What’s Decidable about Causally Consistent Memory Models?

ORI LAHAV, Tel Aviv University, Israel
UDI BOKER, Interdisciplinary Center (IDC) Herzliya, Israel

Causal consistency is one of the most fundamental consistency models weaker than sequential consistency. In this paper, we study the verification of safety properties for finite-state concurrent programs running under a causally consistent shared-memory model. While this problem was recently shown to be undecidable for a certain form of causal consistency (the Release/Acquire fragment of the C/C++11 memory model), we show that it is actually decidable for two other well-studied variants of causal consistency (one is stronger than Release/Acquire and the other is weaker). For each of the two variants, our proof proceeds by developing a novel operational semantics that is equivalent to the existing declarative semantics and constitutes a well-structured transition system. Furthermore, since Release/Acquire consistency coincides with the other two variants for write/write-race-free programs, the decidability for such programs under Release/Acquire consistency follows. In addition, the alternative operational semantics may be of independent use in the investigation of causally consistent shared-memory models and their verification.

1 INTRODUCTION

Suppose that one wants to verify that a given sequential program satisfies a certain safety specification (e.g., that it never crashes). If the data domain is bounded, we can represent the program as a finite-state transition system, and this verification problem is trivially decidable. Moving to concurrent programs, assuming (non-realistic) sequentially consistent shared-memory semantics, does not change much—the memory constitutes another finite-state system, and its synchronization with the interleaving of the systems representing the different threads is easily expressible as a finite-state system as well. On the other hand, if the memory does not ensure sequential consistency, but rather provides weaker consistency guarantees, the decidability of the safety verification problem is completely unclear.

In this paper, we are interested in the safety verification problem under causally consistent shared memory. Causal consistency is one of the most fundamental consistency models weaker than sequential consistency, which is especially common and well studied in distributed databases (see, e.g., [Lloyd et al. 2011]). Roughly speaking, by allowing nodes to disagree on the relative order of some memory operations, and require global consensus only on the order of “causally related” operations, causal consistency allows scalable, partition tolerant and available implementations.

Nowadays, causal consistency has become central also in multithreaded programming. The Release/Acquire fragment (RA) of the C/C++11 standard [Batty et al. 2011; ISO/IEC 14882:2011 2011; ISO/IEC 9899:2011 2011] is a form of causal consistency, which specifies the guarantees C and C++ ensure for their widely used memory_order_release and memory_order_acquire synchronization accesses. Causal consistency guarantees are also provided by multiprocessor architectures, such as POWER [Alglave et al. 2014; Sarkar et al. 2011], that are not “multi-copy atomic” (that is, different threads may detect stores of another thread at different times). Specifically, as shown in [Lahav et al. 2016], a natural strengthening of RA, called SRA (for Strong Release/Acquire), precisely captures the guarantees provided by the POWER architecture for programs compiled from the C/C++’s release/acquire fragment. (Excluding read-modify-writes, SRA is, in fact, equivalent to standard causal consistency in distributed databases as defined in [Burckhardt 2014].) A natural
weaker variant of RA, called WRA (for Weak Release/Acquire), was also considered as a useful candidate for shared-memory concurrency semantics [Kokologiannakis et al. 2017; Lahav 2019].

Despite their centrality, until recently not much was known about the safety verification problem under causal consistency models. The challenge arises first since the standard semantics of causal consistency models is declarative (identifying program behaviors with partially ordered execution histories that obey certain formal consistency constraints), while verification is typically applied on operational models. Moreover, operational versions of causal consistency models are inherently infinite-state, as threads may generally read from an unbounded past. In fact, the reduction of Atig et al. [2010] from reachability in lossy FIFO channel machines to safety verification under x86-TSO semantics can be straightforwardly adapted to causally consistent models (specifically, RA, SRA and WRA). This implies a non-primitive recursive lower bound on the safety verification problem under causal consistency. In fact, very recently, Abdulla et al. [2019] proved that under RA the safety verification problem is undecidable.

Our main contribution in this paper is to establish the decidability of safety verification under both the SRA and the WRA causal consistency models. 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, since RA, SRA and WRA coincide on write/write-race-free programs, we obtain the decidability of safety verification under RA for this large and widely used class of programs.

To obtain decidability, we develop novel operational semantics for programs running under SRA and WRA that is equivalent to their declarative semantics. The semantics are infinite-state, but as we show, can be utilized to decide the safety verification problem by using the framework of well-structured transition systems [Abdulla 2010; Abdulla et al. 2000; Finkel and Schnoebelen 2001].

The key idea in these semantics is to maintain the potential of future operations of each thread in the machine state. Under SRA, it suffices for this potential to consist of ordered lists of optional future reads, while WRA also requires to delimit the potential reads with future writes. Thus, read transitions are very simple, they only consume a prefix of the potential, and the complexity is left for write transitions that need to properly increase the potentials of the different threads in a way that ensures causal consistency. Our fundamental observation is that the update of the potential of a certain thread when another thread executes a write instruction can be defined solely in terms of the existing potentials of the two threads. Centered about potential, the two semantics we develop can be made “lossy”, making them suitable well-structured transition systems. Indeed, losing some parts of the possible potential never allows for additional behaviors. This intuition is made precise in our correspondence proofs, which establish simulations (forward for one direction and backward for the converse) between these lossy semantics and the straightforward “operationalizations” of SRA’s and WRA’s declarative semantics.

Related Work
Different causally consistent shared-memory models, their verification problems and approaches to address these problems were recently outlined in [Lahav 2019], where the problems we resolve were left open. As mentioned above, Abdulla et al. [2019] proved the undecidability of safety verification under RA. Operational “message-passing” semantics for SRA was developed in [Lahav et al. 2016], but it is inadequate for our purposes since making it “lossy” would affect its allowed outcomes.

The safety verification problem was previously investigated under TSO—the “total store ordering” model of x86 multiprocessors, which, being multi-copy-atomic, is stronger than any of the models studied here. Atig et al. [2010, 2012] establish the decidability of this problem (and the 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 cannot be applied for the models studied here. Notably, causal consistency models cannot be fully explained by program transformations (instruction reordering and merging) [Lahav and Vafeiadis 2016], whereas, with the exception of the recent undecidability in [Abdulla et al. 2019], all existing results (of [Atig et al. 2012] in particular) are for models that are fully accounted for by such transformations.

More recently, Abdulla et al. [2018a] greatly simplified 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.

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 [Doherty et al. 2019; Kaiser et al. 2017; Lahav and Vafeiadis 2015; Turon et al. 2014; Vafeiadis and Narayana 2013], (bounded) model checking [Abdulla et al. 2019, 2018b; Kokologiannakis et al. 2017; Lesani et al. 2016] and robustness verification [Brutschy et al. 2018; Lahav and Margalit 2019; Nagar and Jagannathan 2018]. The latter approach 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 [Bouajjani et al. 2017]. It is, however, completely independent from verification of client programs assuming causal consistency, as we study here.

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 straightforward operational versions of SRA’s and WRA’s declarative semantics. In §5 and §6 we introduce our novel operational semantics of SRA and WRA, and show how they are used to decide the safety verification problem. We conclude in §7. Appendices A to E provide full proofs.

2 THE SAFETY VERIFICATION PROBLEM 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 (§2.1), interpret its programs as transition systems (§2.2), and present the generic framework of declarative execution graphs (§2.3).

2.1 Programming Language

Let \( \text{Val} \subset \mathbb{N} \), \( \text{Loc} \subset \{ x, y, \ldots \} \), \( \text{Reg} \subset \{ a, b, \ldots \} \) be finite sets of values, (shared) locations, and register names. 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); if \( e \) \( \text{goto} \) \( n \) 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 \( r \); \( r := \text{FADD}(x, e) \) atomically increments \( x \) by the value of \( e \) and loads the old value of \( x \) to \( r \); \( r := \text{XCHG}(x, e) \) atomically swaps \( x \) to the value of \( e \) and loads the old value of \( x \) to \( r \); and
\( v \in \text{Val} \subseteq \mathbb{N} \)
\( x, y, z \in \text{Loc} \subseteq \{x, y, \ldots\} \)
\( r \in \text{Reg} \subseteq \{a, b, \ldots\} \)
\( \tau, \pi, \eta \in \text{Tid} \subseteq \{T_1, T_2, \ldots\} \)
\( S \in \text{SProg} \triangleq \{0, 1, \ldots, N\} \rightarrow \text{Inst} \)
\( P : \text{Tid} \rightarrow \text{SProg} \)

Values
Locations
Registers
Thread identifiers
Concurrent programs

Definition 2.1. A label \( l \) is either \( R(x, v_r) \) (read label), \( W(x, v_w) \) (write label) or \( \text{RMW}(x, v_{rR}, v_{w}) \) (RMW label), where \( x \in \text{Loc} \) and \( v_r, v_w \in \text{Val} \). We denote by \( \text{Lab} \) the set of all labels. The functions \( \text{typ}, \text{loc}, \text{val}_{R}, \text{val}_{W} \) return (when applicable) the type (\( R/W/RMW \)), location, read value and written value of a given label.

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

Fig. 2. Transitions of LTS induced by a sequential program \( S \in \text{SProg} \).

\( r := \text{CAS}(x, e_R, e_w) \) 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 set of the form \( \{0, 1, \ldots, N\} \) (the possible values of the program counter) to instructions. We denote by \( \text{SProg} \) the set of all sequential programs. A concurrent program \( P \) is a top-level parallel composition of sequential programs, defined as a mapping from a finite set \( \text{Tid} \subseteq \{T_1, T_2, \ldots\} \) of thread identifiers to \( \text{SProg} \). In our examples, we often write sequential programs as sequences of instructions delimited by line breaks, use ‘\( \equiv \)’ for parallel composition, and refer to the program threads as \( T_1, T_2, \ldots \) following their left-to-right order in the program listing (see, e.g., Example 3.5 on Page 8).

2.2 From Programs to Labeled Transition Systems

Sequential and concurrent programs induce labeled transition systems.

Preliminaries on labeled transition systems. A labeled transition system (LTS) \( A \) over an alphabet \( \Sigma \) is a triple \( (Q, Q_0, T) \), 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 \( A.\mathcal{Q} \), \( A.\mathcal{Q}_0 \) and \( A.T \) the three components of an LTS \( A \); write \( \overset{a}{\rightarrow}_A \) for the relation \( \{(q, q') \mid (q, \sigma, q') \in A.T\} \) and \( \rightarrow_A \) for \( \bigcup_{\sigma \in \Sigma} \overset{\sigma}{\rightarrow}_A \). A state \( q \in A.\mathcal{Q} \) is reachable in \( A \) if \( q_0 \overset{\sigma_1}{\rightarrow}_A \cdots \overset{\sigma_n}{\rightarrow}_A q \) for some \( q_0 \in A.\mathcal{Q}_0 \). A sequence \( \sigma_1, \ldots, \sigma_n \) is a trace of \( A \) if \( q_0 \overset{\sigma_1}{\rightarrow}_A \cdots \overset{\sigma_n}{\rightarrow}_A q \) for some \( q_0 \in A.\mathcal{Q}_0 \) and \( q \in A.\mathcal{Q} \). The set of predecessors of a set \( S \subseteq A.\mathcal{Q} \) w.r.t. a symbol \( \sigma \in \Sigma \), denoted by \( \text{pred}_{\sigma}(S) \), is given by \( \{q \in A.\mathcal{Q} \mid \exists q' \in S. q \overset{\sigma}{\rightarrow}_A q'\} \). In addition, the set of predecessors of a set \( S \subseteq A.\mathcal{Q} \), denoted by \( \text{pred}_A(S) \), is given by \( \bigcup_{\sigma \in \Sigma} \text{pred}_{\sigma}(S) \).

For sequential programs the alphabet is the set of \( \text{labels} \) (extended with \( \epsilon \) for silent transitions), as defined next.

Definition 2.1. A label \( l \) is either \( R(x, v_r) \) (read label), \( W(x, v_w) \) (write label) or \( \text{RMW}(x, v_{rR}, v_{w}) \) (RMW label), where \( x \in \text{Loc} \) and \( v_r, v_w \in \text{Val} \). We denote by \( \text{Lab} \) the set of all labels. The functions \( \text{typ}, \text{loc}, \text{val}_{R}, \text{val}_{W} \) return (when applicable) the type (\( R/W/RMW \)), location, read value and written value of a given label.
A sequential program \( S \in \text{SProg} \) induces an LTS over \( \text{Lab} \cup \{ \varepsilon \} \), whose states are pairs \( s = \langle \text{pc}, \phi \rangle \) where \( \text{pc} \in \mathbb{N} \) (called program counter) and \( \phi : \text{Reg} \to \text{Val} \) (called local store, and extended to expressions in the obvious way). Its only initial state is \( \langle 0, \lambda r \in \text{Reg}, 0 \rangle \), and its transitions are given in Fig. 2, following the informal description above. (In particular, a read instruction in \( S \) induces \( |\text{Val}| \) transitions with different read labels.) We identify sequential programs with their induced LTSs (when writing, e.g., \( S.Q \) and \( \rightarrow_S \)).

In turn, a concurrent program \( P \) is identified with an LTS over \( \text{Tid} \times (\text{Lab} \cup \{ \varepsilon \}) \). Its states are functions, often denoted by \( \bar{p} \), assigning a state in \( P(\tau).Q \) to every \( \tau \in \text{Tid} \); its initial states set is \( \{ \bar{p} \mid \forall \tau \in \text{Tid}, \bar{p}(\tau) \in P(\tau).Q_0 \} \); and its transitions are “interleaved transitions” of \( P \)’s components, formally given by:

\[
\text{lab} \in \text{Lab} \cup \{ \varepsilon \} \quad \bar{p}(\tau) \xrightarrow{\text{lab}} \bar{p}[\tau \mapsto s']
\]

2.3 From Labeled Transition Systems to Execution Graphs

We present the general notions used to assign declarative semantics to concurrent programs. First, we define execution graphs, starting with their nodes, called events.

Definition 2.2. An event is a triple \( e = \langle \tau, s, l \rangle \), where \( \tau \in \text{Tid} \) is a thread identifier, \( s \in \mathbb{N} \) is a serial number inside each thread and \( l \in \text{Lab} \) is a label (as defined in Def. 2.1). The function \( \text{tid} \) returns the thread identifier of an event. The functions \( \text{typ}, \text{loc}, \text{val}_l, \text{val}_w \) are lifted to events in the obvious way. We denote by \( E \) the set of all events, and use \( R, W, \text{RMW} \) for its subsets:

\[
R \triangleq \{ e \mid \text{typ}(e) \in \{ R, \text{RMW} \} \} \quad W \triangleq \{ e \mid \text{typ}(e) \in \{ W, \text{RMW} \} \} \quad \text{RMW} \triangleq \{ e \mid \text{typ}(e) = \text{RMW} \}
\]

We employ subscripts and superscripts to restrict sets of events to certain location and thread identifier (e.g., \( W_x = \{ w \in W \mid \text{loc}(w) = x \} \) and \( E^\tau = \{ e \in E \mid \text{tid}(e) = \tau \} \)).

Our representation of events induces a sequenced-before partial order on events, in which 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 that totally orders the writes to each location.

Definition 2.3. Let \( E \) be a set of events.

- A relation \( \text{rf} \subseteq E \times E \) is a reads-from relation for \( E \) if the following hold:
  - If \( \langle w, r \rangle \in \text{rf} \), then \( w \in W, r \in R, \text{loc}(w) = \text{loc}(r) \) and \( \text{val}_l(w) = \text{val}_l(r) \).
  - If \( \langle w_1, r \rangle, \langle w_2, r \rangle \in \text{rf} \), then \( w_1 = w_2 \) (that is, \( \text{rf}^{-1} = \{ \langle r, w \rangle \mid \langle w, r \rangle \in \text{rf} \} \) is functional).
  - \( \forall r \in E \cap R. \exists w. \langle w, r \rangle \in \text{rf} \) (each read event reads from some write).

- A relation \( \text{mo} \subseteq E \times E \) is a modification order for \( E \) if it is a disjoint union of relations \( \{ \text{mo}_x \}_{x \in \text{Loc}} \) such that each \( \text{mo}_x \) is a strict total order on \( E \cap W_x \).

Definition 2.4. An execution graph is a triple \( G = \langle E, \text{rf}, \text{mo} \rangle \) where \( E \) is a finite set of events, \( \text{rf} \) is a reads-from relation for \( E \) and \( \text{mo} \) is a modification order for \( E \). We denote by \( \text{EGraph} \) the set of all execution graphs. The components of \( G \) are denoted by \( G.E, G.\text{rf} \) and \( G.\text{mo} \), and \( G.\text{po} \) denotes the restriction of sequenced-before to \( G.E \) (i.e., \( G.\text{po} \triangleq \{ \langle e_1, e_2 \rangle \in G.E \times G.E \mid e_1 < e_2 \} \)). For a set \( E \subseteq E \), we write \( G.E \) for \( G.E \cap E \) (e.g., \( G.W = G.E \cap W \) and \( G.E^\tau = G.E \cap E^\tau \)).

The next definition is used to associate execution graphs to concurrent programs.

Notation 2.5. For a set \( E \) of events, thread identifier \( \tau \in \text{Tid} \) and label \( l \in L \), NextEvent\( (E, \tau, l) \) denotes the event given by \( \langle \tau, 1 + \max(\{ n \in \mathbb{N} \mid \exists l' \in \text{Lab}. \langle \tau, n, l' \rangle \in E \} \), l \rangle \).
Definition 2.6. Let \( P \) be a concurrent program and let \( \bar{P} \in P.Q \). An execution graph \( G \) is generated by \( P \) with final state \( \bar{p} \) if \( \langle \bar{p}, G_0 \rangle \rightarrow^* \langle \bar{p}, G \rangle \) for some \( \bar{p}_0 \in P.Q_0 \), where \( G_0 \) denotes the empty execution graph (given by \( G_0 \triangleq \langle \emptyset, \emptyset, \emptyset \rangle \)) and \( \rightarrow \) is defined by:

\[
\bar{p} \xrightarrow{r,f} \bar{p}' \quad E' = E \cup \{ \text{NextEvent}(E, r, l) \} \quad \text{rf} \subseteq \text{rf}' \quad \text{mo} \subseteq \text{mo}' \quad \bar{p} \xrightarrow{r,f} \bar{p}' \quad \langle \bar{p}, (E, \text{rf}, \text{mo}) \rangle \rightarrow \langle \bar{p}', (E', \text{rf}', \text{mo}') \rangle \quad \langle \bar{p}, G \rangle \rightarrow \langle \bar{p}', G \rangle
\]

We say that \( G \) is generated by \( P \) if it is generated by \( P \) with some final state.

Multiple examples below (e.g., Examples 3.5 and 3.7 on Pages 8 and 9) illustrate execution graphs generated by concurrent programs.

In turn, a declarative model \( X \) is a set of execution graphs, where we refer to the elements of \( X \) as \( X \)-consistent. The reachability problem under a declarative model is defined as follows.

Definition 2.7. A state \( \bar{p} \) 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 \( \bar{p} \).

Definition 2.8. The reachability problem under a declarative model \( X \) is given by:

**Input:** a concurrent program \( P \) and a “bad state” \( \bar{p} \in P.Q \).

**Question:** is \( \bar{p} \) reachable under \( X \)?

## 3 DECLARATIVE CAUSALLY CONSISTENT MEMORY MODELS

In this section, we formulate the three declarative models: Weak Release/Acquire (WRA), Release/Acquire (RA), and Strong Release/Acquire (SRA). Our presentation follows [Lahav 2019]. Figure 3 illustrates the different consistency constraints described below.

Notation 3.1 (Relations). Given a relation \( R \), \( \text{dom}(R) \) denotes its domain; \( R^\prime \) 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 \( [A] ; R ; [B] = R \cap (A \times B) \).

All causal consistency models are based on the following basic derived “happens-before” relation:

\[
G.\text{hb} \triangleq (G.\text{po} \cup G.\text{rf})^+
\]

The happens-before relation captures the “causality relation” in execution graphs. In words, \( \text{hb} \) is the smallest transitive relation that contains the program order (po) and the reads-from (rf) relations. We note that all reads synchronize with the writes they read from (rf \( \subseteq \) hb), in contrast to more elaborate models like RC11 [Lahav et al. 2017], 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:

\( G.\text{hb} \) is irreflexive  \hspace{1cm} (irr-hb)

In particular, this condition forbids so-called “load-buffering” behaviors [Maranget et al. 2012], which are allowed in weaker models (and unless restricted appropriately lead to the infamous “out-of-thin-air” problem [Batty et al. 2015]).

The next condition requires that the modification order \( \text{mo} \) “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:

\[
G.\text{mo} ; G.\text{hb} \text{ is irreflexive} \hspace{1cm} \text{(write-coherence)}
\]

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

\[ (G.\text{hb} \cup G.\text{mo})^+ \text{ is irreflexive} \quad \text{(irr-hb-mo)} \]

Note that \((\text{hb} \cup \text{mo})\)-cycles involving only one location are already disallowed by \text{write-coherence} (using the fact that \text{mo} is total on writes to the same location). But, \text{irr-hb-mo} imposes constraints on the relation between \([W_x]; \text{mo}; [W_y] \) and \([W_y]; \text{mo}; [W_y] \) also for \(x \neq y\) (see the \(2+2\)W program in Example 3.8 below). In turn, in WRA, \text{mo} plays no role, and imposing either of these conditions (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 \(r\) being aware of some write event \(w\) with \text{hb} from \(w\) to (some event of) \(r\). They do, however, differ in their interpretation of one write being “later” than another. RA and SRA employ the modification order \text{mo} for this purpose precisely and require:

\[ G.\text{mo} ; G.\text{hb} ; G.\text{rf}^{-1} \text{ is irreflexive} \quad \text{(read-coherence)} \]

Indeed, if a read event \(r\) reads from a write event \(w_1\), while being aware of an \text{mo}-later write event \(w_2\) to the same location, we have \(\langle w_1, w_2 \rangle \in \text{mo}, \langle w_2, r \rangle \in \text{hb} \) and \(\langle r, w_1 \rangle \in \text{rf}^{-1}\). WRA imposes a weaker condition by using \text{hb} to order writes. To state its formal condition, it is convenient to use a per-location restriction of the happens-before relation:

\[ G.\text{hb}\rvert_{\text{loc}} \triangleq \{ (e_1, e_2) \in G.\text{hb} \mid \text{loc}(e_1) = \text{loc}(e_2) \} \]

Using \((\text{hb})\rvert_{\text{loc}}\), the condition of WRA is given by:

\[ G.\text{hb}\rvert_{\text{loc}} ; [W] ; G.\text{hb} ; G.\text{rf}^{-1} \text{ is irreflexive} \quad \text{(weak-coherence)} \]

Again, if a read event \(r\) reads from a write event \(w_1\), while being aware of an \text{hb}-later write event \(w_2\) to the same location, we have \(\langle w_1, w_2 \rangle \in \text{hb}\rvert_{\text{loc}} ; [W], \langle w_2, r \rangle \in \text{hb} \) and \(\langle r, w_1 \rangle \in \text{rf}^{-1}\). Note that \text{write-coherence} implies that \([W] ; \text{hb}\rvert_{\text{loc}} ; [W] \subseteq \text{mo}\), and so \text{weak-coherence} 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 \text{mo}-predecessors:

\[ G.\text{mo} ; G.\text{mo} ; G.\text{rf}^{-1} \text{ is irreflexive} \quad \text{(atomicity)} \]

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

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

(That is, \(G.\text{rf} ; [\text{RMW}]\) 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 Example 3.9). To see that atomicity implies weak-atomicity (in the presence of \text{write-coherence} or \text{irr-hb-mo}), assume a violation of weak-atomicity, and note that since \text{mo} must order the two RMWs and \text{write-coherence} (or \text{irr-hb-mo}) dictates that \text{mo} ; \text{rf} is irreflexive, it entails a violation of atomicity.

Figure 3 illustrates the different constrains. The next table lists the constraints of each model.

<table>
<thead>
<tr>
<th>Model</th>
<th>\text{irr-hb}</th>
<th>\text{weak-coherence}</th>
<th>\text{weak-atomicity}</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA</td>
<td>\text{irr-hb}</td>
<td>\text{write-coherence}</td>
<td>\text{read-coherence}</td>
</tr>
<tr>
<td>SRA</td>
<td>\text{irr-hb-mo}</td>
<td>\text{read-coherence}</td>
<td>\text{atomicity}</td>
</tr>
</tbody>
</table>
Recall that write-coherence and read-coherence together imply weak-coherence; write-coherence and atomicity together imply weak-atomicity; and irr-hb-mo implies both irr-hb and write-coherence. Therefore, the following proposition holds.

