, and $W'(f(k)) \notin dom(G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G'.\mathsf{hb}^? ; [\mathsf{E}^\tau])$.

C3) Due to adding $W(1) = w$ and $W(2) = \mathtt{O_W}(x)$, we should ensure that for every $2 < k \leq |L|$, if $\mathtt{loc}(W(k)) = x$ then $\langle W(k), w \rangle \notin G.\mathsf{hb}^?$. First observe that $W(k) \neq w$, as otherwise we would have had $W'(f(k)) \in dom(G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G'.\mathsf{hb}^? ; [\mathsf{E}^\tau])$, since $W'(f(k)) = W(k) = w$, $\langle w, e \rangle \in G'.\mathtt{rf}$ and $e \in \mathsf{W}$, which contradicts (C1 in) the $\langle G', \tau \rangle$-consistency of W'. Then, observe that $\langle W(k), w \rangle \notin G.\mathsf{hb}$, as $w \in \mathsf{W}$, and we showed while handling C1 that $\langle W(k), w \rangle \notin G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^?$.

- $\pi = \tau$ and $L = o \cdot \mathtt{O_W}(x) \cdot (L' \setminus\setminus \mathcal{P}(\eta, L'))$ for some $\eta \in \mathsf{Tid}$ and $L' \in \mathcal{B}'(\eta)$.
Let $P = \mathcal{P}(\eta, L')$, $W' = W'_{\langle \eta, L' \rangle}$, $m = \min(P)$ and $f = \lambda k \in \{3, \ldots, |L|\}$. $\mathrm{MMap}^{-1}_{\langle L', P \rangle}(k-2)$. We define:

$$W \triangleq \lambda k \in \{1, \ldots, |L|\}. \begin{cases} w & k = 1 \\ \mathtt{O_W}(x) & k = 2 \\ W'(f(k)) & k > 2 \end{cases}$$

By the fact that W' is a $\langle G', L', tid_{\mathtt{RMW}} \rangle$-write-list, we get that W is a $\langle G, L, tid_{\mathtt{RMW}} \rangle$-write-list, and we show that it is $\langle G, \tau \rangle$-consistent:

C1) The difference from the previous case is that we have the $\langle G', \tau \rangle$-consistency of $W'_{\langle \eta, L' \rangle}$ rather than of $W'_{\langle \tau, L' \rangle}$. Hence, we should show that for every $2 < k \leq |L|$, we still have $W(k) \notin dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau \cup \{w\}])$. Assume first toward contradiction some k such that $W(k) \in dom(G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^? ; [\mathsf{E}^\tau])$. Since $W'(f(k)) = W(k)$, $f(k) > m$, $W'(m) = e$ and $e = \max_{G'.\mathrm{po}} G'.\mathsf{E}^\tau$, we have $W'(f(k)) \in dom(G'.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G'.\mathsf{hb}^? ; [\{W'(j) \mid 1 \leq j < f(k)\}])$, contradicting (C1 in) the $\langle G', \eta \rangle$-consistency of W'. Next, assume toward contradiction some k, such that $\langle W(k), w \rangle \in G.\mathsf{hb}|_{\mathtt{loc}} ; [\mathsf{W}] ; G.\mathsf{hb}^?$. Then, we reach an analogous contradiction, since $\langle w, e \rangle \in G'.\mathtt{rf}$.

C2) Handled exactly as in the analogous case of the WRITE step (referring to e instead of w).

C3) Due to adding $W(1) = w$ and $W(2) = \mathtt{O_W}(x)$, we should ensure that for every $2 < k \leq |L|$, if $\mathtt{loc}(W(k)) = x$ then $\langle W(k), w \rangle \notin G.\mathsf{hb}^?$. Indeed, assume toward contradiction that

$\langle W(k), w\rangle \in G.\mathsf{hb}^?$. Then, since $\langle w, e\rangle \in G'.\mathtt{rf}$, $W'(m) = e$ and $e \in \mathrm{W}$, we get that $\langle W'(f(k)), W'(m)\rangle \in G'.\mathsf{hb}|_{\mathtt{loc}}$; [W] ; $G'.\mathsf{hb}^?$. Since $f(k) > m$, this contradicts (C1 in) the $\langle G', \eta\rangle$-consistency of W'. □