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

For write/write-race free programs the three models coincide. Inspired by the DRF models of [Adve and Hill 1990], we show that write/write-race freedom of SRA-consistent execution graphs suffices, so that users of this theorem can adhere to a safe programming discipline without even understanding the two weaker models, WRA and RA. The proof is given in Appendix D.

**Definition 3.3.** An execution graph \( G \) is write/write-race free if for every \( w_1, w_2 \in G.W \) with \( 1\mathsf{oc}(w_1) = 1\mathsf{oc}(w_2) \), we have \( w_1 = w_2, \langle w_1, w_2 \rangle \in G.hb \) or \( \langle w_2, w_1 \rangle \in G.hb \).

**Theorem 3.4.** 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.

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 (where initialization, which is not implicitly included in our execution graphs, has been made explicit). To simplify the presentation, instead of referring to reachable program states, we consider possible program outcomes assigning final values to (some) registers. An outcome \( O \) is allowed for a program under a declarative model \( X \) if some state in which the registers are assigned their values in \( O \) is reachable under \( X \) (see Def. 2.7).

We use program comment annotations (“//”) to denote particular outcomes.

**Example 3.5 (Store buffering).** The following program outcome is allowed by all three causal consistency models. The justifying execution graph is presented on the right.

\[
\begin{align*}
x &:= 0 & y &:= 0 \\
x &:= 1 & y &:= 1 \\
a &:= y & b &:= x
\end{align*}
\]

It can be easily verified that the execution graph is SRA-consistent, and thus it is also RA-consistent and WRA-consistent.

**Example 3.6 (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 \text{rf}-edges as depicted on the right. However, we have \text{hb}\_loc from W(x, 0) to W(x, 1), \text{hb} from W(x, 1) to R(x, 0) and \text{rf} from W(x, 0) to R(x, 0). Hence, \textit{weak-coherence} does not hold, and the execution graph is not WRA-consistent.

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

\[
\begin{align*}
  x &:= 0 \quad | \quad a := y \parallel 1 \\
  y &:= 1 \quad | \quad b := x \parallel 0 \\
  &\text{WRA} \quad \text{RA} \quad \text{SRA}
\end{align*}
\]

\text{(MP-trans)}

\text{Example 3.7 (Independent reads of independent writes).} A main difference between the causal consistency models and the x86-TSO model [Owens et al. 2009] is that the former are non-multi-copy-atomic. Namely, different threads may observe different stores in different orders. Thus, unlike x86-TSO, the three 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:

\[
\begin{align*}
  x &:= 0 \quad | \quad a := x \parallel 1 \\
  y &:= 1 \quad | \quad b := y \parallel 0 \\
  &\checkmark \text{WRA} \quad \text{RA} \quad \text{SRA}
\end{align*}
\]

\text{(IRIW)}

\text{Example 3.8.} The following examples demonstrate other aspects of the different models. In particular, the two examples on the left demonstrate that, unlike RA and SRA, WRA does not provide “sequential-consistency-per-location”—even programs with a single location may exhibit non-sequentially-consistent behaviors. The (2+2W) example is taken from [Wickerson et al. 2017].

\text{(WW)}

\[
\begin{align*}
  x &:= 1 \quad | \quad a := x \parallel 2 \\
  b &:= x \parallel 1 \\
  &\checkmark \text{WRA} \quad \text{RA} \quad \text{SRA}
\end{align*}
\]

\text{(Oscillating)}

\[
\begin{align*}
  x &:= 1 \quad | \quad a := x \parallel 1 \\
  b &:= y \parallel 0 \\
  &\checkmark \text{WRA} \quad \text{RA} \quad \text{SRA}
\end{align*}
\]

\text{(2+2W)}

\[
\begin{align*}
  x &:= 1 \quad | \quad y := 1 \\
  &\checkmark \text{WRA} \quad \text{RA} \quad \text{SRA}
\end{align*}
\]

\text{(2RMW)}

\[
\begin{align*}
  x &:= 0 \quad | \quad a := x \parallel 0 \\
  b &:= x \parallel 0 \\
  &\checkmark \text{WRA} \quad \text{RA} \quad \text{SRA}
\end{align*}
\]

\text{(SB+RMWs)}
Remark 1. Our presentation follows C/++11’s mathematical formalization [Batty et al. 2011; Lahav et al. 2017], where the RA model above is the fragment of the C/++11 model consisting of release stores, acquire reads and acquire-release RMWs. In turn, the SRA model is a strengthening of RA proposed in [Lahav et al. 2016], whereas WRA is a natural weakening of RA which is sufficiently strong for Thm. 3.4 to hold. In addition, WRA and SRA appear (in multiple disguises) in the literature, especially as correctness criteria for distributed databases and data structures:

1. WRA (without RMWs) is equivalent to a basic causal consistency model called CC in [Bouajjani et al. 2017], when CC is applied to the standard read/write memory sequential specification and differentiated histories (no value is written twice). For differentiated histories, CC requires that for every read event $r$ in $G$, the restriction of $hb$ on the prefix $dom(hb; \{(r)\})$ can be extended to a total order in which the value of the last write to $loc(r)$ is $val_{\delta}(r)$. This condition is equivalent to the constraints of WRA.

2. As proved in [Lahav et al. 2016], SRA precisely coincides with the POWER model of [Alglave et al. 2014], when the latter is restricted to programs that result from compiling C/++11 programs in the release/acquire fragment, using the standard compilation scheme [Mapping 2019] (that is, placing $lwsync$ before every store and $ctrl+isync$ after every load).

3. SRA (without RMWs) is equivalent to the causal convergence model, denoted by CCv, of [Bouajjani et al. 2017] (when applied to the standard read/write memory sequential specification), as well as to the causal consistency model of [Lloyd et al. 2011] when restricted to single-instruction transactions. These models are formulated in [Burckhardt 2014; Cerone et al. 2017] in terms of visibility ($vis$) and arbitration ($ar$) relations. One direction of the correspondence follows by setting $vis = hb$ and taking $ar$ to be a total order extending $hb \cup mo$. For the converse, one takes $rf$ to relate each read $r$ with the $ar$-maximal write to the same location that is $vis$-before $r$, and sets $mo = \bigcup_{x \in Loc}[W_x]; ar; [W_x]$. Furthermore, our program order ($po$) corresponds to session order ($so$), and SRA’s consistency ensures strong session guarantees ($so \subseteq vis$) [Terry et al. 1994]. Intended for distributed systems, these models do not provide RMWs. Nevertheless, when a 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 [Li et al. 2012] (see also [Bernardi and Gotsman 2016]).

4. SRA without write events (implementing stores $x := e$ as $r := XCHG(x, e)$) precisely captures the parallel snapshot isolation model (PSI) [Ardekani et al. 2013; Bernardi and Gotsman 2016; Cerone et al. 2015; Sovran et al. 2011] when restricted to single-instruction transactions.

4 OPERATIONAL CAUSALLY CONSISTENT MEMORY MODELS

While the above formulation of the casual consistency models is declarative, it is straightforward to “operationalize” these definitions. Indeed, for the models above, instead of first generating a program execution graph and then checking certain consistency conditions, one may impose consistency at each step of an incremental construction of the execution graph. This results in equivalent operational presentations, which are arguably simpler and easier to relate to the alternative semantics we define below. In this section, we present such operationalized versions of the declarative semantics above, formulating them as memory systems.

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

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

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 ⊥ for uninitialized locations). Formally, it is defined by SC.Q ≜ Loc → (Val ∪ {⊥}), SC.Q₀ ≜ λx ∈ Loc. ⊥ and →SC is given by:

\[
\begin{align*}
\text{WRITE} & : \mu' = \mu[x \mapsto v_w] \quad \text{READ} & : \mu(x) = v_R \quad \text{RMW} & : \mu(x) = v_R, \mu' = \mu[x \mapsto v_w]
\end{align*}
\]

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

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

**Definition 4.3.** A concurrent system is a pair, denoted by \(P_M\), where \(P\) is a concurrent program and \(M\) is a memory system. A concurrent system \(P_M\) induces an LTS over the alphabet \((\text{Tid} \times (\text{Lab} \cup \{\varepsilon\})) \cup \{\varepsilon\}\) whose set of states is \(P.Q \times M.Q\); its initial states set is \(P.Q_0 \times M.Q_0\); and its transitions are “synchronized transitions” of \(P\) and \(M\), formally given by:

\[
\begin{align*}
\forall l \in \text{Lab} \quad \langle \overline{p}, m \rangle & \xrightarrow{\tau,l}_{P \parallel M} \langle \overline{p}', m' \rangle \\
\forall m \in \text{Lab} \quad \langle \overline{p}, m \rangle & \xrightarrow{\tau,e}_{P \parallel M} \langle \overline{p}', m' \rangle \\
\forall m \in \text{Lab} \quad \langle \overline{p}, m \rangle & \xrightarrow{\varepsilon}_{P \parallel M} \langle \overline{p}', m' \rangle
\end{align*}
\]

In the sequel we identify concurrent systems with their induced LTS.

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 \(P_M\) for some \(m \in M.Q\).

**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. 2.7) coincides with the set of program states that are reachable under \(M\).

Next, we present the memory systems opWRA and opSRA that characterize the models WRA and SRA.\(^1\) The states of these systems are execution graphs capturing (partially ordered) histories of executed actions, and the only initial state is \(G_0\) (recall that \(G_0\) denotes the empty execution graph \(\langle \emptyset, \emptyset, \emptyset \rangle\)). Formally, \(M.Q \triangleq \text{EGraph} \text{ and } (M.Q_0 \triangleq \{G_0\}) \text{ for } M \in \{\text{opWRA}, \text{opSRA}\}. \) Before providing the transitions, we refer the reader to Fig. 4 on Page 15, which illustrates a run of opSRA (or opWRA) for the SB example.

**Remark 2.** Following [Kokologiannakis et al. 2017], 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. It is possible to take a step further and develop an equivalent semantics with more compact 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 [Doherty et al. 2019]). 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 [Kaiser et al. 2017] for RA.

\(^1\)For completeness, a memory system opRA that characterizes RA is presented in Appendix E, but it is not used in the sequel.
Weak Release/Acquire. The transitions of opWRA are given by:

<table>
<thead>
<tr>
<th>WRITE</th>
<th>READ</th>
<th>RMW</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e = \text{NextEvent}(G,E,r,W(x,v_R)) )</td>
<td>( e = \text{NextEvent}(G,E,r,R(x,v_R)) )</td>
<td>( e = \text{NextEvent}(G,E,r,\text{RMW}(x,v_R,v_W)) )</td>
</tr>
<tr>
<td>( G',E = G.E \cup {e} )</td>
<td>( G',E = G.E \cup {e} )</td>
<td>( G',E = G.E \cup {e} )</td>
</tr>
<tr>
<td>( G',rf = G.rf )</td>
<td>( w \in G.W_x )</td>
<td>( w \in G.W_x )</td>
</tr>
<tr>
<td>( G',mo = G.mo )</td>
<td>( \text{val}_w(w) = v_R )</td>
<td>( \text{val}_w(w) = v_R )</td>
</tr>
</tbody>
</table>

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 \( (w \in G.W_x) \), writing the value being read \( (\text{val}_w(w) = v) \), and the thread executing the read must not be aware of an \( \text{hb} \)-later write to the same location \( (w \notin \text{dom}(G,\text{hb})_{10c}; [W]; G.\text{hb}^2; [E^f]) \). 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 \( (w \notin \text{dom}(G.rf; [RMW])) \).

**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 \( G,\text{hb} \). 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 \( (P,G) \) is reachable in opWRA, then \( G \) is a WRA-consistent execution graph that is generated by \( P \) with final state \( \overline{p} \). \( \square \)

Remark 3. Instead of requiring \( w \notin \text{dom}(G.rf; [RMW]) \) in the RMW step, we may equivalently require that \( \{e \in \text{RMW} \mid (w,e) \in G.rf\} \subseteq E^f \) (namely, if \( w \) is read by an RMW event, then that RMW event is in thread \( r \)). Indeed, \( w \notin \text{dom}(G.rf; [RMW]) \) trivially implies this condition. Conversely, if this condition holds then since \( w \notin \text{dom}(G,\text{hb})_{10c}; [W]; G.\text{hb}^2; [E^f] \), we cannot have \( w \in \text{dom}(G.rf; [RMW]) \). While this reformulation is an unnecessary complication at this stage, it plays a key role in the alternative WRA semantics in §6.

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

<table>
<thead>
<tr>
<th>WRITE</th>
<th>READ</th>
<th>RMW</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e = \text{NextEvent}(G,E,r,W(x,v_W)) )</td>
<td>( e = \text{NextEvent}(G,E,r,R(x,v_R)) )</td>
<td>( e = \text{NextEvent}(G,E,r,\text{RMW}(x,v_R,v_W)) )</td>
</tr>
<tr>
<td>( G',E = G.E \cup {e} )</td>
<td>( G',E = G.E \cup {e} )</td>
<td>( G',E = G.E \cup {e} )</td>
</tr>
<tr>
<td>( G',rf = G.rf \cup {(w,e)} )</td>
<td>( w \in G.W_x )</td>
<td>( w \in G.W_x )</td>
</tr>
<tr>
<td>( G',mo = G.mo )</td>
<td>( \text{val}_w(w) = v_R )</td>
<td>( \text{val}_w(w) = v_R )</td>
</tr>
<tr>
<td>( G',mo = G.mo )</td>
<td>( w \notin \text{dom}(G.mo; G.\text{hb}^2; [E^f]) )</td>
<td>( w \notin \text{dom}(G.mo) )</td>
</tr>
</tbody>
</table>

A WRITE step by thread \( \tau \) adds a fresh write event \( e \) placed after all events of thread \( r \) and extends \( \text{mo} \) to order \( e \) after all existing writes to the same location. A READ step by thread \( \tau \) adds a corresponding fresh read event and justifies it with a reads-from edge. This is exactly as in opWRA, but instead of using \( G.\text{hb})_{10c} \) to capture later writes (as per weak-coherence), we now use \( \text{mo} \) (as per read-coherence). An RMW step is a combination of WRITE and READ, but it is enforced to choose the reads-from source of the freshly added RMW event to be the \text{mo}-maximal write to the relevant location in the current execution graph.
This semantics exploits the fact that \( \text{hb} \cup \text{mo} \) is acyclic in SRA-consistent execution graphs (global agreement between \( \text{mo} \) and \( \text{hb} \), as per \( \text{irr-hb-mo} \)). Hence, to generate an SRA-consistent execution graph in a run of an operational semantics, we can follow a total order extending \( \text{hb} \cup \text{mo} \), which guarantees that writes are executed following their \( \text{mo} \)-order. In turn, since RMWs should read from their immediate \( \text{mo} \)-predecessor, we require that RMWs read from the current \( \text{mo} \)-maximal write. Accordingly, the next theorem is proved exactly as for WRA (Thm. 4.6), using \( G.\text{hb} \cup G.\text{mo} \) instead of \( G.\text{hb} \) when traversing an SRA-consistent execution graph \( G \).

**Theorem 4.7.** \( \text{opSRA} \) characterizes SRA.

## 5 Making Strong Release/Acquire Lossy: The \( \text{loSRA} \) Memory System

For resolving the reachability problem under SRA, we introduce an alternative memory system, which we call \( \text{loSRA} \) (for “lossy-SRA”). In this section, we present \( \text{loSRA} \), establish its equivalence to \( \text{opSRA} \), and show how it is used to decide the reachability problem. We begin with an intuitive discussion to motivate our definitions, and later spell out the formal details.

A memory state of \( \text{loSRA} \) maintains a collection of “read-option” lists for each thread, called the potential of the thread, where each read option \( o \) contains a location \( \text{loc}(o) \), a value \( \text{val}(o) \) and two other components that are explained below. Each read-option list stands for a sequence of possible future reads of the thread, listing the writes 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 \( \text{val}(o_1) \) from location \( \text{loc}(o_1) \) and then \( \text{val}(o_2) \) from location \( \text{loc}(o_2) \). The lists in the potential do not ascribe mandatory continuations, but rather possible futures (hence, read options). In the beginning, the empty list is assigned to all threads—before any write is executed, the threads cannot perform any read. In addition, the semantics is designed so that read-option lists are “lossy,” allowing a non-deterministic step that removes arbitrary options from the lists.

The read-option lists in the potentials dictate the possible read steps threads can take: for a thread \( \tau \) to read \( v \) from \( x \), an option \( o \) with \( \text{val}(o) = v \) and \( \text{loc}(o) = x \) must be the first in each of \( \tau \)’s lists. In turn, the step consumes these options, discarding the first element from each of \( \tau \)’s lists.

A write step is more involved, encapsulating the requirements of \( \text{opSRA} \). First, since \( \text{opSRA} \) performs write events following their \( \text{mo} \)-order, when a thread writes to \( x \), it cannot later read \( x \) 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 \( \text{loc}(o) = x \) appears in its potential. Second, when a thread performs a write of \( v \) to \( x \), it should allow future reads from this write. That is, a read option \( o \) with \( \text{loc}(o) = x \) and \( \text{val}(o) = v \) may be added, possibly several times, to every list of every thread. But, where in the lists should we allow to add \( o \)? The following examples demonstrate two possible cases. We use in them the notation \( o^v_x \) for a read option with \( \text{loc}(o^v_x) = x \) and \( \text{val}(o^v_x) = v \).

**Example 5.1.** Consider the IRIW program with its (SRA-allowed) outcome in Example 3.7. 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 \( x \), a read option \( o^0_x \) 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 \( y \), a read option \( o^1_y \) should be placed before \( o^0_x \) in the lists of \( T_3 \).

**Example 5.2.** Consider the MP program with its outcome in Example 3.6. 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 \( o^0_x \) 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 \( o^1_y \) in the list of \( T_1 \) placed before \( o^0_x \). Finally, \( T_2 \) may run: read 1 from \( y \) (consuming \( o^1_y \)) and then 0 from \( x \) (consuming \( o^0_x \)).
The restriction we impose on the positions of the added read options stems from the following key observation about causally consistent memory.2

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

Indeed, if thread \( \tau \) has just performed a write \( w \), then after thread \( \pi \) reads from \( w \), it “synchronizes” with \( \tau \) and it is thus confined by the sequences of reads that \( \tau \) may perform. Hence, to allow the addition of a read option \( o \) in certain positions of a list \( L \) of some thread \( \pi \), we require a justification: the suffix of \( L \) after the first occurrence of \( o \) should be a subsequence of a read-option list of the writing thread \( \tau \). This guarantees that after \( \pi \) reads from a write \( w \) of \( \tau \), it will not be able to read something that \( \tau \) could not read at the time that it wrote \( w \). (Revisiting Example 5.2, the read option \( o^1 \) cannot be placed before \( o^0 \), because \( T_1 \) cannot have \( o^0 \) in its lists at the point of writing \( 1 \) to \( y \).

Now, since the potential of thread \( \tau \) is used both for (1) dictating future reads of \( \tau \), and (2) justifying placement of read options that are generated by \( \tau \)’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 5.3.** Consider the following program, whose annotated outcome is allowed under SRA:

\[
\begin{align*}
\text{x} &:= 0 & \text{y} &:= 0 & \text{z} &:= 0 & \text{d}_1 &:= x \# 1 & \text{e}_1 &:= y \# 1 & \text{f}_1 &:= z \# 1 \\
\text{x} &:= 1 & \text{y} &:= 1 & \text{z} &:= 1 & \text{d}_2 &:= y \# 1 & \text{e}_2 &:= z \# 1 & \text{f}_2 &:= x \# 1 \\
\text{a}_1 &:= z \# 1 & \text{b}_1 &:= x \# 1 & \text{c}_1 &:= y \# 1 & \text{d}_3 &:= z \# 0 & \text{e}_3 &:= x \# 0 & \text{f}_3 &:= y \# 0 \\
\text{a}_2 &:= y \# 0 & \text{b}_2 &:= z \# 0 & \text{c}_2 &:= x \# 0 \\
\end{align*}
\]

\((SIRIW)\)

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

**Example 5.4.** Consider the following program:

\[
\begin{align*}
\text{x} &:= 0 & \text{y} &:= 0 & \text{a} &:= z \# 1 & \text{c} &:= w \# 1 \\
\text{x} &:= 1 & \text{y} &:= 1 & \text{w} &:= 1 & \text{d} &:= y \# 0 \\
\text{z} &:= 1 & \text{z} &:= 1 & \text{b} &:= x \# 0 \\
\end{align*}
\]

\((WRA, RA, SRA)\)

The annotated outcome is disallowed under SRA (as well as the other models). Indeed, since \( T_3 \) reads \( x = 0 \) after \( z = 1 \), the read of \( z \) must read from the write of \( T_2 \). But then, \( T_4 \), after reading \( w = 1 \) (from \( T_3 \)) cannot read \( y = 0 \).

---

2A weaker observation, which only considers single reads, was essential for the soundness of OGRA—an Owicki Gries logic for RA introduced in [Lahav and Vafeiadis 2015].
However, the semantics we described so far allows this outcome as in the following snippet:

\[
\begin{align*}
\{o\} & \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \\
\{o\} & \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \\
\{o\} & \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\} \rightarrow \{o\}
\end{align*}
\]

What went wrong? The problem arises when T₃ performs the read of 1 from z. At this point it has two possible futures, \( o' \) and \( o'' \). Since read options, consisting of location and value, do not uniquely identify writes, it may read 1 from z, and remain with both \( o' \) and \( o'' \). Now, it uses one of these options to justify the position of \( o' \) in the list of T₄, and the other for its own read. However, in a single run of opSRA, when reading 1 from z, T₃ must pick which write event to read from, and after doing that, either it cannot read \( x = 0 \) or it cannot read \( y = 0 \).

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 \( \tau \) to read \( v \) from \( x \), a read option \( o \) with \( \text{val}(o) = v \) and \( \text{loc}(o) = x \) and some unique writing thread identifier must be the first in every of \( \tau \)'s read-option lists. In this example, the two \( o_1 \) options will have different thread identifiers, which forces T₃ 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, an 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 \textit{mo}-maximal write to the relevant location. To achieve this in loSRA, we include an additional field in read options, which is a binary flag that can be set to either R or RMW. Intuitively, an RMW value means that the read option is set to read from the \textit{mo}-maximal write. Accordingly, an RMW step may only consume read options marked as RMW. In addition, a \textit{write} step to \( x \)—since it always replaces the \textit{mo}-maximal write to \( x \) in the execution graph—may choose to mark any of the added read options as RMW, but it can only execute when no read option (of any thread) with location \( x \) is already marked as an RMW.

Next, we turn to the formal definitions and provide additional examples.

\textbf{Notation 5.5 (Sequences).} We use \( e \) to denote the empty sequence. The length of a sequence \( s \) is denoted by \( |s| \) (in particular \(|e| = 0\)). We often identify sequences with their underlying functions (whose domain is \{1, ..., |s|\}), and write \( s(k) \) for the symbol at position \( 1 \leq k \leq |s| \) in
s. We write $\sigma \in s$ if $\sigma$ appears in $s$, that is if $s(k) = \sigma$ for some $1 \leq k \leq |s|$. We use “$\cdot$” for the concatenation of sequences, which is lifted to concatenation of sets 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 $\sigma \cdot S$).

**Definition 5.6.** Read options, read-option lists and potentials are defined as follows:

1. A read option is a quadruple $o = (\tau, x, v, u)$, where $\tau \in \text{Tid}$, $x \in \text{Loc}$, $v \in \text{Val}$ and $u \in \{R, \text{RMW}\}$.
   
   The functions $\text{tid}$, $\text{loc}$, $\text{val}$ and $\text{rmw-flag}$ return the thread identifier ($\tau$), location ($x$), value ($v$), and RMW flag ($u$) of a given read option.

2. A read-option list $L$ is a sequence of read options.

3. A potential $B$ is a finite non-empty set of read-option lists.

We define an ordering on read-option lists, which naturally extends to potentials, and to functions assigning potentials to threads.

**Definition 5.7.** The (overloaded) relation $\sqsubseteq$ is defined by:

1. on read-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: $B \sqsubseteq B'$ if $B(\tau) \subseteq B'(\tau)$ for every $\tau \in \text{Tid}$.

The loSRA memory system is formally defined as follows.

**Definition 5.8.** loSRA is defined by: loSRA,$0$ is the set of functions $B$ assigning a potential to every $\tau \in \text{Tid}$; loSRA,$[\{\epsilon\}]$ and the transitions are as follows:

\[
\begin{align*}
\forall \pi \in \text{Tid}, L' \in B'(\pi). \exists n \geq 0, u_1, \ldots, u_n, L_0, \ldots, L_n. \\
L' = L_0 \cdot (\tau, x, v_0, u_1) \cdot L_1 \ldots (\tau, x, v_n, u_n) \cdot L_n \\
\land L_0 \ldots L_n \in B(\pi) \land L_1 \ldots L_n \in B(\tau) \\
\land (\pi = \tau \implies \forall o \in L_0 \ldots L_n. \text{loc}(o) \neq x) \\
\land \forall o \in L_0 \ldots L_n. \text{loc}(o) = x \implies \text{rmw-flag}(o) = R \\
\end{align*}
\]

\[
\begin{array}{c}
\text{WRITE} \quad \frac{\text{loc}(o) = x}{B} \xrightarrow{\tau, R(x, v_0)_{\text{loSRA}}} B' \\
\text{RMW} \quad \frac{\text{val}(o) = v_\tau}{B} \xrightarrow{\tau, \text{rmw}(x, v_\tau)_{\text{loSRA}}} B' \\
\end{array}
\]

\[
\begin{align*}
\forall \pi \in \text{Tid}, L' \in B'(\pi). \exists n \geq 0, u_1, \ldots, u_n, L_0, \ldots, L_n. \\
L' = L_0 \cdot (\tau, x, v_0, u_1) \cdot L_1 \ldots (\tau, x, v_n, u_n) \cdot L_n \\
\land L_0 \ldots L_n \in B(\pi) \land L_1 \ldots L_n \in B(\tau) \\
\land (\pi = \tau \implies \forall o \in L_0 \ldots L_n. \text{loc}(o) \neq x) \\
\land \forall o \in L_0 \ldots L_n. \text{loc}(o) = x \implies \text{rmw-flag}(o) = R \\
\end{align*}
\]

\[
\begin{array}{c}
\text{READ} \quad \frac{\text{loc}(o) = x}{B} \xrightarrow{\tau, R(x, v_\tau)_{\text{loSRA}}} B' \\
\end{array}
\]

\[
\begin{array}{c}
\text{LOWER} \quad B' \sqsubseteq B \\
\end{array}
\]

Figure 4 illustrates a run of loSRA for the SB program (Example 3.5) together with the corresponding run of opSRA.

The definition of the write step generally follows the intuitive explanation above. Every read-option list after the write transition is obtained from some previous list ($L_0 \ldots L_n \in B(\pi)$), with the addition of $n \geq 0$ read options of the current write, provided that: (1) the suffix of the existing list right after the position of the first added option is a read-option list of the writing thread ($L_1 \ldots L_n \in B(\tau)$); (2) the lists of the writing thread (which are not discarded in this transition) cannot have options to read from $x$ besides the ones that are currently added ($\pi = \tau \implies \forall o \in L_0 \ldots L_n. \text{loc}(o) \neq x$)); and (3) the original lists (which are not discarded in this transition) cannot have an RMW option for $x$ ($\forall o \in L_0 \ldots L_n. \text{loc}(o) = x \implies \text{rmw-flag}(o) = R$). 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). We also note that several RMW options can be added, but only one of them may be later fulfilled, due to condition (3).
Remark 4. The write step above insists on having a justification in the form of a complete read-option list of the writing thread \((L_1 \ldots L_n \in B(\tau))\). It suffices, however, for the suffix after the first added read option to be a subsequence of some list of the writing thread \((\{L_1 \ldots L_n\} \subseteq B(\tau))\). Indeed, this less restrictive step is derivable by combining a lower step and a write step. Note also that for \(\pi = \tau\) (adding read options in the lists of the thread that performed the write), this means that no justification is needed (since \(L_0 \ldots L_n \in B(\tau)\) implies \(\{L_1 \ldots L_n\} \subseteq B(\tau)\)).

The read step requires the first option in all lists in the executing thread’s potential the read to be the same, and consumes it from all these lists. Note that, by definition, the potential \(B'(\tau)\) is non-empty, and so the set \(B(\tau)\) as defined in the step is non-empty. When all options are consumed, \(\tau\)’s potential consists of a single empty list.

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 \(\text{RMW}\).

The lower transition allows to remove read options, as well as full read-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 5.9. Consider the 2+2W program with its (SRA-disallowed) outcome in Example 3.8. 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 \(y := 2\) by \(T_1\). Before executing this write, \(T_1\) may not have any read options of location \(y\) in its lists. Hence, a read option of the form \(\langle \pi, y, 1, u \rangle\) should be added to \(T_1\)’s potential after \(T_1\) executed \(y := 2\). This contradicts our assumption that \(y := 2\) was the last executed write.

Example 5.10. Consider the 2RMW program with its (SRA-disallowed) outcome in Example 3.8. To try to obtain this outcome in loSRA, the \(x := 0\) by \(T_1\) must add a read option \(\langle T_1, x, 0, \text{RMW} \rangle\) 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, can only proceed after the other option marked with \(\text{RMW}\) is discarded. Hence, the second RMW cannot execute, and this outcome cannot be obtained by loSRA.

Next, we prove the equivalence of loSRA and opSRA. For the formal proof, we define a relation \(\Upsilon \subseteq \text{loSRA}.Q \times \text{opSRA}.Q\), formalizing the intuitive simulation relation discussed so far between loSRA’s lists and opSRA’s execution graphs. For defining \(\Upsilon\), we first define a “write list” for an execution graph \(G\) and a read-option list \(L\), which links the read options in \(L\) to write events in \(G\).

Definition 5.11. A write list is a sequence \(W\) of write events. A write list \(W\) is a \((G, L)\)-write-list if \(|L| = |W|\) and the following hold for every \(1 \leq k \leq |W|\) with \(L(k) = \langle \tau, x, v, u \rangle\):

- \(W(k) \in G.W\).
- \(\text{tid}(W(k)) = \tau, \text{loc}(W(k)) = x\) and \(\text{val}_w(W(k)) = v\).
- if \(u = \text{RMW}\), then \(W(k) \notin \text{dom}(G.\text{mo})\).

A write list \(W\) is \((G, \tau)\)-consistent if, intuitively, the extension of \(G\) with a sequence of read events in thread \(\tau\) reading from the sequence of write events in \(W\) satisfies read-coherence. Thus, we should ensure that thread \(\tau\) is not already aware of some write that is \(\text{mo}\)-later than some write of \(W\), and furthermore, that after reading from a write \(w_1\) of \(W\), thread \(\tau\) will not become aware of some write that is \(\text{mo}\)-later than some write \(w_2\) that appears after \(w_1\) in \(W\). Formally:

Definition 5.12. A write list \(W\) is \((G, \tau)\)-consistent if for every \(1 \leq k \leq |W|\), we have that \(W(k) \notin \text{dom}(G.\text{mo}; G.\text{hb}) = [E^+ \cup \{W(j) \mid 1 \leq j < k\}]\).

Now, \(\Upsilon\) relates a loSRA state \(B\) with an execution graph \(G\) if for each of \(B\)’s read-option lists there is an appropriate write list. Formally:
This formulation "works backwards"—choosing read options to omit from the target state for the corresponding list \(\pi\) may have arbitrary additional lists in addition to the above mandatory lists.

\[
\{1\} \langle W \rangle \text{Proof Outline.}
\]

Next, we establish the equivalence of \(\text{loSRA}\) and \(\text{opSRA}\) for every \(\pi\) \in \text{Tid}. We construct \(\mathcal{B}'\) such that \(\mathcal{B}' \rhd \mathcal{G}'\) and \(\mathcal{G} \rhd \mathcal{G}'\) (as depicted on the right).

**Definition 5.13.** A state \(\mathcal{B} \in \text{loSRA, Q}\) matches an execution graph \(\mathcal{G}\), denoted by \(\mathcal{B} \rhd \mathcal{G}\), if for every \(\tau \in \text{Tid}\) and \(L \in \mathcal{B}(\tau)\), there exists a \((\mathcal{G}, \tau)\)-consistent \((\mathcal{G}, L)\)-write-list. The following alternative formulation of the write step will be 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 5.14.** An index choice for a state \(\mathcal{B}' \in \text{loSRA, Q}\) 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 \(\mathcal{B}'\) justifies a \((\tau, \mathcal{W}(x, v_w))\)-step, denoted by \(\mathcal{P} \models (\tau, \mathcal{W}(x, v_w))\), if the following hold for every \(\pi\) \in \text{Tid}\) and \(L' \in \mathcal{B}'(\pi)\):

- For every \(k \in \mathcal{P}(\pi, L')\), we have \(L'(k) = \{\langle x, v_w, R \rangle, \langle x, x, v_{w}, \text{rmw} \rangle\}\).
- For every \(k \in \{1, \ldots, |L'|\} \setminus \mathcal{P}(\pi, L')\):
  - If \(\text{loc}(L'(k)) = x\), then \(\text{rmw-flags}(L'(k)) = R\).
  - If \(k > p\) for some \(p \in \mathcal{P}(\pi, L')\) or \(\pi = \tau\), then \(\text{loc}(L'(k)) \neq x\).

Now, a predecessor of \(\mathcal{B}'\) with respect to a write step intuitively satisfies two constraints: (1) For each list \(L'\) of \(\mathcal{B}'\), there is a corresponding list \(L\) in \(\mathcal{B}\) that possibly lack some read options of the form \(\langle x, v_w, u \rangle\), corresponding to the new read options of \(\mathcal{B}'\); and (2) If a list \(L'\) is different from the corresponding list \(L\) then there is a list of \(\tau\) in \(\mathcal{B}\) that justifies this difference. Notice that \(\mathcal{B}\) may have arbitrary additional lists in addition to the above mandatory lists.

**Notation 5.15 (List Operations).** 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 \(\text{Map}_{\langle L, P \rangle}\) (formally, \(\text{Map}_{\langle L, P \rangle} \triangleq \lambda k \in \{1, \ldots, |L|\} \setminus P. k - \{j \in P \mid j < k\}\)).
- \(L \setminus P\) 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 \(L \setminus P\) is denoted by \(\text{MMap}_{\langle L, P \rangle}\) (formally, \(\text{MMap}_{\langle L, P \rangle} \triangleq \lambda k \in \{\min(P), \ldots, |L|\} \setminus P. \text{Map}_{\langle L, P \rangle}(k) \setminus \min(P) + 1\)).

**Definition 5.16.** The source of \(\mathcal{B}'\) w.r.t. a thread \(\tau\) and an index choice \(\mathcal{P}\) for \(\mathcal{B}'\), denoted by \(\text{src}(\mathcal{B}', \tau, \mathcal{P})\), is given by:

\[
\text{src}(\mathcal{B}', \tau, \mathcal{P}) \triangleq \lambda \pi \in \text{Tid}. \begin{cases} \{L' \setminus \mathcal{P}(\pi, L') \mid L' \in \mathcal{B}'(\pi)\} & \pi \neq \tau \\ \{L' \setminus \mathcal{P}(\tau, L') \mid L' \in \mathcal{B}'(\tau)\} \cup \{L' \setminus \mathcal{P}(\eta, L') \mid \mathcal{P}(\eta, L') \neq \emptyset, \eta \in \text{Tid}\} & \pi = \tau \end{cases}
\]

The following proposition follows directly from our definitions.

**Proposition 5.17.** \(\mathcal{B} \xrightarrow{\tau, \mathcal{W}(x, v_w)} \text{loSRA} \mathcal{B}'\) if and only if there exists an index choice \(\mathcal{P}\) for \(\mathcal{B}'\) such that \(\mathcal{P} \models (\tau, \mathcal{W}(x, v_w))\) and \(\text{src}(\mathcal{B}', \tau, \mathcal{P})(\pi) \subseteq \mathcal{B}(\pi)\) for every \(\pi \in \text{Tid}\).

Next, we establish the equivalence of \text{loSRA} and \text{opSRA}. Full proofs are provided in Appendix A.

**Theorem 5.18.** Every trace of \text{loSRA} is a trace of \text{opSRA}.

**Proof Outline.** We show that \(\gamma\) constitutes a forward simulation from \text{loSRA} to \text{opSRA}. Handling the lower step is easy (new write lists are obtained by restricting the ones we have).

Now, suppose that (1) \(\mathcal{B} \rhd \mathcal{G}\), witnessed by a \((\mathcal{G}, \pi)\)-consistent \((\mathcal{G}, L)\)-write-list \(\mathcal{W}(\pi, L)\) for every \(\pi\) \in \text{Tid}\) and \(L \in \mathcal{B}(\pi)\); and (2) \(\mathcal{B} \xrightarrow{\tau, l} \text{loSRA} \mathcal{B}'\). We construct \(\mathcal{G}'\) such that \(\mathcal{B}' \rhd \mathcal{G}'\) and \(\mathcal{G} \xrightarrow{\tau, l} \text{opSRA} \mathcal{G}'\) (as depicted on the right).
We describe the write and read steps (the rmw step is obtained by carefully combining them).

For a write step, \( G \) is trivially constructed by adding a new write event \( w \) to \( G \), placed last in \( \text{po} \) inside the writing thread \( \tau \), and last in \( \text{mo} \) among the writes to the same location. Then, \( G \xrightarrow{\tau,l}\text{opSRA} G' \) is trivial. To show that \( B' \not\supseteq G' \), we construct for every \( \pi \in \text{Tid} \) and \( L' \in B'(\pi) \), a \( \langle G', \pi \rangle \)-consistent \( \langle G', L' \rangle \)-write-list \( W' \) as follows:

\[
W' \triangleq \begin{cases} \lambda k. & k \in P \\ W(\text{Map}(L', P)(k)) & k < \min(P) \\ \max_{G, \text{mo}}\{W(\text{Map}(L', P)(k)), W_r(\text{MMap}(L', P)(k))\} & \text{otherwise} \\
\end{cases}
\]

where: (1) \( P \) is the set of “new” positions in \( L' \); (2) \( W \) is a \( \langle G, \pi \rangle \)-consistent \( \langle G, L' \setminus P \rangle \)-write-list; and (3) \( W_r \) is a \( \langle G, \tau \rangle \)-consistent \( \langle G, L' \setminus P \rangle \)-write-list. In words, the write list \( W' \) maps: (1) the new options—to the new write event \( w \); (2) the “old” options that appear before the first new option—as mapped in the existing write list \( W \) for the corresponding list in \( B(\pi) \); and (3) each old option that appears after the first new option—to the \( \text{mo} \)-maximal write event among (a) its mapping in the existing write list \( W \) for the corresponding list in \( B(\pi) \) and (b) its mapping in the existing write list \( W_r \) for the justifying list in \( B(\tau) \) (that is, \( L' \setminus P \)). Roughly speaking, picking the \( \text{mo} \)-maximal write in the third case ensures that \( W' \) is \( \langle G', \pi \rangle \)-consistent: In the new state, thread \( \pi \) might not be able to read from a write that it previously read from (since it “synchronized” with \( \tau \) that may be aware of a later write) and might not be able to read from the write that \( \tau \) reads from (since \( \pi \) may be already aware of a later write); but, it \textit{may} read from the later write among these two writes.

In turn, to simulate a \textit{read} \textit{step} of \textit{loSRA} in \textit{opSRA}, we need to pick a write event \( w \) from which the added read event \( r \) will read-from in \( G' \). We pick \( w \) to be the \textit{po}-minimal event among the write events that the \( W'_{(\tau, L)} \) lists associate to the first option in some \( L \in B(\tau) \). (All these options are consumed during the step, and their corresponding writes are all in the same thread as dictated by the thread identifier stored in the read options). The consistency of the \( W'_{(\tau, L)} \) lists ensures that \( w \not\in \text{dom}(G, \text{mo}; G, \text{hb} \rangle [E^r]) \), so we can make the \textit{read} \textit{step} in \textit{opSRA} when reading from \( w \). To show that \( B' \not\supseteq G' \), for every thread \( \pi \neq \tau \), we simply reuse the write list \( W'_{(\pi, L)} \) that we had for \( G' \); while for thread \( \tau \) itself, we shift its write lists by one (setting \( W'_{(\tau, L)} \triangleq \lambda k. W'_{(\tau, L)}(1 + k) \)), and use the \textit{po}-minimality of \( w \) to establish \( \langle G', \tau \rangle \)-consistency.

For the converse, we favor backward simulation over forward simulation, since \textit{loSRA} requires to “guess” the future, and without knowing the target state, we cannot construct the next step.

**Theorem 5.19.** Every trace of \textit{opSRA} is a trace of \textit{loSRA}.

**Proof Outline.** We show that \( \gamma^{-1} \) constitutes a backward simulation from \textit{opSRA} to \textit{loSRA}.

For the main proof obligation (depicted on the right), suppose that \( G \xrightarrow{\tau,l}\text{opSRA} G' \) and \( B' \not\supseteq G' \), where the latter is witnessed by a \( \langle G', \pi \rangle \)-consistent \( \langle G', L' \rangle \)-write-list \( W' \) for every \( \pi \in \text{Tid} \) and \( L' \in B'(\pi) \). We construct a state \( B \) such that \( B \xrightarrow{\tau,l}\text{loSRA} B' \) and \( B \not\supseteq G \).

Again, we describe here the \textit{write} and \textit{read} steps (the \textit{rmw} step is obtained by carefully combining them).

First, for a \textit{write} step, let \( w \) be the write event that is added in \textit{opSRA}'s transition from \( G \) to \( G' \), and let \( \mathcal{P} \) be the index choice for \( B' \) that assigns the set of “new” positions in \( B' \) (formally, \( \mathcal{P} \triangleq \lambda \pi \in \text{Tid}, L' \in B'(\pi). \{ k \mid W'_{(\pi, L')} (k) = w \} \)). We define \( B \triangleq \text{src}(B', \tau, \mathcal{P}) \), and show that \( B \xrightarrow{\tau,l}\text{loSRA} B' \) using Prop. 5.17. Now, to show that \( B \not\supseteq G \), we let \( \pi \in \text{Tid} \) and \( L \in B(\pi) \), and construct a \( \langle G, \pi \rangle \)-consistent \( \langle G, L \rangle \)-write-list \( W \). Following the definition of \( B \), we either have
\[ L = L' \setminus P(\pi, L') \text{ for some } L' \in B'(\pi), \text{ or } (\pi = \tau \text{ and } L = L' \setminus P(\eta, L') \text{ for some } \eta \in \text{Tid and } L' \in B'(\eta)). \]

In the first case, we define \( W \triangleq \lambda k. W'_{(\pi, L')} (\text{Map}^{-1}(\lambda l. P(l, \eta, L'))(k)) \), and using the fact that \( W'_{(\pi, L')} \) is a \((G', \pi)\)-consistent \((G', L')\)-write-list, we show that \( W \) is a \((G, \pi)\)-consistent \((G, L)\)-write-list. In the latter, we define \( W \triangleq \lambda k. W'_{(\eta, L')} (\text{Map}^{-1}(\lambda l. P(l, \eta, L'))(k)) \), and using the fact that \( W'_{(\eta, L')} \) is a \((G', \eta)\)-consistent \((G', L')\)-write-list, we show that \( W \) is a \((G, \tau)\)-consistent \((G, L)\)-write-list.

In turn, for a read step, let \( r \) be the read event that is added in opSRA’s transition from \( G \) to \( G' \), and \( w \) be the write event that \( r \) reads from in \( G' \). Then, we construct \( B \) by setting \( B = B'[\tau \mapsto \langle \text{tid}(w), \text{loc}(r), \text{val}(r), R \rangle \cdot B'()] \), and \( B \overset{\tau}{\rightarrow}_{\text{loSRA}} B' \) follows by definition. Now, to show that \( B \subset G \), we let \( \pi \in \text{Tid} \) and \( L = B(\pi) \), and construct a \((G, \pi)\)-consistent \((G, L)\)-write-list \( W \). For a thread \( \pi \neq \tau \), we can simply use \( W'_{(\pi, L')} \) (as there is no change in the lists of \( \pi \)). For the reading thread \( \tau \), we define \( W \triangleq w \cdot W'_{(\tau, L')} \). Using the facts that \( W'_{(\tau, L')} \) is a \((G', \tau)\)-consistent \((G', L')\)-write-list and that \( w \notin \text{dom}(G, \text{mo} ; G, \text{hb}^+) \) (as required by opSRA’s read steps), we show that \( W \) is a \((G, \tau)\)-consistent \((G, L)\)-write-list.

\[ \square \]

5.1 Decidability of the Reachability Problem under SRA

We show how loSRA is used for establishing the decidability of the reachability problem under the declarative SRA model (see Def. 2.8). First, given the equivalence between SRA and opSRA (Thm. 4.7) and between opSRA and loSRA (Theorems 5.18 and 5.19), it suffices to establish the decidability of reachability under loSRA. That is: given a concurrent program \( P \) and a “bad state” \( \tilde{p} \in P.\text{Q} \), check whether \( \tilde{p} \) is reachable (see Def. 4.4) under the memory system loSRA. To show the decidability of this problem, we use the framework of well-structured transition systems. More precisely, we reduce reachability under loSRA to coverability in a well-structured transition system that meets the conditions ensuring that coverability is decidable.

5.1.1 Preliminaries on well-structured transition systems. We recall the relevant definitions and propositions about well-structured transition systems.

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 \( \{ s \in S \mid \exists u \in U. s \preceq u \} \); a set \( U \subseteq S \) is called upward closed if \( U = \uparrow U \); and a set \( S \subseteq U \subseteq S \) is called a basis of \( U \) if \( U = \uparrow \text{B} \). 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 \( A.\text{Q} \) such that \( \preceq \) is compatible with \( A \), that is: if \( q_1 \rightarrow_A q_2 \) and \( q_1 \preceq q_3 \), then there exists \( q_4 \in A.\text{Q} \) such that \( q_3 \rightarrow_A q_4 \) and \( q_2 \preceq q_4 \). The coverability problem for \( (A, \preceq) \) asks whether an input state \( q \in A.\text{Q} \) is coverable, namely: is some state \( q' \) with \( q \preceq q' \) reachable in \( A \)?

It is known (see, e.g., [Abdulla et al. 2000; Finkel and Schnoebelen 2001]) that coverability is decidable for a WSTS \( (A, \preceq) \) provided that \( \preceq \) is decidable and \( A \) admits the following properties:

(i) effective initialization: there exists an algorithm accepting a state \( q \in A.\text{Q} \) and deciding whether \( \uparrow\{q\} \cap A.\text{Q}_0 = \emptyset \).

(ii) effective pred-basis: there exists an algorithm accepting a state \( q \in A.\text{Q} \) and returning a finite basis of \( \uparrow\text{pred}_A(\uparrow\{q\}) \).

Roughly speaking, these conditions ensure that backward reachability analysis starting from \( q \) will converge to a fixed point, that each step in this calculation is effective, and that we can finally check whether some initial state is included in the fixed point.

5.1.2 loSRA as a Well-Structured Transition System. The \( \subseteq \) ordering on the states of loSRA is clearly decidable and also forms a wqo. Indeed, by Higman’s lemma, \( \subseteq \) is a wqo on the set of all
read-option lists. In turn, its lifting to potentials (which are finite by definition) is a wqo on the set of all potentials (see [Schmitz and Schnoebelen 2012]). Finally, by Dickson’s lemma the pointwise lifting of \( \sqsubseteq \) to functions assigning a potential to every \( \tau \in \text{Tid} \) (i.e., states of \( \text{loSRA} \)) is also a wqo.

Now, let \( P \) be a concurrent program. The \( \sqsubseteq \) ordering is naturally lifted to states of the concurrent system \( P_{\text{loSRA}} \) (that is, pairs \( \langle \bar{p}, B \rangle \in P.Q \times \text{loSRA}.Q \), see Def. 4.3) by defining \( \langle \bar{p}, B \rangle \sqsubseteq \langle \bar{\bar{p}}, B' \rangle \) iff \( \bar{p} = \bar{\bar{p}} \) and \( B \sqsubseteq B' \). Since \( P.Q \) is (by definition) finite and \( \sqsubseteq \) is a wqo on \( \text{loSRA}.Q \), we clearly have that \( \sqsubseteq \) is a wqo of \( P_{\text{loSRA}}.Q \). Since the first component (the program state) in \( \sqsubseteq \)-ordered pairs of \( P_{\text{loSRA}} \)'s states is equal, reachability under \( \text{loSRA} \) is reduced to coverability in \( P_{\text{loSRA}}.\sqsubseteq \).

It remains to show that the concurrent system \( P_{\text{loSRA}} \) equipped with \( \sqsubseteq \) is indeed a WSTS that admits effective initialization and effective pred-basis.

First, since \( \text{LOWER} \) is explicitly included in \( \text{loSRA} \), \( \sqsubseteq \) is clearly compatible with \( P_{\text{loSRA}} \). Indeed, given \( \langle \bar{p}_1, B_1 \rangle, \langle \bar{p}_2, B_2 \rangle, \langle \bar{p}_3, B_3 \rangle \in P_{\text{loSRA}}.Q \) such that \( \langle \bar{p}_1, B_1 \rangle \rightarrow_{P_{\text{SRA}}} \langle \bar{p}_2, B_2 \rangle \) and \( \langle \bar{p}_1, B_1 \rangle \sqsubseteq \langle \bar{p}_3, B_3 \rangle \) (so \( \bar{p}_1 = \bar{p}_3 \)), for \( \langle \bar{p}_4, B_4 \rangle = \langle \bar{p}_2, B_2 \rangle \), we have \( \langle \bar{p}_3, B_3 \rangle \rightarrow_{P_{\text{loSRA}}} \langle \bar{p}_4, B_4 \rangle \) (since \( B_3 \rightarrow_{\text{loSRA}} B_1 \) using the lower step) and \( \langle \bar{p}_2, B_2 \rangle \sqsubseteq \langle \bar{p}_4, B_4 \rangle \).

Second, \( P_{\text{loSRA}} \) trivially admits effective initialization. Indeed, the states \( \langle \bar{p}, B \rangle \) for which \( \uparrow \{ \langle \bar{p}, B \rangle \} \cap P_{\text{loSRA}}.Q \neq \emptyset \) are exactly the initial states themselves—\( P.Q_0 \times \{ \tau_r, \epsilon \} \).

Third, we show that \( P_{\text{loSRA}} \) admits effective pred-basis. For this matter, we demonstrate how to calculate a finite basis \( Q^\alpha \) of \( \text{pred}_{\text{loSRA}}^\alpha (\uparrow \{ B' \}) \) for each \( \alpha \) of the form \( (\tau, W(x, v_k), (\tau, R(x, v_k)), (\tau, \text{RMW}(x, v_R, v_k)), \epsilon) \) or \( (\tau, (\tau, x, v_k), u) \). Then, a finite basis of \( \text{pred}_{\text{loSRA}}^\alpha (\uparrow \{ \langle \bar{p}, B' \rangle \}) \) is given by \( \text{pred}_{\text{loSRA}}^\alpha (\uparrow \{ \langle \bar{p}, B' \rangle \}) \) for \( \alpha \neq \epsilon \); and by \( \{ \langle \bar{p}, B' \rangle \} \) for \( \alpha = \epsilon \) (silent memory step). In addition, for a silent program step, a finite basis of \( \text{pred}_{\text{loSRA}}^\alpha (\uparrow \{ \langle \bar{p}, B' \rangle \}) \) is given by \( \text{pred}_{\text{loSRA}}^\alpha (\uparrow \{ \langle \bar{p}, B' \rangle \}) \).

**Silent memory step** The set of predecessors of \( B' \) with respect to a silent memory step (i.e., using \( \text{LOWER} \)) is very simple—it contains any state \( B \) such that \( B' \sqsubseteq B \). Thus, \( \{ B' \} \) is a finite basis of \( \text{pred}_{\text{loSRA}}^\alpha (\uparrow \{ B' \}) \), as well as of \( \text{pred}_{\text{loSRA}}^\epsilon (\uparrow \{ B' \}) \).

**Read** A predecessor \( B \) of \( B' \) with respect to a read step \( B \) is similar to \( B' \), except for having in each read-option list of \( \tau \) an additional first read option of the form \( \langle \tau_w, x, v_k, R \rangle \). Hence, for \( \alpha = (\tau, R(x, v_k)) \), the set \( \{ B' \mid \tau \mapsto \langle \tau_w, x, v_k, u \rangle \cdot B'(\tau) \mid \tau_w \in \text{Tid}, u \in \{ R, \text{RMW} \} \) is a finite basis of \( \text{pred}_{\text{loSRA}}^\alpha (\uparrow \{ B' \}) \). It is also a basis of \( \text{pred}_{\text{loSRA}}^\alpha (\uparrow \{ B' \}) \) for a state \( B'' \sqsubseteq B' \), a corresponding read option \( \langle \tau_w, x, v_k, u \rangle \) appears in the lists of \( \tau \) in \( \text{pred}_{\text{loSRA}}^\alpha (\uparrow \{ B'' \}) \) before some additional read options, ensuring that \( \text{pred}_{\text{loSRA}}^\alpha (\uparrow \{ B'' \}) \sqsubseteq \text{pred}_{\text{loSRA}}^\alpha (\uparrow \{ B' \}) \).

**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 \( B' \), using Prop. 5.17 and the following technical lemma (proved in Appendix A):

**Lemma 5.20.** Let \( P \) be an index choice for \( B' \in \text{loSRA}.Q \) such that \( P \models \langle \tau, W(x, v_k) \rangle \). If \( B'_0 \sqsubseteq B' \), then \( \text{src}(B'_0, \tau, P_0) \sqsubseteq \text{src}(B, \tau, P) \) for some index choice \( P_0 \) for \( B'_0 \) such that \( P_0 \models \langle \tau, W(x, v_k) \rangle \).

By Prop. 5.17 and Lemma 5.20, we get a finite basis of \( \text{pred}_{\text{loSRA}}^{\langle \tau, W(x, v_k) \rangle} (\uparrow \{ B' \}) \), given by:

\[
\text{src}(B', \tau, P) \mid P \text{ is an index choice for } B' \text{ such that } P \models \langle \tau, W(x, v_k) \rangle.
\]

**RMW** The predecessor with respect to an RMW step intuitively combines the predecessors with respect to the read and write steps. By Prop. 5.17 and Lemma 5.20, we get that the set

\[
\text{src}(B', \tau, P)[\tau \mapsto \langle \tau_w, x, v_k, \text{RMW} \rangle \cdot \text{src}(B', \tau, P)(\tau)] \mid \tau_w \in \text{Tid} \text{ and } P \text{ is an index choice for } B' \text{ such that } P \models \langle \tau, W(x, v_k) \rangle
\]

is a finite basis of \( \text{pred}_{\text{loSRA}}^{\langle \tau, \text{RMW}(x, v_R, v_k) \rangle} (\uparrow \{ B' \}) \).
6 MAKING WEAK RELEASE/ACQUIRE LOSSY: THE loWRA MEMORY SYSTEM

In this section we establish the decidability of the reachability problem under WRA. As we did for SRA, we introduce an alternative memory system, which we call loWRA (for "lossy-WRA"), establish its equivalence to opWRA, and show how it is used to decide the reachability problem in the framework of well-structured transition systems.

The loWRA memory system is based on similar ideas as loSRA. In particular, loWRA also employs option lists and maintains the causality constraints, following the “shared-memory causality principle” (see §5), by requiring justifications for the positioning of added read options in other threads’ lists. The major difference is that loWRA allows existing read options from location \( x \) to appear in the potential of thread \( \tau \) after \( \tau \) writes to \( x \) (i.e., the write step does not have the \( (\pi = \tau \implies \forall o \in L_0 \cdots L_n. \ loc(o) \neq x) \) conjunct included in loSRA’s write step). Thus, loWRA allows certain outcomes that loSRA does not (e.g., the first three in Example 3.8).

However, we need certain limitations on accesses to \( x \) when thread \( \tau \) writes to \( x \) and adds it in other threads’ lists. First, if \( \tau \) 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 \( o_1, \ldots, o_n \). Second, if \( o_1, \ldots, o_n \) are added in a list of another thread \( \pi \), then after consuming \( o_1 \) but before consuming \( o_n \) thread \( \pi \) should not write to \( x \). Both these scenarios would violate weak-coherence.

To support these restrictions, the lists of loWRA include, in addition to read options, write options. These take the form \( O_w(x) \) where \( x \in \text{Loc} \). 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 \( \tau \) writes to \( x \), it (1) has to discard all of its lists that do not begin with \( O_w(x) \), and consume the \( O_w(x) \) option from the head of each of its remaining lists; (2) cannot place read options in its own lists after some \( O_w(x) \) option; and (3) cannot place read options in other threads’ lists in a way that will make some \( O_w(x) \) option appear between two of the new read options. The “shared-memory causality principle” (see §5) now applies not only to read options, but also to write options: if \( \tau \) has just performed a write \( w \), then after \( \pi \) reads from \( w \), it “synchronizes” with \( \tau \), an so its continuations (sequences of both reads and writes) should all be possible continuations of \( \tau \). In fact, as our correspondence proofs show, enforcing the “shared-memory causality principle” and conditions (1)-(3) above suffice to precisely capture (for now, the RMW-free fragment of) WRA.

Example 6.1. The annotated outcome of the Oscillating program in Example 3.8 can be obtained with the following (prefix of) run (using subscripts and superscripts for locations and values while eliding thread identifiers in read options):

\[
\{O_w(x)\} \{\epsilon\} \{\{O_w(x)\} \xrightarrow{T_1} \{\epsilon\} \{O_w(x) \cdot o^0_1\} \xrightarrow{T_3} \{\{O_w(x) \cdot o^1_1\} \cdot o^2_1\} \xrightarrow{T_2} \{\{o^1_2 \cdot o^0_2 \cdot o^1_3\} \cdot o^2_3\} \xrightarrow{T_2} \cdots
\]

We start with an \( O_w(x) \) option for \( T_1 \) and \( T_3 \). Then, \( T_1 \) executes its write: consumes its \( O_w(x) \), and adds two read options to \( T_2 \) and one to \( T_3 \). Now, \( T_3 \) executes its write: consumes its \( O_w(x) \) and adds in the list of \( T_2 \) a read option in between the two options that were added by \( T_1 \). This is justified since (after consuming \( O_w(x) \)) \( T_3 \) has \( o^1_3 \) in its list. Finally, \( T_2 \) can run and perform the three reads.

Example 6.2. We demonstrate why loWRA disallows the annotated outcome of the MP-trans program in Example 3.6. The first executed operation must be \( x := 0 \) by \( T_1 \). As \( T_3 \) reads \( 0 \) from \( x \), a corresponding read option \( o^0_3 \) has to be added to lists of \( T_3 \). Then, since \( T_3 \) will read \( 1 \) from \( x \) (which is written by \( T_2 \)) before it reads \( 0 \), when \( T_2 \) executes \( x := 1 \), a read option \( o^3_3 \) has to be added to lists of \( T_3 \) and be placed before \( o^0_3 \). The semantics of loWRA requires a justification for placing \( o^1_3 \) before \( o^3_3 \); a list of \( T_2 \) that contains \( O_w(x) \) and somewhere after it \( o^3_3 \). Hence, when \( T_1 \) executes \( x := 0 \), the read option \( o^0_3 \) should also be added to the lists of \( T_2 \) after \( O_w(x) \). Now, since \( T_2 \) reads \( 1 \) from \( y \) before it executes \( x := 1 \), when \( T_1 \) executes \( y := 1 \), a read option \( o^0_3 \) has to be added to lists of \( T_2 \),
and be placed before \(O_w(x)\) (which precedes \(a^0_x\)). In turn, this requires a justification in the form of a list of \(T_1\), that contains \(O_v(x)\) that precedes \(a^0_x\). Therefore, when \(T_1\) executes \(x := 0\), the read option \(a^0_x\) should also be added to the lists of \(T_1\), somewhere after \(O_w(x)\), which is disallowed by \(l_0\)WRA.

Finally, RMWs in \(l_0\)WRA are handled differently than in \(l_0\)SRA. Indeed, all we have in WRA is that two RMWs never read from the same event (and thus we cannot require, as required in \(l_0\)SRA, that after executing a write, no RMW will read from a write that was executed earlier). Naively, it 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 \(\tau\) an RMW-option, then later when it reads from a write of thread \(\tau\), it may still be able to perform an RMW, while thread \(\tau\) never had such option. To resolve this mismatch, we utilize the observation in Remark 3, and slightly modify the read options. Thus, instead of marking read options with RMW flags, we instrument them with RMW thread identifiers, denoting the 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 \(\tau\) can only consume read options whose RMW thread identifier is \(\tau\). Now, instead of saying that \(\pi\) has some option that \(\tau\) 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 options, write lists) overlap with these of \(\S 5\). To improve readability, we use the same terms, and the ambiguity is resolved by the context.

**Definition 6.3.** An option \(o\) is either \((\tau, x, v, \pi_{\text{RMW}})\) (read option) or \(O_v(x)\) (write option), where \(\tau, \pi_{\text{RMW}} \in \text{Tid}\), \(x \in \text{Loc}\) and \(v \in \text{Val}\). The functions \(\text{typ}, \text{tid}, \text{loc}, \text{val}\) and \(\text{rmw-tid}\) return (when applicable) the type (R/W), thread identifier (\(\tau\)), location (\(x\)), value (\(v\)), and RMW thread identifier (\(\pi_{\text{RMW}}\)) of a given option.

Option lists and potentials, as well as the \(\subseteq\) ordering, are defined exactly as in Definitions 5.6 and 5.7 (using Def. 6.3 instead of \(l_0\)SRA’s read options).

**Definition 6.4.** The memory system \(l_0\)WRA is defined by: \(l_0\)WRA.Q is the set of functions \(B\) assigning a potential to every \(\tau \in \text{Tid}\); \(l_0\)WRA.Q\(_0\) = \(\{B\mid \forall \tau \in \text{Tid}, L \in B(\tau), o \in L. \text{typ}(o) = W\}\); and the transitions are as follows:

The \text{READ}, \text{RMW} and \text{LOWER} steps are exactly as in \(l_0\)SRA. The \text{WRITE} step follows the intuitive explanation above: The writing thread consumes a write option to the written location. Then, every option list after the \text{WRITE} transition is obtained from some previous list \((O_v(x) \cdot L_0 \cdot \ldots \cdot L_n \in B(\tau))\) for the writing thread and \(L_0 \cdot \ldots \cdot L_n \in B(\tau)\) 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
obtained by following will not become aware of some write that is write event to which are now defined as sequences of write events and write options.

The simulation relation $\gamma$ between loWRA’s and opWRA’s states is obtained via write lists, which are now defined as sequences of write events and write options. In addition, we employ a mapping $\text{tid}_{\text{loWRA}} : W \to \text{Tid}$ (existentially quantified in the simulation relation) that assigns for every write event the (unique) thread identifier that may read it in an RMW operation.

**Definition 6.5.** A write list is a sequence of write events and write options. Let $G$ be an execution graph, $L$ an option list and $\text{tid}_{\text{loWRA}} : W \to \text{Tid}$. A write list $W$ is a $\langle G, L, \text{tid}_{\text{loWRA}} \rangle$-write-list if $|L| = |W|$ and the following hold for every $1 \leq k \leq |W|:

- If $\text{typ}(L(k)) = W$ then $W(k) = L(k)$.
- If $L(k) = \langle \tau, x, \nu, \pi_{\text{loWRA}} \rangle$ then $W(k) \in G.W$ and $\text{tid}(W(k)) = \tau$, $1\text{oc}(W(k)) = x$, $\text{val}_w(W(k)) = \nu$ and $\text{tid}_{\text{loWRA}}(W(k)) = \pi_{\text{loWRA}}$.

The following notion of $\langle G, \tau \rangle$-consistency of a write list $W$ intuitively means that weak-coherence is satisfied by the extension of the execution graph $G$ with a sequence of reads and writes of thread $\tau$ obtained by following $W$. For an element $w \in W$ that is in $G.W$, the corresponding extension of $G$ is a read event reading from $w$, and for an element of $W$ of the form $0_w(x)$, the extension of $G$ is a write event to $x$ (writing an arbitrary value). Thus, we should ensure that (1) $\tau$ is not already aware of some write that is $\text{hb}_{\text{loWRA}}$-later than some write of $W$; (2) after reading from a write $w_1$ of $W$, $\tau$ will not become aware of some write that is $\text{hb}_{\text{loWRA}}$-later than some write $w_2$ that appears after $w_1$ in $W$; (3) if $\tau$ is already aware of some write $w$ to $x$, then it cannot write to $x$ and then read from $w$; and (4) if $\tau$ 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 6.6.** A write list $W$ is a $\langle G, \tau \rangle$-consistent if for every $1 \leq k \leq |W|$ with $W(k) \in E$:

- $C1$ $W(k) \notin \text{dom}(G.\text{hb}_{\text{loWRA}}) ; |W| ; G.\text{hb}^2 ; [E^\tau \cup \{W(j) \mid 1 \leq j < k\}]$.
- $C2$ If $W(i) = 0_w(1\text{oc}(W(k)))$ for some $i < k$, then $W(k) \notin \text{dom}(G.\text{hb}_{\text{loWRA}}) ; [E^\tau]$.
- $C3$ For every $j < k$, if $W(i) = 0_w(1\text{oc}(W(k)))$ for some $j < i < k$, then $(W(k), W(j)) \notin G.\text{hb}^2$.

The simulation relation $\gamma$ is defined as for loSRA (Def. 5.13) using the new notions for write lists. In addition, it requires the existence of a mapping $\text{tid}_{\text{loWRA}} : W \to \text{Tid}$ relating every write event to the unique thread that may read from it in an RMW event, as enforced by the second requirement in the following definition. (Without such mapping, our backward simulation argument would fail, as we could have $G \cdot \tau.W(x, v_\omega)_{\text{opWRA}} G'$ and $B' \gamma G'$, where the “new” read options in $B'$ have different RMW thread identifiers, so no $B$ will satisfy $B \cdot \tau.W(x, v_\omega)_{\text{loWRA}} B'$ and $B \gamma G$.)

**Definition 6.7.** A state $B \in \text{loWRA.Q}$ matches an execution graph $G$, denoted by $B \gamma G$, if there exists a function $\text{tid}_{\text{loWRA}} : W \to \text{Tid}$, such that:

- For every $\tau \in \text{Tid}$ and $L \in B(\tau)$, there exists a $\langle G, \tau \rangle$-consistent $\langle G, L, \text{tid}_{\text{loWRA}} \rangle$-write-list.
- For every $\langle w, e \rangle \in G.\text{rf}; [\text{RMW}]$, we have $\text{tid}(e) = \text{tid}_{\text{loWRA}}(w)$.

As before, an alternative “backwards” formulation of the write step is convenient to use in our proofs. An index choice (Def. 5.14), as well as the source definition $(\text{src}(B', \tau, P))$, see Def. 5.16), are defined for states $B' \in \text{loWRA.Q}$ exactly as for states of $\text{loSRA.Q}$. 

What’s Decidable about Causally Consistent Memory Models?

Definition 6.8. An index choice $P$ for $B' \in \text{loWRA}$ justifies a $\langle \tau, w(x, v_W) \rangle$-step, denoted by $P \models \langle \tau, w(x, v_W) \rangle$, if the following hold:

- There exists $\pi_{\text{RMW}} \in \text{Tid}$ such that $L(k) = \langle \tau, x, v_W, \pi_{\text{RMW}} \rangle$ for every $\pi \in \text{Tid}$, $L' \in B'(\pi)$ and $k \in P(\pi, L')$.
- For every $\pi \in \text{Tid}$, $L' \in B'(\pi)$ and $k \in \{1, \ldots, |L'|\} \setminus P(\pi, L')$:
  - If $p_1 < k < p_2$ for some $p_1, p_2 \in P(\pi, L')$, then $L'(k) \neq O_w(x)$.
  - If $\pi = \tau$ and $k < p$ for some $p \in P(\pi, L')$, then $L'(k) \neq O_w(x)$.

Proposition 6.9. $B \xrightarrow{\tau, w(x, v_W)}_{\text{loWRA}} B'$ iff there exists an index choice $P$ for $B'$ such that $P \models \langle \tau, w(x, v_W) \rangle$, $0_w(x) \cdot \text{src}(B', \tau, P)(\tau) \subseteq B(\tau)$ and $\text{src}(B', \tau, P)(\pi) \subseteq B(\pi)$ for every $\pi \in \text{Tid} \setminus \{\tau\}$.

Next, we establish the equivalence of loWRA and opWRA. A full proof is provided in Appendix B.

Theorem 6.10. The traces of loWRA and the traces of opWRA coincide.

Proof Outline. Very roughly speaking, the proof proceeds similarly to the proof for SRA. Here, we only mention one notable detail that concerns the forward simulation (in which we show that every loWRA’s step from a state $B$ to a state $B'$ has a corresponding opWRA’s step from the state $G$, where $B \vdash G$, to some state $G'$ with $B' \vdash G'$).

When loWRA makes a write step, we construct $\langle G', \pi \rangle$-consistent write lists for each option list of every thread $\pi$ of $B'$. Note that each such option list has “old” read options that appeared in the corresponding list of $B$, and “new” read options that were added due to the write step. In a write list of a thread $\pi$, we associate each old read option $o$ that appears after a new option with the $\text{hb}$-maximal write among (i) the write event $w_1$ that was associated with $o$ in the corresponding list of $\pi$ in $B$, and (ii) the write event $w_2$ that was associated with $o$ in the justifying list of thread $\tau$ of $B$. For this construction to work, we need to know that $w_1$ and $w_2$ are ordered by $\text{hb}$. This holds due to the fact that read options include the writing thread, and so we have $\text{tid}(w_1) = \text{tid}(w_2)$.

To establish the decidability of the reachability problem under WRA, we follow the same arguments as in §5.1 and use the equivalence of loWRA and opWRA. The details are given in Appendix C.

7 CONCLUSION

We established the decidability of reachability under two main causal consistency models, SRA and WRA. For that matter, we developed novel operational semantics for the two models that meet the requirements for decidability of the framework of well-structured transition systems. Interestingly, under RA, which lies in between SRA and WRA, reachability is undecidable [Abdulla et al. 2019]. Intuitively, this stems from the fact that RA requires to maintain $\text{mo}$ during the execution, while SRA allows the execution of writes following $\text{hb} \cup \text{mo}$ and WRA does not employ $\text{mo}$ at all.

Besides the theoretical interest, Abdulla et al. [2018a] demonstrate that similar verification procedures with non-primitive recursive complexity may be practical for challenging (even though naturally quite small) algorithms and synchronization mechanisms. We plan to explore this in the future. We also believe that our semantics may be of independent interest in the development of verification techniques for programs running under weak consistency, including, but not limited to, 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., [Kuperstein et al. 2011; Suzanne and Miné 2016]). Other directions for future work include investigating the decidability of other variants of causal consistency as well as models in which only some accesses provide causal consistency, handling transactions (with semantics following, e.g., [Chong et al. 2018; Dongol et al. 2017]), and studying verification of parametrized programs under causal consistency (which is decidable for TSO [Abdulla et al. 2018a]).
REFERENCES


A FULL PROOFS - SRA

Theorem 5.18. Every trace of IoSRA is a trace of opSRA.

Proof. We show that \( \gamma \) constitutes a forward simulation relation from IoSRA to opSRA. First, the initial states clearly match: we have \( \lambda r \cdot \{ \epsilon \} \models G_0 \). Now, suppose that \( B \models \gamma G \) and \( B \overset{\tau}{\longrightarrow}_{\text{IoSRA}} B' \).

We show that there exists \( G' \) such that \( B' \models \gamma G' \) and \( G \overset{\tau,l}{\longrightarrow}_{\text{opSRA}} G' \). Consider the possible cases:

- \( l = W(x, v_w) \): Let \( w = \text{NextEvent}(G,E,\tau, l) \). Let \( G' \) be the execution graph defined by \( G'.E = G.E \cup \{ w \} \), \( G'.rf = G.rf \) and \( G'.mo = G.mo \cup (G.W_x \times \{ w \}) \). By definition, we have \( G \overset{\tau,l}{\longrightarrow}_{\text{opSRA}} G' \). We show that \( B' \models \gamma G' \). By Prop. 5.17, since \( B \overset{\tau,l}{\longrightarrow}_{\text{IoSRA}} B' \), there exists an index choice \( P \) for \( B' \) that justifies a \( (\tau,l) \)-step, such that \( \text{src}(B',\tau,P)(\pi) \subseteq B(\pi) \) for every \( \pi \in \text{Tid} \). Let \( \pi \in \text{Tid} \) and \( L' \in B'(\pi) \). We construct a \( (G',\pi) \)-consistent \( (G',L') \)-write-list \( W' \).

Let \( P \overset{\tau,l}{\longrightarrow}_{\text{IoSRA}} \pi \), \( L \overset{\tau}{\longrightarrow}_{\tau} L' \backslash P \), \( f \overset{\tau}{\longrightarrow}_{\tau} \text{Map}(L',P) \), \( L_r \overset{\tau}{\longrightarrow}_{\tau} L' \backslash P \) and \( f_r \overset{\tau}{\longrightarrow}_{\tau} \text{MMap}(L',P) \) (the last two are only defined if \( P \neq \emptyset \)). Since \( B \models \gamma G \), there exist a \( (G,\pi) \)-consistent \( (G,L) \)-write-list \( W \), and a \( (G,\tau) \)-consistent \( (G,L) \)-write-list \( W_r \). We define \( W' \) as follows:

\[
W' \overset{\tau,l}{\longrightarrow}_{\text{IoSRA}} \lambda k \in \{ 1, \ldots, |L'| \}, \begin{cases} \text{w} & k \in P \\ \text{w}(f(k)) & k < \min(P) \\ \text{max}_{\text{G.mo}}(W(f(k)), W_r(f_r(k))) & \text{otherwise} \end{cases}
\]

It is easy to see that \( W' \) is a \( (G',L') \)-write-list. In particular, to show that \( \tau \text{-write-list} \) implies \( \text{RMW} \) implies \( \tau \text{-write-list} \), we use the fact that \( P \) justifies a \( (\tau,l) \)-step, and so for every \( k \in \{ 1, \ldots, |L'| \} \backslash P \), we have that \( \tau \text{-write-list} \) implies \( \text{loc}(L'(k)) \neq x \).

We show that \( W' \) is \( (G',\pi) \)-consistent. Let \( 1 \leq k \leq |L'| \). We prove that \( W'(k) \neq \text{dom}(G'.mo \cup G'.hb^2) \cap \{ E^\pi \cup \{ W'(j) \mid 1 \leq j < k \} \} \). Suppose otherwise. First, note that we cannot have \( k \in P \), since \( w \) is a maximal element in \( G'.mo \). Let \( w_\pi = W(f(k)) \) and \( w_r = W_r(f_r(k)) \) (the latter is only defined if \( k < \min(P) \)).

Consider the two possible cases:

- \( W'(k) \in \text{dom}(G'.mo \cup G'.hb^2) \cap \{ E^\pi \} \): The definition of \( W' \) ensures that \( \langle w_\pi, W'(k) \rangle \in G'.mo^2 \), and so \( w_\pi \in \text{dom}(G'.mo \cup G'.hb^2) \cap \{ E^\pi \} \). Since \( W \) is \( (G,\pi) \)-consistent, we have that \( w_\pi \neq \text{dom}(G.mo \cup G.hb^2) \cap \{ E^\pi \} \), and therefore it must be the case that \( \pi = \tau \) and \( \langle w_\pi, w \rangle \in G'.mo \). Hence, \( \text{loc}(w_\pi) = x \), and so \( \text{loc}(L'(k)) = x \), which contradicts the fact that \( P \) justifies a \( (\tau,l) \)-step.

- \( \langle W'(k), W'(j) \rangle \in G'.mo \cup G'.hb^2 \) 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 \( W'(k) = \text{max}_{\text{G.mo}}(w_\pi, w_r) \).
    
    Hence, we have \( \langle w_r, W'(k) \rangle \in G'.mo^2 \), and so \( \langle w_r, w \rangle \in G'.mo \cup G'.hb^2 \). Now, if \( \langle w_r, w \rangle \in G'.mo \cup G'.hb^2 \cap \{ E^\pi \} \), which contradicts the fact that \( W \) is \( (G,\pi) \)-consistent. Therefore, we have \( \langle w_r, w \rangle \in G'.mo \). Hence, \( \text{loc}(w_r) = x \), and so \( \text{loc}(L'(k)) = x \), which contradicts the fact that \( P \) justifies a \( (\tau,l) \)-step.

  * \( W'(j) \neq w \): In this case, we must have \( \langle W'(k), W'(j) \rangle \in G.mo \cup G.hb^2 \). The definition of \( W' \) ensures that \( \langle w_\pi, W'(k) \rangle \in G.mo^2 \), and so \( \langle w_\pi, W'(j) \rangle \in G.mo \cup G.hb^2 \). Now, since \( W \) is \( (G,\pi) \)-consistent, we cannot have \( W'(j) = W(f(j)) \). Hence, \( j > \min(P) \) and \( W'(j) = W_r(f_r(j)) \).

    Let \( w_r = W_r(f_r(j)) \). It follows that \( k > \min(P) \), and so \( \langle w_r, W'(k) \rangle \in G.mo^2 \).

    Hence, we have \( \langle w_r, w_r \rangle \in G.mo \cup G.hb^2 \). This contradicts the fact that \( W \) is \( (G,\pi) \)-consistent.

- \( l = R(x, v_R) \): By definition, since \( B \overset{\tau,l}{\longrightarrow}_{\text{IoSRA}} B' \), there exists a read option \( o \) with \( \text{loc}(o) = x \) and \( \text{val}(o) = v_R \) such that \( B(\tau) = o \cdot B'(\tau) \). Since \( B \models \gamma G \), for every \( L \in B(\tau) \) there exists a \( (G,\tau) \)-consistent \( (G,L) \)-write-list \( W_L \). Let \( A = \{ W_L(1) \mid L \in B(\tau) \} \). Since \( B(\tau) \) is non-empty, we
know that $A$ is not empty. Since each $W_i$ is a $(G, L)$-write-list, we have that $\text{tid}(w) = \text{tid}(o)$ for every $w \in A$. Hence, $G, \text{po}$ totally orders $A$. Let $w = \min_{G, \text{po}} A$ and let $L_{\min} \in \mathcal{B}(\pi)$ such that $w = W_{\min}(1)$. Let $r = \text{NextEvent}(G, E, \pi, l)$ and let $G'$ be the execution graph given by $G'.E = G.E \cup \{r\}$, $G'.r.f = G.r.f \cup \{(w, r)\}$ and $G'.m.o = G.m.o$.

We show that $G \xrightarrow{r,l}_{\text{toSR}} G'$. By definition, it suffices to show the following:

- $w \in G.W_x$ and $\text{val}_w(w) = v_R$: We have $w = W_{\min}(1)$, and since $W_{\min}$ is a $(G, L_{\min})$-write-list, we have that $w \in G.W$, $\text{loc}(w) = \text{loc}(W_{\min}(1)) = \text{loc}(L_{\min}(1)) = \text{loc}(o) = x$ and $\text{val}_w(w) = \text{val}_w(W_{\min}(1)) = \text{val}(L_{\min}(1)) = \text{val}(o) = v_R$.

- $w \notin \text{dom}(G.m.o \cup G.hb') \cup [\mathcal{E}]$: Since $W_{\min}$ is $(G, \pi)$-consistent and $w = W_{\min}(1)$, we cannot have $w \in \text{dom}(G.m.o \cup G.hb') \cup [\mathcal{E}]$.

It remains to show that $B' \not\geq G'$. Let $\pi \in \text{Tid}$ and $L' \in B'(\pi)$. We define a $(G', \pi)$-consistent $(G', L')$-write-list. Consider two cases:

- $\pi \neq \tau$: Let $L = o \cdot L'$. Then, $L \in \mathcal{B}(\pi)$. Let $W' = \lambda k \in \{1, \ldots, |L'|\}$. $W_l(1 + k)$ is easy to see that $W'$ is a $(G', L')$-write-list. Suppose by contradiction that $W'(k) \in \text{dom}(G'.m.o \cup G'.h\beta') \cup [\mathcal{E} \cup \{(w, j) \mid 1 \leq j < k\}]$. Then it follows that $W_l(1 + k) \subset \text{dom}(G.m.o \cup G.h\beta') \cup [\mathcal{E} \cup \{(w, j) \mid 1 \leq j < k\}]$, which contradicts the fact that $W_l$ is $(G, \tau)$-consistent. Hence, we must have $W'(k, w) \in G.m.o \cup G.h\beta'$.

By definition, $B \xrightarrow{r,l}_{\text{toSR}} B'$, there exists a read option $o$ with $\text{loc}(o) = x$, $\text{val}(o) = v_R$ and $\text{rmw-flag}(o) = \text{RMW}$ such that $L(1) = o$ for every $L \in \mathcal{B}(\pi)$. Since $B \not\geq G$, for every $L \in \mathcal{B}(\pi)$ there exists a $(G, \pi)$-consistent $(G, L)$-write-list $W_l$. Moreover, since $\text{rmw-flag}(o) = \text{RMW}$, we have $W_l(1) = \max_{G.m.o} G.W_x$ for every $L \in \mathcal{B}(\pi)$. Let $w = \max_{G.m.o} G.W_x$, $\epsilon = \text{NextEvent}(G.E, \pi, l)$ and $G'$ be the execution graph given by $G'.E = G.E \cup \{\epsilon\}$, $G'.r.f = G.r.f \cup \{(w, \epsilon)\}$ and $G'.m.o = G.m.o \cup (G.W_x \times \{\epsilon\})$.

For showing that $G \xrightarrow{r,l}_{\text{toSR}} G'$, it suffices, by definition, to show that $\text{val}_w(w) = v_R$. Indeed, since $B(\pi)$ is (by definition) non-empty, we can take some $L \in \mathcal{B}(\pi)$. We have $w = W_l(1)$, and since $W_l$ is a $(G, L)$-write-list, we have that $\text{val}_w(w) = \text{val}_w(W_l(1)) = \text{val}(L(1)) = \text{val}(o) = v_R$.

It remains to show that $B' \not\geq G'$. Using Prop. 5.17, since $B \xrightarrow{r,l}_{\text{toSR}} B'$, we know that there exists an index choice $P$ for $B'$ that justifies a $(\tau, W(x, w))$-step, such that $\text{src}(B', \tau, P)(\pi) \subseteq \mathcal{B}(\pi)$ for every $\pi \in \text{Tid} \setminus \{\tau\}$ and $o \cdot \text{src}(B', \tau, P)(\pi) \subseteq \mathcal{B}(\pi)$.

Let $\pi \in \text{Tid}$ and $L' \in B'(\pi)$. We construct a $(G', \pi)$-consistent $(G', L')$-write-list $W'$. Let $P \triangleq P(\pi, L')$ and (the last two are only defined if $P \neq \emptyset$):

$$L \triangleq \begin{cases} L' \setminus P & \pi \neq \tau \\ o \cdot (L' \setminus P) & \pi = \tau \end{cases}$$

$$f \triangleq \begin{cases} \text{Map}(L', P) & \pi \neq \tau \\ \lambda k \in \{1, \ldots, |L'|\} \setminus P. \text{Map}(L', P)(k) + 1 & \pi = \tau \end{cases}$$

$$L_\tau \triangleq o \cdot L' \setminus P$$

$$f_\tau \triangleq \lambda k \in \{\min(P), \ldots, |L'|\} \setminus P. \text{MMap}(L', P)(k) + 1$$
Since $\mathcal{B} \supseteq G$, 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 $W_{\tau}$. We define $W'$ as follows:

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

It is easy to see that $W'$ is a $\langle G', L' \rangle$-write-list. In particular, to show that $\text{rmw-flag}(L'(k)) = \text{RMW}$ implies $W'(k) \notin \text{dom}(G'.\text{mo})$, we use the fact that $\mathcal{P}$ justifies a $\langle \tau, w(\pi, \nu) \rangle$-step, and so for every $k \in \{1, \ldots, |L'|\} \setminus P$, we have that $\text{rmw-flag}(L'(k)) = \text{RMW}$ implies $\text{loc}(L'(k)) \neq x$.

We show that $W'$ is $\langle G', \pi \rangle$-consistent.

Let $1 \leq k \leq |L'|$. We prove that $W'(k) \notin \text{dom}(G'.\text{mo}; G'.\text{hb}^2; |E^\pi \cup \{W'(j) \mid 1 \leq j < k\}|)$. Suppose otherwise. First, note that we cannot have $k \in P$, since $e$ is a maximal element in $G'.\text{mo}$. 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:

1. $\langle W'(k), W'(j) \rangle \in G'.\text{mo}; G'.\text{hb}^2$ 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 $W'(k) = \max_{G, \text{mo}}\{w_\pi, w_{\tau}\}$.
     - There are three possibilities:
       - $W'(k) = w$: Then $\text{loc}(w_{\tau}) = \text{loc}(L'(k)) = x$, which contradicts the fact that $\mathcal{P}$ justifies a $\langle \tau, w(\pi, \nu) \rangle$-step.
       - $(W'(k), w) \in G'.\text{mo}; G'.\text{hb}^2$: This contradicts the $\langle G, \tau \rangle$-consistency of $W_{\tau}$, as $W_{\tau}(1) = w$ and $(w_\pi, W'(k)) \notin G'.\text{mo}^2$, implying that $(w_{\tau}, W_{\tau}(1)) \in G.\text{mo}^2; G.\text{hb}^2$.
       - $(W'(k), e) \in G'.\text{mo}; G'.\text{hb}^2; G'.\text{po}$: This also contradicts the $\langle G, \tau \rangle$-consistency of $W_{\tau}$, as we get that $w_{\tau} \in \text{dom}(G.\text{mo}; G.\text{hb}^2; |E^\pi|)$.

2. $W'(j) \neq e$: In this case, we must have $(W'(k), W'(j)) \in G.\text{mo}; G.\text{hb}^2$. The definition of $W'$ ensures that $(w_\pi, W'(k)) \in G.\text{mo}^2$, and so $(w_\pi, W'(j)) \in G.\text{mo}; G.\text{hb}^2$. 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 $(w_{\tau}, W_{\tau}(k)) \notin G.\text{mo}^2$.

Hence, we have $(w_\pi, w_{\tau}') \in G.\text{mo}^2; G.\text{hb}^2$. This contradicts the fact that $W_{\tau}$ is $\langle G, \tau \rangle$-consistent.

Finally, suppose that $\mathcal{B} \supseteq G$ and $\mathcal{B} \xrightarrow{\text{loSRA}} \mathcal{B}'$ (using the lower step). We show that $\mathcal{B}' \supseteq G$. Let $\tau \in \text{Tid}$ and $L' \in \mathcal{B}'(\tau)$. We define a $\langle G, \tau \rangle$-consistent $\langle G, L' \rangle$-write-list $W'$. By definition, since $\mathcal{B} \xrightarrow{\text{loSRA}} \mathcal{B}'$, there exists $L \in \mathcal{B}(\tau)$ such that $L' \subseteq L$. Let $f : \{1, \ldots, |L'|\} \to \mathbb{N}$ be an increasing function such that $L'(k) = L(f(k))$ for every $k \in \text{dom}(f)$. Since $\mathcal{B} \supseteq G$, 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 $(G, L')$-write-list. We show that $W'$ is $\langle G, \tau \rangle$-consistent. Let $1 \leq k \leq |L'|$. Suppose by contradiction that $W'(k) \in \text{dom}(G.\text{mo}; G.\text{hb}^2; |E^\pi \cup \{W'(j) \mid 1 \leq j < k\}|)$. It follows that $W(f(k)) \in \text{dom}(G.\text{mo}; G.\text{hb}^2; |E^\pi \cup \{W(f(j)) \mid 1 \leq j < k\}|)$. This contradicts the fact that $W$ is $\langle G, \tau \rangle$-consistent. 

\(\square\)

**Theorem 5.19.** Every trace of opSRA is a trace of loSRA.
PROOF. We show that $\gamma^{-1}$ constitutes a backward simulation from opSRA to loSRA.\footnote{Recall that a backward simulation from an LTS $A$ to an LTS $B$ is a relation $R \subseteq A.Q \times B.Q$ such that (1) $R$ is total (for every $q \in A.Q$ we have $(q, p) \in R$ for some $p \in B.Q$); (2) if $(q, p) \in R$ and $q \in A.Q_0$, then $p \in B.Q_0$; and (3) if $q \xrightarrow{} A q'$ and $(q', p') \in R$, then there exists $p \in B.Q$ such that $p \xrightarrow{} B p'$.}

The two first requirements of a backward simulation clearly hold for $\gamma$: (1) $\gamma^{-1}$ is total, as for every state $G$ of opSRA, we have $(\lambda \tau \in \text{Tid}. \{\epsilon\}) \gamma G$. (2) Consider a state $B$ of loSRA, such that $B \gamma G_0$. By the definition of $\gamma$, it should be possible to link every read option of $B$ to some write event of $G_0$. Since there are no write events in $G_0$, there cannot be read options in $B$, having $B = \lambda \tau \in \text{Tid}. \{\epsilon\} \subseteq \text{loSRA}_0$.

We move to the third requirement. Suppose that $G \xrightarrow{\tau,l} \text{opSRA} G'$ and $B' \gamma G'$. We construct a state $B$ such that $B \xrightarrow{\tau,l} \text{loSRA} B'$ and $B \gamma G$. Consider the possible cases:

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

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

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

By Prop. 5.17, to show that $B \xrightarrow{\tau,l} \text{loSRA} B'$, it suffices to prove that $P$ justifies a $(\tau, \text{W}(x, \nu_w))$-step. Thus, we show that the following hold for every $\pi \in \text{Tid}$ and $L' \in B'(\pi)$, where $P = P(\pi, L')$ and $W' = W'_{(\pi, L')}:

- Let $k \in P$. To see that $L'(k) \in \{(\tau, x, \nu_w, \tau), (\tau, x, \nu_w, \text{RMW})\}$, note that since $k \in P$, we have $W'(k) = w$, and since $W'$ is a $(G', L')$-write-list, we must have $\tau = \text{tid}(w) = \text{tid}(L'(k))$, $x = 1\text{oc}(w) = 1\text{oc}(L'(k))$ and $\nu_w = \text{val}_w(w) = \text{val}_w(L'(k))$.

- Let $k \in \{1, \ldots, |L'|\} \setminus P$, such that $1\text{oc}(L'(k)) = x$. We show that $\text{rmw-flag}(L'(k)) = \mathbb{R}$. Let $w' = W'(k)$. Since $k \not\in P$, we have $w' \neq w$. Since $G'.mo = G.mo \cup (G.W_x \times \{w\})$, it follows that $\langle w', w \rangle \in G'.mo$. However, since $W'$ is a $(G', L')$-write-list, if $\text{rmw-flag}(L'(k)) = \mathbb{R}$, then we must have $w' = \text{max}_{G'.mo} G'.W_x$, reaching a contradiction.

- Suppose $\pi = \tau$ and let $k \in \{1, \ldots, |L'|\} \setminus P$. We show that $1\text{oc}(L'(k)) \neq x$. Suppose otherwise. Let $w' = W'(k)$. Since $k \not\in P$, we have $w' \neq w$. Hence, since $G'.mo = G.mo \cup (G.W_x \times \{w\})$, we have $\langle w', w \rangle \in G'.mo$. Thus, $\langle w', W'(m) \rangle \in G'.mo \cup G'.hb^2$. Since $k > m$, this contradicts the fact that $W'$ is $(G', \pi)$-consistent.

Thus, we have shown that for every $k$, $W'(k) \notin \text{dom}(G'.mo) \cup G'.hb^2 \cup \{\mathbb{E} \}$. Indeed, if $W'(k) \in \text{dom}(G'.mo) \cup G'.hb^2 \cup \{\mathbb{E} \}$ then since $G'.mo \subseteq G.mo$ and $G'.hb \subseteq G'.hb$, it follows...
that $W'(f(k)) \in dom(G'.mo ; G'.hb^2 ; [E^r])$, which contradicts the $\langle G', \pi \rangle$-consistency of $W'$. Analogously, if $W(k) \in dom(G.mo ; G.hb^2 ; \{W(j) \mid 1 \leq j < k\})$ then since $f$ is an increasing function, we have $W'(f(k)) \in dom(G'.mo ; G'.hb^2 ; \{W'(f(j)) \mid 1 \leq j < k\})$, which contradicts the $\langle G', \pi \rangle$-consistency of $W'$.

- $\pi = \tau$ and $L = L' \setminus \mathcal{P}(\eta, L')$ for some $\eta \in Tid$ and $L' \in \mathcal{B}(\eta)$. Let $P = \mathcal{P}(\eta, L')$, $m = min(P)$, $W' = W'_{(\eta, L')}$ and $f = \text{MMap}^{-1}_{(L', P)}$. We define $W = lk \in \{1, \ldots, |L|\}$. $W'(f(k))$. 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 $r_{mw-f1ag}(L(k)) \neq R_{mw}$ whenever $1\text{oc}(L(k)) = x$.) It remains to show that $W$ is $\langle G, \tau \rangle$-consistent, namely to prove that $W(k) \notin dom(G.mo ; G.hb^2 ; [E^r \cup \{W(j) \mid 1 \leq j < k\}]$ for every $k$. Indeed, if $W(k) \in dom(G.mo ; G.hb^2 ; \{W(j) \mid 1 \leq j < k\})$ then since $f$ is an increasing function, we have $W'(f(k)) \in dom(G'.mo ; G'.hb^2 ; \{W'(j) \mid 1 \leq j < f(k)\})$, which contradicts the $\langle G', \eta \rangle$-consistency of $W'$. Now, if $W(k) \in dom(G.mo ; G.hb^2 ; [E^r])$, then since $w = \max_{G'.po} G'.E^r$, we have $w \notin dom(G_mo ; G_hb^2 ; [E^r])$.

Let $o$ be the read option given by $o \triangleq (\text{tid}(w), x, \nu_r, R)$. We define $B$ by:

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

By definition, $B \xrightarrow{\tau,l}_{\text{opusra}} B'$.

We show next that $B \not\gamma G$. For a thread $\pi \neq \tau$ and a read-option list $L \in B(\pi)$, observe that $L \in B'(\pi)$, and since $B' \not\gamma G'$, there is a $\langle G', \pi \rangle$-consistent $\langle G', L' \rangle$-write-list $W'$. Since $G.mo \subseteq G'.mo$ and $G.hb \subseteq G'.hb$, $W'$ is also $\langle G, \pi \rangle$-consistent $\langle G, L \rangle$-write-list.

Consider a read-option list $L \in B(\pi)$. Let $L' \in B'(\tau)$ such that $L = o \cdot L'$. Since $B' \not\gamma G'$, 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. Let $1 \leq k \leq |L|$. We prove that $W(k) \notin dom(G.mo ; G.hb^2 ; [E^r \cup \{W(j) \mid 1 \leq j < k\}]$). Suppose otherwise. Consider the two possible cases:

- $k = 1$. Then $w \in dom(G.mo ; G.hb^2 ; [E^r])$, which contradicts the properties of $w$ as stated above.

- $k > 1$. Observe that $W(k) = W'(k-1)$. If $W(k) \in dom(G.mo ; G.hb^2 ; [E^r \cup \{W(j) \mid 2 \leq j < k\}]$ then $W'(k-1) \in dom(G'.mo ; G'.hb^2 ; [E^r \cup \{W'(j) \mid 1 \leq j < k-1\}]$, contradicting the $\langle G', \tau \rangle$-consistency of $W'$. Thus, $\langle W(k), W(1) \rangle \in G.mo ; G.hb^2$. Yet, $W(1) = w$, $r \in E^r$ and $\langle w, r \rangle \in G'.rf$. Hence, $W'(k-1) \in dom(G'.mo ; G'.hb^2 ; [E^r])$, contradicting the $\langle G', \tau \rangle$-consistency of $W'$.

- $l = R_{mw}(x, \nu_r, \nu_W)$: This case combines the proofs given for the read and write lists.

Let $e = \text{NextEvent}(G,E, \tau, l)$. Since $G \xrightarrow{\tau,l}_{\text{opusra}} G'$, we have $G'.E = G.E \cup \{e\}$, $G'.mo = G.mo \cup (W_x \times \{e\})$, $G'.rf = G.rf \cup \{(w, e)\}$ and $\text{val}_w(w) = \nu_r$, where $w = \max_{G.mo} W_x$. Since $B' \not\gamma G'$, for every $\pi \in Tid$ and $L' \in B'(\pi)$ there exists a $\langle G', \pi \rangle$-consistent $\langle G', L' \rangle$-write-list $W'_{(\pi, L')}$. 


Let $\mathcal{P}$ be the index choice for $\mathcal{B}'$ that assigns the set of “new” positions in $\mathcal{B}'$:

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

Then, we define:

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

where $o$ is the read option given by $o \triangleq \langle \text{tid}(w), x, v_r, \text{RMW} \rangle$.

The arguments for why $\mathcal{B} \xrightarrow{\tau,l}_{\text{loSRA}} \mathcal{B}'$ are analogous to those of the write case. Using Prop. 5.17, to show that $\mathcal{B} \xrightarrow{\tau,l}_{\text{loSRA}} \mathcal{B}'$, it suffices to prove that $\mathcal{P}$ justifies a $\langle \tau, W(x, v_b) \rangle$-step. This is done exactly as in the write case.

It remains to show that $\mathcal{B} \wedge G$. Let $\pi \in \text{Tid}$ and $L \in \mathcal{B}(\pi)$. We show that there exists a $(G, \pi)$-consistent $(G, L)$-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 $P = \mathcal{P}(\tau, L')$, $W' = W'_{(\tau, L')}$ and $f = \lambda k \in \{2, ..., |L|\}$. $\text{Map}^{-1}_{(L, P)}(k - 1)$. We define

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

Using the fact that $W'$ is a $(G', L')$-write-list and that $w = \max_{G, mo} W_x$, it is easy to see that $W$ is a $(G, L)$-write-list. (In particular, note that for $k > 1$, $\text{rmw-flag}(L(k)) \neq \text{RMW}$ whenever $\text{loc}(L(k)) = x$.) It remains to show that $W$ is $(G, \tau)$-consistent, namely to prove that for every $k \in \{1, ..., |L|\}$, we have $W(k) \notin \text{dom}(G.mo; G.hb^7; [E^T \cup \{W(j) \mid 1 \leq j < k\}])$.

For $k = 1$, this is trivial since $W(1) = w = \max_{G, mo} W_x$. Let $k \in \{2, ..., |L|\}$. If $W(k) \notin \text{dom}(G.mo; G.hb^7; [E^T])$ then since $G.mo \subseteq G'.mo$ and $G.hb \subseteq G'.hb$, we have $W'(f(k)) \notin \text{dom}(G'.mo; G'.hb^7; [E^T])$, which contradicts the $(G', \tau)$-consistency of $W'$. Analogously, if $\langle W(k), W(j) \rangle \in G.mo; G.hb^7$ for $2 \leq j < k$ then $\langle W'(f(k)), W'(f(j)) \rangle \in G'.mo; G'.hb^7$, and since $f$ is an increasing function this contradicts the $(G', \tau)$-consistency of $W'$. Now, if $\langle W(k), W(1) \rangle \in G.mo; G.hb^7$, then since $W(1) = w$, $G'.E = G.E \cup \{e\}$ and $G'.rf = G.rf \cup \{(w, e)\}$, we have $W'(f(k)) \notin \text{dom}(G'.mo; G'.hb^7; [G'E^T])$, which contradicts the $(G', \tau)$-consistency of $W'$.
- $\pi = \tau$ and $L = o \cdot (L' \setminus \mathcal{P}(\eta, L'))$ for some $\eta \in \text{Tid}$ and $L' \in \mathcal{B}'(\eta)$. Let $P = \mathcal{P}(\eta, L')$, $m = \min(P)$, $W' = W'_{(\eta, L')}$ and $f = \lambda k \in \{2, ..., |L|\}$. $\text{MMap}^{-1}_{(L, P)}(k - 1)$. We define

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

As above, $W$ is a $(G, L)$-write-list, and we show that it is $(G, \tau)$-consistent. Namely, we prove that for every $k \in \{1, ..., |L|\}$, we have that $W(k) \notin \text{dom}(G.mo; G.hb^7; [E^T \cup \{W(j) \mid 1 \leq j < k\}])$.

Again, for $k = 1$, this is trivial since $W(1) = w = \max_{G, mo} W_x$. Let $k \in \{2, ..., |L|\}$. If $\langle W(k), W(j) \rangle \in G.mo; G.hb^7$ for $2 \leq j < k$, then $\langle W'(f(k)), W'(f(j)) \rangle \in G'.mo; G'.hb^7$, and since $f$ is an increasing function this contradicts the $(G', \tau)$-consistency of $W'$. Now, if $\langle W(k), W(1) \rangle \in G.mo; G.hb^7$, then since $W(1) = w$ and $\langle w, e \rangle \in G'.rf$, we have $\langle W'(f(k)), e \rangle \in G'.mo; G'.hb^7$. However, $W'(m) = e$ and $f(k) > m$, implying that
\( W'(f(k)) \in \text{dom}(G'.\text{mo} ; G'.\text{hb})^2 ; \{ \{ W'(j) \mid 1 \leq j < f(k) \} \} \), which contradicts the \( \langle G', \eta \rangle \)-consistency of \( W' \).

Lastly, if \( W(k) \in \text{dom}(G.\text{mo} ; G.\text{hb})^2 ; \{ E \} \), then \( \langle W'(f(k)), e \rangle \in \langle G'.\text{mo} ; G'.\text{hb} \rangle^2 \) (since \( e = \max_{G'.\text{po}} G'.\text{E} \)). However, \( W'(m) = e \) and \( f(k) > m \), implying that \( W'(f(k)) \in \text{dom}(G'.\text{mo} ; G'.\text{hb})^2 ; \{ \{ W'(j) \mid 1 \leq j < f(k) \} \} \), which contradicts the \( \langle G', \eta \rangle \)-consistency of \( W' \).

\[ \square \]

**Lemma 5.20.** Let \( \mathcal{P} \) be an index choice for \( \mathcal{B}' \in \text{loSRA}_Q \) such that \( \mathcal{P} \models \langle \tau, w(x, \nu) \rangle \). If \( \mathcal{B}' \subseteq \mathcal{B}' \), then \( \text{src}(\mathcal{B}', \tau, \mathcal{P}) \subseteq \text{src}(\mathcal{B}, \tau, \mathcal{P}) \) for some index choice \( \mathcal{P}_0 \) for \( \mathcal{B}' \) such that \( \mathcal{P}_0 \models \langle \tau, \nu(x, \nu) \rangle \).

**Proof.** Since \( \mathcal{B}' \subseteq \mathcal{B}' \), for every \( \pi \in \text{Tid} \), there exists a function \( F_\pi : \mathcal{B}'(\pi) \rightarrow \mathcal{B}'(\pi) \) such that for every \( L'_0 \in \mathcal{B}'(\pi) \), we have \( L'_0 \subseteq F_\pi(L'_0) \), witnessed by a strictly increasing function \( f_{\langle \pi, L'_0 \rangle} : \{ 1, \ldots, |L'_0| \} \rightarrow \{ 1, \ldots, |F_\pi(L'_0)| \} \), such that \( L'_0(k) = (F_\pi(L'_0))(f_{\langle \pi, L'_0 \rangle}(k)) \) for every \( k \in \{ 1, \ldots, |L'_0| \} \).

We define \( \mathcal{P}_0 \) to be the positions in \( \mathcal{P} \) that originated in \( \mathcal{B}' \), according to the \( f_{\langle \pi, L'_0 \rangle} \) functions. That is,

\[ \mathcal{P}_0 \triangleq \lambda \pi \in \text{Tid}, L'_0 \in \mathcal{B}'(\pi). \{ k \in \{ 1, \ldots, |L'_0| \} \mid f_{\langle \pi, L'_0 \rangle}(k) \in \mathcal{P}(\pi, F_\pi(L'_0)) \}. \]

It is easy to verify that \( \mathcal{P}_0 \) justifies a \( \langle \tau, w(x, \nu) \rangle \)-step. Let \( \mathcal{B}_0 = \text{src}(\mathcal{B}', \tau, \mathcal{P}_0) \). We show that \( \mathcal{B}_0 \subseteq \text{src}(\mathcal{B}', \tau, \mathcal{P}) \).

Recall that for every thread \( \pi \in \text{Tid} \), we have that every list \( L_0 \in \mathcal{B}_0(\pi) \) is equal to \( L'_0 \setminus \mathcal{P}_0(\pi, L'_0) \) (resp. to \( L'_0 \setminus \mathcal{P}_0(\eta, L'_0) \)) for some list \( L'_0 \) of \( \mathcal{B}'(\pi) \) (resp. for some list \( L'_0 \) of \( \mathcal{B}'(\eta) \) for some \( \eta \in \text{Tid} \)). Hence, we can define a function \( H_\pi : \mathcal{B}_0(\pi) \rightarrow \text{src}(\mathcal{B}', \tau, \mathcal{P})(\pi) \), by setting \( H_\pi(L_0) = F_\pi(L'_0) \setminus \mathcal{P}(\pi, F_\pi(L'_0)) \). Observe that for every \( L_0 \in \mathcal{B}_0(\pi) \), we have \( L_0 \subseteq H_\pi(L_0) \), witnessed by the function \( h_{\langle \pi, L_0 \rangle} : \{ 1, \ldots, |L_0| \} \rightarrow \{ 1, \ldots, |H_\pi(L_0)| \} \), defined by

\[ h_{\langle \pi, L_0 \rangle}(k) \triangleq \text{Map}(F_\pi(L'_0), \mathcal{P}(\pi, F_\pi(L'_0)))(f_{\langle \pi, L'_0 \rangle}(\text{Map}^{-1}_{\langle L'_0, \mathcal{P}(\pi, L'_0) \rangle}(k))), \]

for every \( k \in \{ 1, \ldots, |L_0| \} \). (Respectively, we define \( H_\pi(L_0) = F_\eta(L'_0) \setminus \mathcal{P}(\eta, F_\eta(L'_0)) \), witnessed analogously.)

\[ \square \]
B  FULL PROOFS - WRA

THEOREM 6.10. The traces of loWRA and the traces of opWRA coincide.

PROOF of (⊆) direction: Every trace of loWRA is a trace of opWRA. We show that \( G \subseteq \gamma \) constitutes a forward simulation relation from loWRA to opWRA. First, the initial states clearly match: for every \( B \in \text{loWRA}, G_0 \) we have \( B \gamma G_0 \), since (using any function \( \text{tid}_{\text{opWRA}} : W \rightarrow \text{Tid} \)) for every \( \tau \in \text{Tid} \) and \( L \in B(\tau) \), \( L \) itself, having only write options, is a \( (G, \tau) \)-consistent \( (G, L) \)-write-list, regardless of what \( G \) is.

Now, suppose that \( B \gamma G \) and \( B \xrightarrow{\tau} \text{loWRA} \ B' \). Let \( \text{tid}_{\text{opWRA}} : W \rightarrow \text{Tid} \) that satisfies the conditions of Def. 6.7. We show that there exists \( G' \) such that \( B' \gamma G' \) and \( G \xrightarrow{\tau} \text{opWRA} \ G' \). Consider the possible cases:

- \( I = \mathcal{W}(x, v_h) \): Let \( w = \text{NextEvent}(G.E, \tau, l) \). Let \( G' \) be the execution graph defined by \( G'.E = G.E \cup \{ w \} \) and \( G'.r.f = G.r.f \). By definition, we have \( G \xrightarrow{\tau} \text{opWRA} \ G' \).

We show that \( B' \gamma G' \). First, since \( B \xrightarrow{\tau} \text{loWRA} \ B' \), by Prop. 6.9, there exists an index choice \( \mathcal{P} \) for \( B' \) that justifies a \( (\tau, l) \)-step, such that \( \sigma_r(B', \tau, \mathcal{P})(\pi) \subseteq \mathcal{B}(\pi) \) for every \( \pi \in \text{Tid} \setminus \{ \tau \} \) and \( \mathcal{O}_0(x) \cdot \sigma_r(B', \tau, \mathcal{P})(\tau) \subseteq \mathcal{B}(\tau) \). Since \( \mathcal{P} \) justifies a \( (\tau, l) \)-step, there exists \( \text{tid}_{\text{opWRA}} \in \text{Tid} \), such that \( L'(k) = \langle \tau, x, v_h, \pi_{\text{opWRA}} \rangle \) for every \( \pi \in \text{Tid} \), \( L' \in B'(\pi) \) and \( k \in \mathcal{P}(\pi, L') \).

Let \( \text{tid}_{\text{opWRA}} = \text{tid}_{\text{opWRA}}[w \mapsto \pi_{\text{opWRA}}] \). Since \( w \notin G'.r.f \), we vacuously have \( \text{tid}(e) = \text{tid}_{\text{opWRA}}(w) \) for every \( (w, e) \in G'.r.f \); [RMW]. It follows that for every \( (w', e) \in G'.r.f \), \( G' \) and \( G \) have the same behavior:

\[
L \triangleq \begin{cases} 
L' \setminus P & \pi \neq \tau \\
\mathcal{O}_0(x) \cdot (L' \setminus P) & \pi = \tau
\end{cases} \\
L_{\tau} \triangleq \mathcal{O}_0(x) \cdot L' \setminus P \\
f_{\tau} \triangleq \lambda k \in \{ \min(P) , \ldots , |L'| \} \setminus P. \text{MMap}_{L', P}(k) + 1
\]

Then, by definition, we have \( L \in B(\pi) \) and \( L_{\tau} \in B(\tau) \). Let \( W \) be a \( (G, \pi) \)-consistent \( (G, L, \text{tid}_{\text{opWRA}}) \)-write-list, and \( W_t \) be a \( (G, \tau) \)-consistent \( (G, L_{\tau}, \text{tid}_{\text{opWRA}}) \)-write-list. Note that for every \( k > \min(P) \) with \( k \notin P \) and \( \text{typ}(L'(k)) = R \), we have \( \text{tid}(W(f(k))) = \text{tid}(L'(k)) \) = \( \text{tid}(L_{\tau}(f_{\tau}(k))) = \text{tid}(W_t(f_{\tau}(k))) \), and so \( G.hb \) must order the two write events, \( W(f(k)) \) and \( W_t(f_{\tau}(k)) \).

We define \( W' \) as follows:

\[
W' \triangleq \lambda k \in \{ 1 , \ldots , |L'| \}. \begin{cases} 
L'(k) & \text{typ}(L'(k)) = W \\
w_f & \text{typ}(L'(k)) = R \land k \in P \\
W(f(k)) & \text{typ}(L'(k)) = R \land k < \min(P) \\
\max_{G, hb}(W(f(k)), W_t(f_{\tau}(k))) & \text{otherwise}
\end{cases}
\]

It is easy to see that \( W' \) is a \( (G', L', \text{tid}_{\text{opWRA}}) \)-write-list. We show that \( W' \) is \( (G', \pi) \)-consistent. Let \( 1 \leq k \leq |L'| \) such that \( W'(k) \in E \). Let \( y = 1\circ(W'(k)), w_{x} = W(f(k)) \) and \( w_{t} = W_t(f_{\tau}(k)) \) (the latter is only defined if \( k > \min(P) \)). We prove that each of the conditions in Def. 6.6 holds:

\footnote{In WRA, the \( \text{mo} \)-component is immaterial and can be defined arbitrarily. We ignore this component in this proof.}
We prove that $W'(k) \notin \text{dom}(G'.hb|_{loc}) \setminus \{W\} \cup \{G'.hb^2; [E^\pi \cup \{W'(j) \mid 1 \leq j < k\}]\}$. Suppose otherwise. First, note that we cannot have $k \in P$, since $w$ is a maximal element in $G'.hb$.

Consider the two possible cases:

- $W'(k) \in \text{dom}(G'.hb|_{loc}) \setminus \{W\} \cup \{G'.hb^2; [E^\pi]\}$: The definition of $W'$ ensures that $\langle w_\pi, W'(k) \rangle \in G'.hb|_{loc}$, and so $w_\pi \in \text{dom}(G'.hb|_{loc}) \setminus \{W\} \cup \{G'.hb^2; [E^\pi]\}$. Since $W$ is $(G, \pi)$-consistent, we have that $w_\pi \notin \text{dom}(G'.hb|_{loc}) \setminus \{W\} \cup \{G'.hb^2; [E^\pi]\}$, and therefore it must be the case that $\pi = \tau$ and $\langle w_\pi, w \rangle \in G'.hb|_{loc}$. It follows that $w_\pi \in \text{dom}(G'.hb^2; [E^\pi])$. But, since $\pi = \tau$, we have $W(1) = L(1) = \pi(x) = \pi(\text{loc}(w_\pi))$, we obtain a contradiction to the fact that $W$ is $(G, \tau)$-consistent.

- $(W'(k), W'(j)) \in G'.hb|_{loc} \setminus \{W\} \cup \{G'.hb^2\}$ for some $1 \leq j < k$. Consider the two possible cases:

  - $W'(j) = w$: In this case we must have $k \geq \min(P)$, and so $W'(k) = \max_{G'.hb} \{w_\pi, w_\tau\}$. Hence, we have $\langle w_\tau, W'(k) \rangle \in G'.hb|_{loc}$, and so $\langle w_\tau, w \rangle \in \text{dom}(G'.hb|_{loc}) \setminus \{W\} \cup \{G'.hb^2; [E^\pi]\}$. Now, if $\langle w_\tau, w \rangle \in G'.hb|_{loc} \setminus \{W\} \cup \{G'.hb^2\}$, then we also have $w_\tau \in \text{dom}(G'.hb|_{loc}) \setminus \{W\} \cup \{G'.hb^2; [E^\pi]\}$, which contradicts the fact that $W_\tau$ is $(G, \tau)$-consistent. Therefore, we have $\langle w_\tau, w \rangle \in G'.hb|_{loc}$. It follows that $w_\tau \in \text{dom}(G'.hb^2; [E^\pi])$. But, since $W_\tau(1) = L_\tau(1) = \pi(x) = \pi(\text{loc}(w_\tau))$, we obtain a contradiction to the fact that $W_\tau$ is $(G, \tau)$-consistent.

  - $W'(j) \neq w$: In this case, we must have $\langle W'(k), W'(j) \rangle \in G'.hb|_{loc} \setminus \{W\} \cup \{G'.hb^2\}$. The definition of $W'$ ensures that $\langle w_\pi, W'(k) \rangle \in G'.hb|_{loc}$, and so $\langle w_\pi, W'(j) \rangle \in G'.hb|_{loc} \setminus \{W\} \cup \{G'.hb^2; [E^\pi]\}$. Now, since $W$ is $(G, \pi)$-consistent, we cannot have $W'(j) = W(f(j))$. Let $w'_\tau = W(f_\tau(j))$. Hence, $j \geq \min(P)$ and $W'(j) = w'_\tau$. It follows that $k \geq \min(P)$, and so $\langle w_\tau, W'(k) \rangle \in G'.hb|_{loc} \setminus \{W\} \cup \{G'.hb^2\}$. This contradicts the fact that $W_\tau$ is $(G, \tau)$-consistent.

Suppose by contradiction that there exists $i < k$ with $W'(i) = \pi(y)$ but $W'(k) \in \text{dom}(G'.hb^2; [E^\pi])$. Note that the definition of $W'$ ensures that $W'(i) = L'(i) = \pi(y)$, and since $W$ is a $(G, L, tid_{\loc})$-write-list, it follows that $W(f(i)) = \pi(y)$. Consider the two possible cases:

- $W'(k) = w$: In this case, we have $y = x$, $\pi = \tau$ and $i < \min(P)$. Since $P$ justifies a $(\tau, W(x, \nu_{\omega}))$-step, we cannot have $L'(i) = \pi(x)$.

- $W'(k) \neq w$: In this case, the definition of $W'$ ensures that $\langle w_\pi, W'(k) \rangle \in G'.hb|_{loc}$, and so $w_\pi \in \text{dom}(G'.hb^2; [E^\pi])$. Since $w_\pi \neq w$ (as $w_\pi \in G.E.$), it follows that $w_\pi \in \text{dom}(G'.hb^2; [E^\pi])$. Since $W(f(i)) = \pi(y)$, this contradicts the fact that $W$ is $(G, \pi)$-consistent.

Suppose by contradiction that there exists $j < i < k$ with $W'(i) = \pi(y)$ but $\langle W'(k), W'(j) \rangle \in G'.hb^2$. Note that the definition of $W'$ ensures that $W'(i) = L'(i) = \pi(y)$, and since $W$ is a $(G, L, tid_{\loc})$-write-list, it follows that $W(f(i)) = \pi(y)$. In addition, since $W_\tau$ is $(G, L_\tau, tid_{\loc})$-write-list, it follows that $W_\tau(f_\tau(i)) = \pi(y)$ 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 $P$ justifies a $(\tau, W(x, \nu_{\omega}))$-step, we cannot have $L'(i) = \pi(x)$.

- $W'(k) \neq w$ and $W'(j) = w$: In this case we must have $i, k > \min(P)$, and so $W'(k) = \max_{G'.hb} \{w_\pi, w_\tau\}$ and $W_\tau(f_\tau(i)) = \pi(y)$. Hence, we have $\langle w_\tau, W'(k) \rangle \in G'.hb|_{loc}$, and so $\langle w_\tau, w \rangle \in G'.hb^2$. Since $w_\pi \neq w$ (as $w_\pi \in G.E.$), it follows that $w_\tau \in \text{dom}(G'.hb^2; [E^\pi])$. Since $W_\tau(f_\tau(i)) = \pi(y)$, this contradicts the fact that $W_\tau$ is $(G, \tau)$-consistent.

- $W'(k) \neq w$ and $W'(j) \neq w$: In this case, we must have $\langle W'(k), W'(j) \rangle \in G'.hb^2$. Let $w'_{\tau} = W(f(j))$ and $w'_\tau = W_\tau(f_\tau(j))$ (the latter is only defined if $j > \min(P)$). Our construction ensures that one of the following holds:
What’s Decidable about Causally Consistent Memory Models? 1:37

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

- $l = R(x, v_R)$

By definition, since $B \xrightarrow{\tau,l}_{\text{opWRB}} B'$, there exists a read option $o$ with $1\text{oc}(o) = x$ and $\text{val}(o) = v_R$ such that $B(\tau) = o \cdot B'(\tau)$. For every $L \in B(\tau)$, let $W_L$ be a $\langle G, \pi \rangle$-consistent $(G, L, \text{tid}_{\text{wr}})$-write-list. Let $A = \{W_L(1) \mid L \in B(\tau)\}$. Since $B(\tau)$ is non-empty, we know that $A$ is not empty. Since each $W_L$ is a $\langle G, L, \text{tid}_{\text{wr}} \rangle$-write-list, we have that $\text{tid}(w) = \text{tid}(o)$ for every $w \in A$. Hence, $G, p$ totally orders $A$. Let $w = \text{min}_G, p A$ and let $L_{\text{min}} \in B(\tau)$ such that $w = W_{L_{\text{min}}}(1)$.

Let $r = \text{NextEvent}(G.E, \tau, I)$ and let $G'$ be the execution graph given by $G'.E = G.E \cup \{r\}$ and $G'.r.f = G.r.f \cup \{(w, r)\}$.

We show that $G \xrightarrow{\tau,l}_{\text{opWRB}} G'$. By definition, it suffices to show the following:
- $w \in G.W_x$ and $\text{val}_w(w) = v_R$: We have $w = W_{L_{\text{min}}}(1)$, and since $W_{L_{\text{min}}}$ is a $(G, L_{\text{min}}, \text{tid}_{\text{wr}})$-write-list, we have that $w \in G.W$, $1\text{oc}(w) = 1\text{oc}(W_{L_{\text{min}}}(1)) = 1\text{oc}(L_{\text{min}}(1)) = 1\text{oc}(o) = x$ and $\text{val}_w(w) = \text{val}_w(W_{L_{\text{min}}}(1)) = \text{val}(L_{\text{min}}(1)) = \text{val}(o) = v_R$.
- $w \notin \text{dom}(G.hb) |_{\text{loc}}^?: [W]; G.hb \sqsupset; [E^?])$: Since $W_{L_{\text{min}}}$ is $\langle G, \pi \rangle$-consistent and $w = W_{L_{\text{min}}}(1)$, we cannot have $w \in \text{dom}(G.hb) |_{\text{loc}}^?: [W]; G.hb \sqsupset; [E^?]$. It remains to show that $B' \not\subseteq G'$. We use the same $\text{tid}_{\text{wr}}$ mapping and show that for every $w \in B', \exists \pi \in \text{Tid}$ and $L' \in B'(\pi)$, there exists a $\langle G', \pi \rangle$-consistent $(G', L', \text{tid}_{\text{wr}})$-write-list. Let $w \in \text{Tid}$ and $L' \in B'(\pi)$. We define a $\langle G', \pi \rangle$-consistent $(G', L', \text{tid}_{\text{wr}})$-write-list. Consider two cases:
- $\pi \neq \tau$: By definition, since $B \xrightarrow{\tau,l}_{\text{opWRB}} B'$, we have $L' \in B(\pi)$. Let $W$ be a $\langle G, \pi \rangle$-consistent $(G, L', \text{tid}_{\text{wr}})$-write-list. It is easy to see that $W$ is also a $\langle G', L', \text{tid}_{\text{wr}} \rangle$-write-list. To see that $W$ is $\langle G', \pi \rangle$-consistent, note that if we have
  * $W(k) \in \text{dom}(G'.hb) |_{\text{loc}}^?: [W]; G'.hb \sqsupset; [E^? \cup \{W(j) \mid 1 \leq j < k\})]$,
  * $W(k) \in \text{dom}(G'.hb) \sqsupset; [E^?])$
  * or $\langle W(k), W(j) \rangle \in G'.hb \sqsupset$, 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, $L \in B(\tau)$. Let $W' = \lambda k \in \{1, \ldots, |L'|\}$. $W_L(1+k)$. It is easy to see that $W'$ is a $(G', L', \text{tid}_{\text{wr}})$-write-list. We show that $W'$ is $\langle G', \pi \rangle$-consistent. Let $1 \leq k \leq |W'|$ such that $W'(k) \in E$.
  * Suppose by contradiction that $W'(k) \in \text{dom}(G'.hb) |_{\text{loc}}^?: [W]; G'.hb \sqsupset; [E^? \cup \{W'(j) \mid 1 \leq j < k\})]$. If $W'(k) \in \text{dom}(G.hb) |_{\text{loc}}^?: [W]; G.hb \sqsupset; [E^? \cup \{W(j) \mid 1 \leq j < k\})]$, then $W_L(1+k) \in \text{dom}(G.hb) |_{\text{loc}}^?: [W]; G.hb \sqsupset; [E^? \cup \{W_L(1+j) \mid 1 \leq j < k\})]$, which contradicts the fact that $W_L$ is $\langle G, \tau \rangle$-consistent. Hence, we must have $\langle W'(k), w \rangle \in G.hb |_{\text{loc}}^?: [W]; G.hb \sqsupset$. Since $L(1) = o$, the definition of $w$ ensures that $\langle w, W_L(1) \rangle \in G.hb \sqsupset$. It follows that $\langle W_L(1+k), W_L(1) \rangle \in G.hb |_{\text{loc}}^?: [W]; G.hb \sqsupset$, which again contradicts the fact that $W_L$ is $\langle G, \tau \rangle$-consistent.
  * Suppose by contradiction that there exists $i < k$ with $W'(i) = 0\text{oc}(\text{loc}(W'(k)))$ (and so, $W_L(1+i) = 0\text{oc}(\text{loc}(W_L(1+k)))$ but $W'(k) \in \text{dom}(G'.hb) \sqsupset; [E^?]$. If $W'(k) \in \text{dom}(G.hb) \sqsupset; [E^?])$, then $W_L(1+k) \in \text{dom}(G.hb) \sqsupset; [E^?]$, which contradicts the fact that $W_L$ is $\langle G, \tau \rangle$-consistent. Hence, we must have $\langle W'(k), w \rangle \in G.hb \sqsupset$. Since $L(1) = o$, the definition of $w$ ensures that $\langle w, W_L(1) \rangle \in G.hb \sqsupset$. It follows that $\langle W_L(1+k), W_L(1) \rangle \in G.hb$ while $W_L(1+i) = 0\text{oc}(\text{loc}(W_L(1+k)))$. This contradicts the fact that $W_L$ is $\langle G, \tau \rangle$-consistent.
• \( L = \text{RMW}(x, v_R, v_H) \):

First, since \( B \xrightarrow{\tau, l} \text{loWRA} \) \( B' \) provide us with the following:

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

2. By Prop. 6.9, there exists an index choice \( \mathcal{P} \) for \( B' \) that justifies a \( \langle \tau, W(x, v_H) \rangle \)-step, such that \( \text{src}(B', \tau, \mathcal{P})(\pi) \subseteq B(\pi) \) for every \( \pi \in \text{Tid} \setminus \{ \tau \} \) and \( o \cdot \text{src}(B', \tau, \mathcal{P})(\tau) \subseteq B(\tau) \).

For every \( L \in B(\tau) \), let \( W_L \) be a \( (G, \tau) \)-consistent \( (G, L, \text{tid}_{\text{RMW}}) \)-write-list. Let \( A = \{ W_L(1) | L \in B(\tau) \} \).

Since \( B(\tau) \) is non-empty, we know that \( A \) is not empty. Since each \( W_L \) is a \( (G, L, \text{tid}_{\text{RMW}}) \)-write-list, we have that \( \text{tid}(w) = \text{tid}(o) \) for every \( w \in A \). Hence, \( G, \text{po} \) totally orders \( A \).

Let \( w = \min_{G, \text{po}} A \) and let \( L_{\text{min}} \in B(\tau) \) such that \( w = W_{L_{\text{min}}}(1) \). Let \( e = \text{NextEvent}(G, \text{E}, \tau, I) \) and let \( G' \) be the execution graph given by \( G'.E = G.E \cup \{ e \} \) and \( G'.rf = G.rf \cup \{ (w, e) \} \).

Note that \( w = W_{L_{\text{min}}}(1) \), and since \( W_{L_{\text{min}}} \) is a \( (G, L_{\text{min}}, \text{tid}_{\text{RMW}}) \)-write-list, we have that:

- \( w \in G.W \).
- \( \text{loc}(w) = \text{loc}(W_{L_{\text{min}}}(1)) = \text{loc}(L_{\text{min}}(1)) = \text{loc}(x) = x \).
- \( \text{val}(w) = \text{val}(W_{L_{\text{min}}}(1)) = \text{val}(L_{\text{min}}(1)) = \text{val}(o) = v_R \).
- \( \text{rmw-tid}(w) = \text{rmw-tid}(W_{L_{\text{min}}}(1)) = \text{rmw-tid}(L_{\text{min}}(1)) = \tau \).

Then, to show that \( G \xrightarrow{\tau, l} \text{opWRA} G' \), it suffices, by definition, to show the following:

- \( w \notin \text{dom}(G.hb)_{\text{loc}} : [W] ; G.hb^2 ; [E^f] \): Since \( W_{L_{\text{min}}} \) is a \( (G, \tau) \)-consistent and \( w = W_{L_{\text{min}}}(1) \), we cannot have \( w \in \text{dom}(G.hb)_{\text{loc}} : [W] ; G.hb^2 ; [E^f] \).

- \( w \notin \text{dom}(G.rf) ; [\text{RMW}] \): Suppose otherwise, and let \( e' \in \text{RMW} \) such that \( \langle w, e' \rangle \in G.rf \).

Then, since \( \text{tid}_{\text{RMW}}(w) = \tau \), the second condition in Def. 6.7 ensures that \( \text{tid}(e) = \tau \). Hence, \( w \in \text{dom}(G.rf) ; [\text{RMW} \cap E^f] \subseteq \text{dom}(G.hb)_{\text{loc}} ; [W] ; G.hb^2 ; [E^f] \), which contradicts the previous item.

It remains to show that \( B' \not\preceq G' \). Since \( \mathcal{P} \) justifies a \( \langle \tau, W(x, v_H) \rangle \)-step, there exists \( \pi_{\text{RMW}} \in \text{Tid} \) such that \( L'(k) = \langle \tau, x, v_H, \pi_{\text{RMW}} \rangle \) for every \( \pi \in \text{Tid} \), \( L' \in B'(\pi) \) and \( k \in \mathcal{P}(\pi, L') \).

Let \( \text{tid}_{\text{RMW}} = \text{tid}_{\text{RMW}}[w \mapsto \pi_{\text{RMW}}] \). Since \( e \notin G'.rf \), we vacuously have \( \text{tid}(e') = \text{tid}_{\text{RMW}}(e) \) for every \( \langle e, e' \rangle \in G'.rf ; [\text{RMW}] \). In addition, we have \( \text{tid}(e) = \tau = \text{tid}_{\text{RMW}}(w) = \text{tid}_{\text{RMW}}(w) \).

Since \( w \) is the unique event such that \( \langle w, e' \rangle \in G'.rf ; [\text{RMW}] \), we have \( \text{tid}(e') = \text{tid}_{\text{RMW}}(w) \).

We show that for every \( \pi \in \text{Tid} \) and \( L' \in B'(\pi) \), there exists a \( (G', \pi) \)-consistent \( (G', L', \text{tid}_{\text{RMW}}) \)-write-list. Let \( \pi \in \text{Tid} \) and \( L' \in B'(\pi) \). We construct a \( (G', \pi) \)-consistent \( (G', L', \text{tid}_{\text{RMW}}) \)-write-list \( W' \). Let \( P \not\preceq \mathcal{P}(\pi, L') \) and (the last two are only defined if \( P \neq \emptyset \)):

\[
L \triangleq \begin{cases} 
L' \setminus P & \pi \neq \tau \\
(\cdot)_{\cdot} \cdot_{\cdot} \cdot(f \cdot \cdot \cdot) & \pi = \tau
\end{cases}
\]

\[
L_{\pi} \triangleq \begin{cases} 
\cdot_{\cdot} \cdot_{\cdot} \cdot(f \cdot \cdot \cdot) & \pi \neq \tau \\
\cdot_{\cdot} \cdot_{\cdot} \cdot_{\cdot} \cdot(f \cdot \cdot \cdot) & \pi = \tau
\end{cases}
\]

Then, by definition, we have \( L \in B(\pi) \) and \( L_{\pi} \in B(\tau) \). Let \( W \) be a \((G, \pi)\)-consistent \((G, L, \text{tid}_{\text{RMW}})\)-write-list, and \( W_{L_{\pi}} \) be a \((G, \tau)\)-consistent \((G, L_{\pi}, \text{tid}_{\text{RMW}})\)-write-list. Note that for every \( k > \min(P) \) with \( k \notin P \) and \( \text{typ}(L'(k)) = \mathbb{R} \), we have \( \text{tid}(W(f(k))) = \text{tid}(L(f(k))) = \text{tid}(L_{\pi}(f_{\pi}(k))) = \text{tid}(W_{L_{\pi}}(f_{\pi}(k))) \), and so \( G.hb \) must order the two write events, \( W(f(k)) \) and
\[ W_r(f_r(k)). \] We define \( W' \) as follows:

\[
\begin{align*}
W' &\triangleq \lambda k \in \{1, ..., |L'|\}. \\
&\begin{cases}
L'(k) & \text{typ}(L'(k)) = W \\
e & \text{typ}(L'(k)) = R \land k \in P \\
W(f(k)) & \text{typ}(L'(k)) = R \land k < \min(P) \\
\max_{G, \cdot hb}\{W(f(k)), W_r(f_r(k))\} & \text{otherwise}
\end{cases}
\]

It is easy to see that \( W' \) is a \( (G', L', \cdot tid_{\cdot hb}) \)-write-list. We show that \( W' \) is \( (G', \pi) \)-consistent. Let \( 1 \leq k \leq |L'| \) such that \( W'(k) \in E \). Let \( y = 1_{\cdot oc}(W'(k)), w_\pi = W(f(k)) \) and \( w_r = W_r(f_r(k)) \) (the latter is only defined if \( k > \min(P) \)). We prove that each of the conditions in Def. 6.6 holds:

- We prove that \( W'(k) \notin \cdot dom(G'.\cdot hb_{\cdot loc}; [W]; G'.\cdot hb^7; [E^\pi] \cup \{W'(j) \ | \ 1 \leq j < k\}) \). Suppose otherwise. First, note that we cannot have \( k \in P \), since \( e \) is a maximal element in \( G'.\cdot hb \). Consider the two possible cases:

  - \( W'(k) \in \cdot dom(G'.\cdot hb_{\cdot loc}; [W]; G'.\cdot hb^7; [E^\pi]) \): The definition of \( W' \) ensures that we have \( (\cdot w_\pi, W'(k)) \in G'.\cdot hb_{\cdot loc} \), and so \( w_\pi \in \cdot dom(G'.\cdot hb_{\cdot loc}; [W]; G'.\cdot hb^7; [E^\pi]) \). Since \( W \) is \( (G, \pi) \)-consistent, we have that \( w_\pi \notin \cdot dom(G.\cdot hb_{\cdot loc}; [W]; G.\cdot hb^7; [E^\pi]) \), and therefore it must be the case that \( (\cdot w_\pi, e) \in G.\cdot hb_{\cdot loc}; [W]; G.\cdot hb^7; G'.\cdot rf^7 \) and \( \pi = \tau \). Now, if \( w_\pi \in \cdot dom(G.\cdot hb^7; [E^\pi]) \), then since \( \pi = \tau \), we have \( W(2) = L(2) = 0_\cdot wb(x) = 0_\cdot wb(1_{\cdot oc}(w_\pi)) \), and we obtain a contradiction to the fact that \( W \) is \( (G, \tau) \)-consistent. Otherwise, we have \( (\cdot w_\pi, w) \in G'.\cdot hb_{\cdot loc}; [W]; G.\cdot hb \). Since \( \pi = \tau \), we have \( L(1) = o \), and the definition of \( w \) ensures that \( (\cdot w, W(1)) \in G.\cdot po^7 \). It follows that \( (\cdot w_\pi, W(1)) \in G.\cdot hb_{\cdot loc}; [W]; G.\cdot hb^7 \), which again contradicts the fact that \( W \) is \( (G, \tau) \)-consistent.

  - \( \langle W'(k), W'(j) \rangle \in G'.\cdot hb_{\cdot loc}; [W]; G'.\cdot hb^7 \) for some \( 1 \leq j < k \). Consider the two possible cases:

    * \( W'(j) = e \): In this case we must have \( k \geq \min(P) \), and so \( W'(k) = \max_{G, \cdot hb}\{w_\pi, w_r\} \). Hence, we have \( (\cdot w_\pi, W'(k)) \in G.\cdot hb_{\cdot loc}^7 \). There are four possibilities:

      - \( W'(k) = w \): In this case we have \( (\cdot w_r, w) \in G.\cdot hb_{\cdot loc}^7 \). Since \( L_r(1) = o \), the definition of \( w \) ensures that \( (\cdot w, W_r(1)) \in G.\cdot po^7 \). Hence, \( (\cdot w_r, W_r(1)) \in G.\cdot hb^7 \). Since \( L_r(2) = 0_\cdot wb(x) = 0_\cdot wb(1_{\cdot oc}(W_r(f_r(k)))) \), we obtain a contradiction to the fact that \( W_r \) is \( (G, \tau) \)-consistent.

      - \( (\cdot W'(k), w) \in G'.\cdot hb_{\cdot loc}; [W]; G'.\cdot hb^7 \): This contradicts the \( (G, \tau) \)-consistency of \( W_r \), as \( (\cdot w, W_r(1)) \in G.\cdot po^7 \) and \( (\cdot w_r, W'(k)) \in G'.\cdot hb_{\cdot loc}^7 \), implying that \( (\cdot w_r, W_r(1)) \in G.\cdot hb_{\cdot loc}; [W]; G.\cdot hb^7 \).

      - \( (\cdot W'(k), e) \in G'.\cdot hb_{\cdot loc}; [W]; G'.\cdot hb^7; G'.\cdot po \): This also contradicts the \( (G, \tau) \)-consistency of \( W_r \), as we get that \( \cdot w_r \in \cdot dom(G.\cdot hb_{\cdot loc}; [W]; G.\cdot hb^7; [E^\pi]) \).

      - \( y = x \) and \( (\cdot W'(k), e) \in G'.\cdot hb; G'.\cdot po \): In this case we have \( (\cdot w_r, e) \in G'.\cdot hb_{\cdot loc}^7; G'.\cdot hb \). Since \( G.\cdot po \), and so \( w_r \in \cdot dom(G.\cdot hb^7; [E^\pi]) \). But, since \( W_r(2) = L_r(2) = 0_\cdot wb(x) = 0_\cdot wb(1_{\cdot oc}(w_\pi)) \), we obtain a contradiction to the fact that \( W_r \) is \( (G, \tau) \)-consistent.

    * \( W'(j) \neq e \): In this case, we must have \( \langle W'(k), W'(j) \rangle \in G.\cdot hb_{\cdot loc}; [W]; G.\cdot hb^7 \). The definition of \( W' \) ensures that \( (\cdot w_\pi, W'(k)) \in G.\cdot hb_{\cdot loc}^7 \), and so \( (\cdot w_\pi, W'(j)) \in G.\cdot hb_{\cdot loc}; [W]; G.\cdot hb^7 \). Now, since \( W \) is \( (G, \pi) \)-consistent, we cannot have \( W'(j) = W(f(j)) \). Let \( w'_r = W_r(f_r(j)) \). Hence, \( j \geq \min(P) \) and \( W'(j) = w'_r \). It follows that \( k \geq \min(P) \), and so \( (\cdot w_r, W'(k)) \in G.\cdot hb_{\cdot loc}^7 \). Hence, we have \( (\cdot w_r, w'_r) \in G.\cdot hb_{\cdot loc}; [W]; G.\cdot hb^7 \). This contradicts the fact that \( W_r \) is \( (G, \tau) \)-consistent.

Suppose by contradiction that there exists \( i < k \) with \( W'(i) = 0_\cdot wb(y) \) but \( W'(k) \notin \cdot dom(G'.\cdot hb^7; [E^\pi]) \). Note that the definition of \( W' \) ensures that \( W'(i) = L'(i) = 0_\cdot wb(y) \), and since \( W \) is a \( (G, L, \cdot tid_{\cdot hb}) \)-write-list, it follows that \( W(f(i)) = 0_\cdot wb(y) \). 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 $\mathcal{P}$ justifies a $(\tau, W(x, v_\tau))$-step, we cannot have $L'(i) = O(y(x)).$
− $W'(k) \neq e$: In this case, the definition of $W'$ ensures that $\langle w_\pi, W'(k) \rangle \in G'.hb\mid_{loc}^?$, and so $w_\pi \in dom(G'.hb\mid_{E})$. Since $w_\pi \neq e$ (as $w_{\pi} \in G.E$), it follows that $w_\pi \in dom(G.hb\mid_{E})$. Since $W(f(i)) = O(y), this contradicts the fact that $W$ is $(G, \pi)$-consistent.

• Suppose by contradiction that there exists $j < i < k$ with $W'(i) = O(y(y))$ but $(W'(k), W'(j)) \in G'.hb\mid_{loc}^?$. Note that the definition of $W'$ ensures that $W'(i) = L'(i) = O(y), and since $W$ is a $(G, L, tid_{\text{trans}})$-write-list, it follows that $W(f(i)) = O(y). In addition, since $W_2$ is $(G, L_\pi, tid_{\text{trans}})$-write-list, it follows that $W_2(f_\pi(i)) = O(y)$ 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 $\mathcal{P}$ justifies a $(\tau, W(x, v_\tau))$-step, we cannot have $L'(i) = O(y(x)).$
− $W'(k) \neq e$ and $W'(j) = e$: In this case we must have $i, k > \min(P)$, and so $W'(k) = \max_{G.hb}\{w_\pi, w_\tau\}$ and $W_2(f_\pi(i)) = O(y). Hence, we have $(w_\tau, W'(k)) \in G.hb\mid_{loc}^?$, and so $(w_\tau, e) \in G'.hb\mid_{loc}^?$. Since $w_\tau \neq e$ (as $w_\tau \in G.E$), it follows that $w_\tau \in dom(G.hb\mid_{E}^?)$. Since $W_2(f_\pi(i)) = O(y), this contradicts the fact that $W_2$ is $(G, \pi)$-consistent.

− $W'(k) \neq e$ and $W'(j) \neq e$: In this case, we must have $(W'(k), W'(j)) \in G.hb\mid_{loc}^?$. Let $w_{\pi} = W(f(j)) and w_\tau = W_2(f_\pi(j)) (the latter is only defined if $j > \min(P))$. Our construction ensures that one of the following holds:

* $W'(j) = w_\tau$: Since $W'(k) \neq e$, the definition of $W'$ ensures that $\langle w_\pi, W'(k) \rangle \in G'.hb\mid_{loc}^?$, and so $\langle w_\pi, w_\tau \rangle \in G.hb\mid_{loc}^?$. This contradicts the fact that $W$ is $(G, \pi)$-consistent.

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

Finally, suppose that $B \not\parallel G$ and $B \not\parallel_{\text{loWRA}} B'$ (using the lower step). Let $tid_{\text{trans}}: W \rightarrow \text{Tid}$ that satisfies the conditions of Def. 6.7. To show that $B' \not\parallel G$, we use the same $tid_{\text{trans}}$ mapping and show that for every $\tau \in \text{Tid}$ and $L' \not\in B'(\tau)$, there exists a $(G', \pi)$-consistent $(G', L', tid_{\text{trans}})$-write-list.

Let $\tau \in \text{Tid}$ and $L' \not\in B'(\tau)$. We define a $(G, \tau)$-consistent $(G, L', tid_{\text{trans}})$-write-list $W'$. By definition, since $B \not\parallel_{\text{loWRA}} B'$, there exists $L \in B(\tau)$ such that $L' \not\subseteq L$. Let $W$ be a $(G, \tau)$-consistent $(G, L, tid_{\text{trans}})$-write-list, and let $f : \{1, ..., |L'|\} \rightarrow \mathbb{N}$ be an increasing function such that $L'(k) = L(f(k)) for every $k \in dom(f)$. It is easy to see that $W' = \lambda k \in \{1, ..., |L'|\}. W(f(k)) is a $(G, L', tid_{\text{trans}})$-write-list. To see that $W'$ is $(G, \tau)$-consistent, note that:

* $W'(k) \in dom(G.hb\mid_{loc}^?; [W]; G.hb\mid_{E}^?; \{W'(j) | 1 \leq j < k\}) implies W'(k) \in dom(G.hb\mid_{loc}^?; [W]; G.hb\mid_{E}^?; \{W(f(j)) | 1 \leq j < k\})$.

* $W'(k) \in dom(G.hb\mid_{E}^?; \{E\}) implies W'(k) \in dom(G.hb\mid_{E}^?; \{E\})$.

* $\langle w_\pi, W'(k) \rangle \in G.hb\mid_{loc}^? implies \langle w_\pi, W'(k) \rangle \in G.hb\mid_{loc}^?$.

Therefore, the $(G, \tau)$-consistency of $W'$ directly follows from the $(G, \pi)$-consistency of $W$. □

**Proof of (2) Direction:** Every trace of opWRA is a trace of loWRA. We show that $\gamma^{-1}$ constitutes a backward simulation from opWRA to loWRA (for the definition of backward simulation, see Footnote 3 on Page 31). The two first requirements of a backward simulation clearly hold for $\gamma$: (1) $\gamma^{-1}$ is total, as for every state $G$ of opWRA, we have $(\lambda g \in \text{Tid}. \{e\}) \gamma G$. (2) Consider a state $B$ of loWRA, such that $B \not\parallel G_0$. By the definition of $\gamma$, it should be possible to link every read option of $B$ to some write event of $G_0$. Since there are no write events in $G_0$, there cannot be read options in $B$, implying that $B \not\in \text{loWRA}_0$. Q.E.D.
We move to the third requirement. Suppose that $G \xrightarrow{\tau,l} \text{opWRA} G'$ and $B' \triangleright G'$, witnessed by a function $\tau_{d_{\text{op}}}: W \rightarrow \text{Tid}$. We construct a state $B$ such that $B \xrightarrow{\tau,l} \text{ioWRA} B'$ and $B \triangleright G$. Consider the possible cases:

- $l = W(x, v_w)$:

Let $w = \text{NextEvent}(G.E, \tau, l)$. Since $G \xrightarrow{\tau,l} \text{opWRA} G'$, we have $G.E = G.E \cup \{w\}$ and $G'.rf = G.rf$. Let $P$ be the index choice for $B'$ that assigns the set of “new” positions in $B'$:

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

Then, we define

$$B \triangleq \lambda \pi \in \text{Tid}, \begin{cases} 0_w(x) \cdot \text{src}(B', \tau, P)(\tau) & \pi = \tau \\ \text{src}(B', \tau, P)(\pi) & \pi \neq \tau \end{cases}$$

By Prop. 6.9, to show that $B \xrightarrow{\tau,l} \text{ioWRA} B'$, it suffices to prove that $P$ justifies a $(\tau, W(x, v_w))$-step. Let $\tau_{d_{\text{op}}} = \tau_{d_{\text{op}}}(w)$. Thus, we show that the following hold for every $\pi \in \text{Tid}$ and $L' \in B'(\pi)$, where $P = \langle \tau, \tau \rangle$ and $W' = W'_{(\tau,L')}:

- Let $k \in P$. Then, we have $W'(k) = w$, and thus $L'(k) = \langle \tau, x, v_w, \tau_{d_{\text{op}}} \rangle$.

- 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 $L'(k) \neq 0_w(x)$. 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'.hb'.

- Since $W'$ is $(G', \tau)$-consistent (by C3), we cannot have $W'(k) = 0_w(1\text{oc}(W'(p_2)))$, and so $L'(k) \neq 0_w(x)$.

- Suppose that $\pi = \tau$ and $k \in \{1, \ldots, |L'|\} \setminus P$ such that $k < p$ for some $p \in P$. We show that $L'(k) \neq 0_w(x)$. Since $p \in P$, we have $W'(p) = w$, and so $W'(p) \in \text{dom}(G'.hb'; [E'])$. Since $W'$ is $(G', \tau)$-consistent (by C2), we cannot have $W'(k) = 0_w(1\text{oc}(W'(p)))$, and so $L'(k) \neq 0_w(x)$.

Next, we prove that $B \triangleright G$, by showing that for every $\pi \in \text{Tid}$ and $L \in B(\pi)$, there exists a $(G, \pi)$-consistent $(G, L, \tau_{d_{\text{op}}})$-write-list. (Since $G.rf \subseteq G'.rf$, the second condition of $\triangleright$ (Def. 6.7), namely that for every $(w, e) \in G.rf; [\text{RMW}]$, we have $\tau_{d}(e) = \tau_{d}(w)$, trivially holds.) Let $\pi \in \text{Tid}$ and $L \in B(\pi)$. Following the construction of $B$, one of the following holds:

- $\pi \neq \tau$ and $L = L' \setminus P(\pi, L')$ for some $L' \in B'(\pi)$. Let $P = P(\pi, L')$, $W' = W'_{(\pi,L')}$ and $f = \text{Map}_{(L', P)}^{-1}$. We define $W \triangleq \lambda k \in \{1, \ldots, |L|\}$. $W'(f(k))$. Using the fact that $W'$ is a $(G', L', \tau_{d_{\text{op}}})$-write-list, it is easy to see that $W$ is an $(G, L, \tau_{d_{\text{op}}})$-write-list.

- $\pi = \tau$ and $L = 0_w(x) \cdot (L' \setminus P(\pi, L'))$ for some $L' \in B(\tau)$. Let $P = P(\pi, L')$, $W' = W'_{(\pi,L')}$ and $f = \lambda k \in \{2, \ldots, |L|\}$. $\text{Map}_{(L', P)}^{-1}(k-1)$. We define:

$$W \triangleq \lambda k \in \{1, \ldots, |L|\}. \begin{cases} 0_w(x) & k = 1 \\ W'(f(k)) & k > 1 \end{cases}$$

By the fact that $W'$ is a $(G', L', \tau_{d_{\text{op}}})$-write-list, we get that $W$ is a $(G, L, \tau_{d_{\text{op}}})$-write-list. It remains to show that it is $(G, \tau)$-consistent. Conditions C1 and C3 in Def. 6.6 follow directly from the $(G', \tau)$-consistency of $W'$. Condition C2, however, deserves more attention, as we added $0_w(x)$ at the start of the list. Assume toward contradiction some $k$, such that
We define \( l = \max_{G', p_0} G' \cdot E' \) and \( \l o c(W(k)) = \l o c(w) \), have \( \langle W'(f(k)), w \rangle \in G'.h b_{1 \l o c}[W] ; G'.h b^2 \), contradicting (C1) in the \((G', \tau)\)-consistency of \( W' \).

\(- \pi = \tau \) and \( L = \emptyset(x) \cdot (L' \setminus |\mathcal{P}(\eta, L')|) \) for some \( \eta \in T i d \) and \( L' \in B'(\eta) \).

Let \( \mathcal{P}(\eta, L') \), \( m = \min(P) \), \( W' = W'_{\eta, L'} \) and \( f = \lambda k \in \{ 2, ..., |L'| \} \).

We define:

\[
W \triangleq \begin{cases} 
0_{w}(x) & \text{if } k = 1 \\
W'(f(k)) & \text{if } k > 1
\end{cases}
\]

By the fact that \( W' \) is a \((G', L', t i d_{\eta})\)-write-list, we get that \( W \) is a \((G, L, t i d_{\eta})\)-write-list.

It remains to show that it is \((G, \tau)\)-consistent. Condition C3 follows directly from the \((G', \eta)\)-consistency of \( W' \).

We prove the other two conditions:

C1) The existence of some \( k \), such that \( W(k) \subseteq dom(G.h b_{1 \l o c}[W] ; G.h b^2 ; \{ (W(j) | 1 \leq j < k) \}) \)

directly contradicts the same condition in the \((G', \eta)\)-consistency of \( W' \). Now, assume toward contradiction some \( k \), such that \( W(k) \subseteq dom(G.h b_{1 \l o c}[W] ; G.h b^2 ; \{ (W(j) | 1 \leq j < k) \}) \).

Then, since \( W'(f(k)) = W(k) \), \( f(k) > m \), \( W'(m) = w \) and \( w = \max_{G', p_0} G' \cdot E' \), we have \( W'(f(k)) \subseteq dom(G'.h b_{1 \l o c}[W] ; G'.h b^2 ; \{ (W'(j) | 1 \leq j < f(k)) \}) \), contradicting (C1) in the \((G', \eta)\)-consistency of \( W' \).

C2) Assume toward contradiction the existence of some \( i < k \), such that \( W(i) = 0_{w}(1 \l o c(W(k))) \)

and \( W(k) \subseteq dom(G.h b_{1 \l o c}[W] ; G.h b^2 ; \{ (W(j) | 1 \leq j < k) \}) \).

First if \( i = 1 \), then \( \l o c(W(k)) = \emptyset \), and as above, since \( W'(f(k)) = W(k) \), \( f(k) > m \), \( W'(m) = w \) and \( w = \max_{G', p_0} G' \cdot E' \), we have \( W'(f(k)) \subseteq dom(G'.h b_{1 \l o c}[W] ; G'.h b^2 ; \{ (W'(j) | 1 \leq j < f(k)) \}) \), contradicting (C1) in the \((G', \eta)\)-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 \( w = \max_{G', p_0} G' \cdot E' \), we have \( W'(f(k)), W'(m) \subseteq G'.h b^2 \), contradicting (C3) in the \((G', \eta)\)-consistency of \( W' \).

\* \( l = R(x, \nu_R) \):

Let \( r = \text{NextEvent}(G.E, \tau, L) \). Since \( \tau_{L_{\text{opWRA}}} G' \), we have that \( G'.E \subseteq G.E \cup \{ r \} \) and \( G'.r f = G.r f \cup \{ (w, r) \} \) for some write event \( w \in G.W_x \) such that \( \text{val}_w(w) = \nu_R \) and \( w \not\in dom(G.h b_{1 \l o c}[W] ; G.h b^2 ; \{ E \}) \).

Let \( o = \langle \text{tid}(w), x, \nu_R, t i d_{\eta}(w) \rangle \). We define \( \mathcal{B} \) by:

\[
\mathcal{B} \triangleq \lambda \pi \in T i d. \begin{cases} 
o \cdot B'(\tau) & \pi = \tau \\
B'(\pi) & \pi \neq \tau
\end{cases}
\]

By definition, we have \( \mathcal{B} \tau_{L_{\text{opWRA}}} \mathcal{B}' \). We show that \( \mathcal{B} \gamma G \). Note that the second condition of \( \gamma \) (Def. 6.7) trivially holds, and we need to show that for every \( \pi \in T i d \) and \( L \in B(\pi) \), there exists a \((G, \pi)\)-consistent \((G, L, t i d_{\eta})\)-write-list.

For \( \pi \neq \tau \) and \( L \in B(\pi) \), observe that \( L \in B'(\pi) \), and since \( G.h b \subseteq G'.h b \), we have that \( \pi_{\langle \pi, L' \rangle} \) is also a \((G, \pi)\)-consistent \((G, L, t i d_{\eta})\)-write-list.

Consider a read-option list \( L \in B(\tau) \). Let \( L' \in B'(\tau) \) 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 \((G', L', t i d_{\eta})\)-write-list, we get that \( W \) is a \((G, L, t i d_{\eta})\)-write-list. It remains to show that it is \((G, \tau)\)-consistent. Given the \((G', \tau)\)-consistency of \( W' \), for C1, we only need to show that \( w \not\in dom(G.h b_{1 \l o c}[W] ; G.h b^2 ; \{ E \}) \), 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 \((G', \tau)\)-consistency of \( W' \). For C3, given the \((G', \tau)\)-consistency of \( W' \), it suffices to handle the case that \( j = 1 \).

Thus, assume toward contradiction some \( 1 < k \leq |L| \) and \( 1 < i < k \), such that \( W(i) = 0_{w}(1 \l o c(W(k))) \) and \( \langle W(k), w \rangle \in G.h b^2 \). Then, since \( r \in G'.E \) and \( \langle w, r \rangle \in G'.r f \), we get that
What’s Decidable about Causally Consistent Memory Models? 1:43

- Due to adding $\pi$ of $\text{tid}(w) = \ell \in \text{e}$ holds:

$$\langle \text{tid}(w), \text{src}(\pi, \tau, P)(\pi) \rangle \neq \tau \quad \text{where o is the read option given by } o \triangleq (\text{tid}(w), x, v_r, \tau).$$

Using Prop. 6.9, to show that $B \xrightarrow{\tau,l} \text{opWRA} B'$, it suffices to prove that $P$ justifies a $\langle \tau, w(x, v_r) \rangle$-step. This is done as in the write case, together with the following observation: Since $e \in G', E$, $e \in \text{RMW}$ and $\langle w, e \rangle \in G'.rf$, the fact that $\text{tid}_{\text{RMW}}$ witnesses $B' \not\subseteq G'$, guarantees that $\text{tid}_{\text{RMW}}(w) = \tau$. It remains to show that $B \not\subseteq G$. We show that for every $\pi \in \text{Tid}$ and $L \subseteq B(\pi)$, there exists a $\langle G, \pi \rangle$-consistent $(G, L, \text{tid}_{\text{RMW}})$-write-list. (The second condition of $\not\subseteq$ (Def. 6.7) trivially holds.) Let $\pi \in \text{Tid}$ and $L \subseteq B(\pi)$.

- $\pi \neq \tau$ and $L = L' \setminus P(\pi, L')$ for some $L' \subseteq B'(\pi)$. This case is exactly the same as the analogous case in the write step.

- $\pi = \tau$ and $L = o \cdot 0w(x) \cdot (L' \setminus P(\pi, L'))$ for some $L' \subseteq B'(\pi)$. Let $P = P(\tau, L'$), $W' = W'$ and $f = lk \in \{3, \ldots, |L|\}$. Map $\text{Map}_{L', P}(k - 2)$. We define:

$$W \triangleq \begin{cases} w & k = 1 \\ 0w(x) & k = 2 \\ W'(f(k)) & k > 2 \end{cases}$$

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

\begin{enumerate}
\item[C1] Observe first that $W(1) = w$ and $w \notin \text{dom}(G.\text{hb}\text{\textunderscore}1\text{oc} ; [W ; G.\text{hb}\text{\textunderscore}2 ; [E^\tau]])$ 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 $W(k) \notin \text{dom}(G.\text{hb}\text{\textunderscore}1\text{oc} ; [W ; G.\text{hb}\text{\textunderscore}2 ; [E^\tau \cup \{W(j) \mid 3 \leq j < k\}])$. It is left to show that $\langle W(k), w \rangle \notin G.\text{hb}\text{\textunderscore}1\text{oc} ; [W ; G.\text{hb}\text{\textunderscore}2 ; [E^\tau \cup \{W(j) \mid 3 \leq j < k\})]$. Indeed, were it not the case, since $\langle w, e \rangle \in G'.rf$ and $e \in E^\tau$, we would have had $W'(f(k)) \in \text{dom}(G'.\text{hb}\text{\textunderscore}1\text{oc} ; [W ; G.\text{hb}\text{\textunderscore}2 ; [E^\tau])$, contradicting (C1) in the $\langle G', \tau \rangle$-consistency of $W'$.\end{enumerate}
$W(k) = w, \langle w, e \rangle \in G'.rf$ and $e \in W$, which contradicts (C1) in the $\langle G', \tau \rangle$-consistency of $W'$. Then, observe that $\langle W(k), w \rangle \notin G'.hb$, as $w \in W$, and we showed while handling C1 that $\langle W(k), w \rangle \notin G'.hb|_{\text{loc}}; [W]; G'.hb^2$.

$\pi = \tau$ and $L = o \cdot 0_w(x) \cdot (L' \setminus \mathcal{P}(\eta, L'))$ for some $\eta \in \text{Tid}$ and $L' \in B'(\eta)$.

Let $P = \mathcal{P}(\eta, L'), W' = W'_{(\eta, L')}, m = \min(P)$ and $f = \lambda k \in \{3, \ldots, |L|\}$. $\text{MMap}_{(L', P)}^{-1}(k - 2)$. We define:

$$W \triangleq \lambda k \in \{1, \ldots, |L|\} \cdot \begin{cases} w & k = 1 \\ 0_w(x) & k = 2 \\ W'(f(k)) & k > 2 \end{cases}$$

By the fact that $W'$ is a $\langle G', L', \text{tid}_{\text{fn}} \rangle$-write-list, we get that $W$ is a $\langle G, L, \text{tid}_{\text{fn}} \rangle$-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 $W'_{(\eta, L')}$ rather than of $W'_{(\tau, L')}$. Hence, we should make sure that for every $2 < k \leq |L|$, we still have $W(k) \notin \text{dom}(G'.hb|_{\text{loc}}; [W]; G'.hb^2; \{E' \cup \{w\}\}).$ Assume first toward contradiction some $k$, such that $W(k) \in \text{dom}(G'.hb|_{\text{loc}}; [W]; G'.hb^2; \{E'\})$. Then, since $W'(f(k)) = W(k), f(k) > m, W'(m) = e$ and $e = \max_{G'.po} G'.E'$, we have $W'(f(k)) \in \text{dom}(G'.hb|_{\text{loc}}; [W]; G'.hb^2; \{\{W'(j) | 1 \leq j < f(k)\}\}),$ contradicting (C1 in) the $\langle G', \eta \rangle$-consistency of $W'$. Next, assume toward contradiction some $k$, such that $\langle W(k), w \rangle \in G'.hb|_{\text{loc}}; [W]; G'.hb^2$. Then, we reach an analogous contradiction, since $\langle w, e \rangle \in G'.rf$.

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 $W(2) = 0_w(x)$, we should ensure that for every $2 < k \leq |L|$, if $1_{\text{loc}}(W(k)) = x$ then $\langle W(k), w \rangle \notin G'.hb^2$. Indeed, assume toward contradiction that $\langle W(k), w \rangle \in G'.hb^2$. Then, since $\langle w, e \rangle \in G'.rf$, $W'(m) = e$ and $e \in W$, we get that $\langle W'(f(k)), W'(m) \rangle \in G'.hb|_{\text{loc}}; [W]; G'.hb^2$. Since $f(k) > m$, this contradicts (C1 in) the $\langle G', \eta \rangle$-consistency of $W'$. \qed
C DECIDABILITY OF THE REACHABILITY PROBLEM UNDER WRA

We show that loWRA can be used for establishing the decidability of the reachability problem for the declarative WRA model (see Def. 2.8). We follow the same structure of arguments that we had for SRA in §5.1. Thus, we establish the decidability of the reachability problem for loWRA (and utilize Theorems 4.6 and 6.10). To do so, as we did for loSRA, it suffices to show that for every concurrent program \( P \), the concurrent system \( P_{\text{loWRA}} \) equipped with the wqo \( \sqsubseteq \) is a WSTS that admits effective pred-basis and effective initialization.

- \( \sqsubseteq \) is compatible with \( P_{\text{loWRA}} \): Exactly as for loSRA.
- Effective initialization: \( P_{\text{loWRA}} \) trivially admits effective initialization. Indeed, the states \( \langle \bar{p}, B \rangle \) for which \( \uparrow \{ \langle \bar{p}, B \rangle \} \cap P_{\text{loWRA}}.Q_0 \neq \emptyset \) are exactly the initial states themselves—\( P.Q_0 \times \text{loWRA}.Q_0 \).
- Effective pred-basis: As we did for \( P_{\text{loSRA}} \), to prove that \( P_{\text{loWRA}} \) has effective pred-basis, it suffices to show how to calculate a finite basis of \( \uparrow \text{pred}^\alpha_{\text{loWRA}}(\uparrow \{ B' \}) \) for \( \alpha \) of the form \( \langle \tau, R(x, v_R) \rangle, \langle \tau, \text{RW}(x, v_R) \rangle, \langle \tau, \text{RMW}(x, v_R, v_R) \rangle \) or \( \varepsilon \).

**Silent memory step** Exactly as for loSRA.

**Read** The calculation is almost the same as for loSRA, with the only difference that for \( \alpha = \langle \tau, R(x, v_R) \rangle \), the set \( \{ B'[\tau \mapsto \langle \tau_w, x, v_R, \pi_{\text{RW}} \rangle \cdot B'(\tau)] \mid \tau_w, \pi_{\text{RW}} \in \text{Tid} \} \) (rather than \( \{ B'[\tau \mapsto \langle \tau_w, x, v_R, u \rangle \cdot B'(\tau)] \mid \tau_w \in \text{Tid}, u \in \{ R, \text{RW} \} \} \) is a finite basis of \( \uparrow \text{pred}^\alpha_{\text{loWRA}}(\{ B' \}) \).

**Write** We need the following technical lemma (proved exactly as Lemma 5.20 for loSRA).

**Lemma C.1.** Let \( P \) be an index choice for \( B' \in \text{loWRA}.Q \) such that \( P \models \langle \tau, \text{W}(x, v_W) \rangle \). If \( B'_0 \subseteq B' \), then \( \text{src}(B'_0, \tau, P_0) \subseteq \text{src}(B, \tau, P) \) for some index choice \( P_0 \) for \( B'_0 \) such that \( P_0 \models \langle \tau, \text{W}(x, v_W) \rangle \).

By Prop. 6.9 and Lemma C.1, we get that the set

\[
\{ \text{src}(B', \tau, P)[\tau \mapsto 0_w(x) \cdot \text{src}(B', \tau, P)(\tau)] \mid P \text{ is an index choice for } B' \text{ such that } P \models \langle \tau, \text{W}(x, v_W) \rangle \}
\]

is a finite basis of \( \uparrow \text{pred}^{(\tau, \text{W}(x, v_W))}_{\text{loWRA}}(\uparrow \{ B' \}) \).

**RMW** The predecessor with respect to an RMW step intuitively combines the predecessors with respect to the read and write steps.

By Prop. 6.9 and Lemma C.1, we get that the set

\[
\{ \text{src}(B', \tau, x)P[\tau \mapsto \langle \tau_w, x, v_R, \tau \rangle \cdot 0_w(x) \cdot \text{src}(B', \tau, P)(\tau)] \mid \tau_w \in \text{Tid} \text{ and } P \text{ is an index choice for } B' \text{ such that } P \models \langle \tau, \text{W}(x, v_W) \rangle \}
\]

is a finite basis of \( \uparrow \text{pred}^{(\tau, \text{RMW}(x, v_R, v_W))}_{\text{loWRA}}(\uparrow \{ B' \}) \).
D PROOF OF THEOREM 3.4

Theorem 3.4. 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 $G'.E = G.E$ and $G'.rf = G.rf$ (but possibly $G'.mo \neq G.mo$) is SRA-consistent. Let $G$ 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 $G$ is empty.

Suppose otherwise and let $G$ be a minimal element in $G$, in the sense that every proper $G.hb$-prefix of $G$ is not in $G$ (where a proper $G.hb$-prefix of $G$ is any execution graph of the form $\langle E_p, [E_p] ; [E_p], [E_p] ; G.mo ; [E_p] \rangle$ for some $E_p \subseteq G.E$ such that $dom(G.hb ; [E_p]) \subseteq E_p$). Note that $G$ cannot be empty, since the empty execution graph $G_0$ is trivially SRA-pre-consistent.

Let $e$ be some $G.hb$-maximal event in $G.E$, and let $E' = G.E \setminus \{e\}$. The minimality of $G$ ensures that the restriction of $G$ to $E'$ (namely, the execution graph $\langle E', [E'] ; G.rf ; [E'], [E'] ; G.mo ; [E'] \rangle$) is SRA-pre-consistent. Let $mo'$ be a modification order for $E'$ such that $G' = \langle E', [E'] ; G.rf ; [E'], mo' \rangle$ is SRA-consistent. Note that our assumption on $P$ ensures that $G'$ is write/write-race free, thus using irr-hb-mo, it follows that $mo' \subseteq G'.hb|_{loc} \subseteq G.hb|_{loc}$.

We consider the possible types of $e$. In each case, we define a modification order $\overline{mo}$ for $G.E$ and show that $\widehat{G} = \langle G.E, G.rf, \overline{mo} \rangle$ is SRA-consistent, which contradicts the SRA-pre-consistency of $G$.

- $\text{typ}(e) = R$: We define $\overline{mo} = mo'$. Then, $\widehat{G}$ satisfies irr-hb-mo, as a $(\widehat{G}.hb \cup mo')$-cycle would have implied a cycle in $G.hb \cup mo' \subseteq G.hb$, 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 $G.hb$-maximal, there exist $w_1, w_2 \in E'$ such that $\langle w_1, w_2 \rangle \in mo'$. Then, $\langle w_1, e \rangle \in G.rf$. It follows that $\langle w_1, w_2 \rangle \in G.hb|_{loc}$, and so $G$ does not satisfy weak-coherence, which contradicts the fact that $G$ is WRA-consistent.

- $\text{typ}(e) = W$: We define $\overline{mo} = mo' \cup (G.W \times \{e\})$ where $x = 1oc(e)$. It is easy to see that $\widehat{G}$ is SRA-consistent.

- $\text{typ}(e) = RMW$: Let $x = 1oc(e)$ and let $w \in G.W$ such that $\langle w, e \rangle \in G.rf$. We define $\overline{mo} = mo' \cup (W \times \{e\}) \cup (\{e\} \times (G.W_x \setminus W))$ where $W = \{w' \in G.W_x \mid \langle w', w \rangle \in mo' \}$. Assume toward contradiction that $\widehat{G}$ is not SRA-consistent. At least one of the following hold:
  - $\text{irr-hb-mo}$ is not satisfied by $\widehat{G}$: Then, since $G'$ is SRA-consistent, there exists $w' \in E'.W$ such that $\langle e, w' \rangle \in \overline{mo}$ and $\langle w', e \rangle \in G.hb$. Hence, we have $\langle w, w' \rangle \in mo' \subseteq G.hb|_{loc}$, and since $\langle w, e \rangle \in G.rf$, this contradicts the fact that $G$ satisfies weak-coherence.
  - $\text{read-coherence}$ is not satisfied by $\widehat{G}$: Then, since $G'$ is SRA-consistent, there exist $w' \in E'$ such that $\langle w, w' \rangle \in \overline{mo}$ and $\langle w', e \rangle \in G.hb$. It follows that $\langle w, w' \rangle \in G.hb|_{loc}$, which again contradicts the fact that $G$ satisfies weak-coherence.
  - $\text{atomicity}$ is not satisfied by $\widehat{G}$: Then, since $G'$ is SRA-consistent, it follows that there exist $w' \in E'.W$ and $u \in E'.RMW$, such that $\langle w', e \rangle, \langle e, u \rangle \in \overline{mo}$ and $\langle w', u \rangle \in G.rf$. The construction of $\overline{mo}$ ensures that $\langle w', w \rangle \in mo'$ and $\langle w, u \rangle \in mo'$. Hence, $\langle w', w \rangle \in G.hb|_{loc}$ and $\langle w, u \rangle \in G.hb|_{loc}$. Now, if $\langle w, w \rangle \in G.hb|_{loc}$, then again we obtain a contradiction to the fact that $G$ satisfies weak-coherence. Otherwise, we have $w' = w$. Thus, we have both $\langle w, e \rangle \in G.rf$ and $\langle w, u \rangle \in G.rf$ (where $e \neq u$ since $u \in E'$), which contradicts the fact the $G$ satisfies weak-atomicity. □
E  RELEASE/ACQUIRE: THE opRA MEMORY SYSTEM

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

Notation E.1. Given a total order $R$ on a set $A$ and elements $a \in A$ and $b \not\in A$, $\text{AddAfter}(R, A, a, b)$ denotes the total order on $A \cup \{b\}$ obtained by placing $b$ as the immediate successor of $a$ (formally, $\text{AddAfter}(R, A, a, b) = R \cup (A' \times \{b\}) \cup \{(b) \times (A \setminus A')\}$ where $A' = \{a' \in A \mid \langle a', a \rangle \in R'\}$).

The transitions of opRA are given by:

<table>
<thead>
<tr>
<th>WRITE</th>
<th>READ</th>
<th>RMW</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e = \text{NextEvent}(G.E, \tau, W(x, v_e))$</td>
<td>$e = \text{NextEvent}(G.E, \tau, R(x, v_e))$</td>
<td>$e = \text{NextEvent}(G.E, \tau, \text{RMW}(x, v_e))$</td>
</tr>
<tr>
<td>$G'.E = G.E \cup {e}$</td>
<td>$G'.E = G.E \cup {e}$</td>
<td>$G'.E = G.E \cup {e}$</td>
</tr>
<tr>
<td>$G'.rf = G.rf$</td>
<td>$(w, e)$</td>
<td>$(w, e)$</td>
</tr>
<tr>
<td>$G'.mo = \text{AddAfter}(G.mo, G.W_X, w, e)$</td>
<td>$w \notin \text{dom}(G.mo; G.hb^?); [E^?]$</td>
<td>$w \notin \text{dom}(G.mo; G.hb^?); [E^?]$</td>
</tr>
<tr>
<td>$w \not\in G.W_X$</td>
<td>$\text{val}_w(w) = v_e$</td>
<td>$w \not\in G.W_X$</td>
</tr>
<tr>
<td>$w \not\in \text{dom}(G.rf; [\text{RMW}])$</td>
<td>$\text{val}_w(w) = v_e$</td>
<td>$\text{val}_w(w) = v_e$</td>
</tr>
</tbody>
</table>

A WRITE step by thread $\tau$ adds a corresponding fresh write event $e$ to the graph (placed after all events of thread $\tau$) 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 $\tau$ is aware. Equivalently, $e$ should be placed as the immediate successor of some event $w$, such that thread $\tau$ is not aware of any mo-successors of $w$ ($w \not\in \text{dom}(G.mo; G.hb^?); [E^?]$). 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 ($w \not\in \text{dom}(G.rf; [\text{RMW}])$).

A READ step by thread $\tau$ adds a corresponding fresh read event and justifies it with a reads-from edge. This is exactly as in opWRA, but instead of using $G.hb|_{10c}$ to capture later writes (as per weak-coherence), we use mo (as per read-coherence).

Finally, rmw 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 E.2. opRA characterizes RA.

The proof proceeds exactly as the proof for WRA (Thm. 4.6).