How Hard Is Weak-Memory Testing?

Chakraborty, Soham; Krishna, Shankara Narayanan; Mathur, Umang; Pavlogiannis, Andreas

DOI
10.1145/3632908

Publication date
2024

Document Version
Final published version

Published in
Proceedings of the ACM on Programming Languages

How Hard Is Weak-Memory Testing?

SOHAM CHAKRABORTY, TU Delft, Netherlands

SHANKARA NARAYANAN KRISHNA, IIT Bombay, India

UMANG MATHUR, National University of Singapore, Singapore and IIT Bombay, India

ANDREAS PAVLOGIANNIS, Aarhus University, Denmark

Weak-memory models are standard formal specifications of concurrency across hardware, programming languages, and distributed systems. A fundamental computational problem is *consistency testing*: is the observed execution of a concurrent program in alignment with the specification of the underlying system? The problem has been studied extensively across Sequential Consistency (SC) and weak memory, and proven to be NP-complete when some aspect of the input (e.g., number of threads/memory locations) is unbounded. This unboundedness has left a natural question open: are there efficient *parameterized* algorithms for testing?

The main contribution of this paper is a deep hardness result for consistency testing under many popular weak-memory models: the problem remains NP-complete even in its *bounded* setting, where candidate executions contain a bounded number of threads, memory locations, and values. This hardness spreads across several Release-Acquire variants of C11, a popular variant of its Relaxed fragment, popular Causal Consistency models, and the POWER architecture. To our knowledge, this is the first result that fully exposes the hardness of weak-memory testing and proves that the problem *admits no parameterization* under standard input parameters. It also yields a computational separation of these models from SC, x86-TSO, PSO, and Relaxed, for which bounded consistency testing is either known (for SC), or shown here (for the rest), to be in polynomial time.

CCS Concepts: • **Software and its engineering → Software verification and validation**; • **Theory of computation** → *Theory and algorithms for application domains*; *Program analysis.*

Additional Key Words and Phrases: concurrency, consistency checking, weak memory models, complexity

ACM Reference Format:

Soham Chakraborty, Shankara Narayanan Krishna, Umang Mathur, and Andreas Pavlogiannis. 2024. How Hard Is Weak-Memory Testing?. *Proc. ACM Program. Lang.* 8, POPL, Article 66 (January 2024), 32 pages. https://doi.org/10.1145/3632908

1 INTRODUCTION

Memory-consistency models play a crucial role in the design, use, and verification of concurrent systems spanning hardware, programming languages, and distributed computing. These models formally define the set of behaviors that the system can exhibit as a whole, accounting for the

Authors' addresses: Soham Chakraborty, TU Delft, Delft, Netherlands, s.s.chakraborty@tudelft.nl; Shankara Narayanan Krishna, IIT Bombay, Mumbai, India, krishnas@cse.iitb.ac.in; Umang Mathur, National University of Singapore, Singapore, Singapore and IIT Bombay, Mumbai, India, umathur@comp.nus.edu.sg; Andreas Pavlogiannis, Aarhus University, Aarhus, Denmark, pavlogiannis@cs.au.dk.

ACM 2475-1421/2024/1-ART66
https://doi.org/10.1145/3632908

intricate communication patterns between its entities due to buffers, caching, message delays, etc. The simplest and most widespread, general model is Sequential Consistency (SC) [Lamport 1978], which defines program behavior by thread interleaving. Although its simplicity is a major advantage, SC fails to capture the additional, complex behaviors that are abundant in modern concurrency.

In contrast, *weak-memory models* are richer and more faithful specifications of concurrent/distributed communication and are developed specifically for the system under consideration. For example, the x86, POWER, and Arm architectures follow their own memory models [Alglave et al. 2021, 2014; Owens 2010], programming languages provide certain primitives for writing weak-memory concurrent programs [Batty et al. 2013], and distributed systems implement various models of causal consistency [Bouajjani et al. 2017; Burckhardt 2014; Hutto and Ahamad 1990]. Naturally, verification techniques are specific to the memory model at hand, so as to account for (and verify) all the possible behaviors that the system can exhibit according to the model.

One of the core computational problems associated with a memory model is that of *consistency testing: is a high-level, observed behavior of a program in alignment with the semantics of the underlying model* [Gibbons and Korach 1997]? The observed behavior is specified in terms of an abstract execution that defines the sequence of instructions each process/thread executed, the shared memory locations it read from/wrote to, along with the respective values read/written. Answering this question requires determining the low-level, unobserved behavior of the architecture that gave rise to the observed behavior of the program; for example, the order in which writes were made visible to (one or more of) the threads, and the dataflow between writes and reads.

Consistency testing is a natural task in both the development and the implementation of memory models. In particular, memory models are contracts between the designers of a system and its users [Adve and Hill 1990]. When designing hardware architectures, memory subsystems, compiler optimizations, and distributed-communication protocols, consistency-testing serves to validate that the contract has been respected [Chen et al. 2009; Gibbons and Korach 1997; Manovit and Hangal 2006; Qadeer 2003; Windsor et al. 2022]. From the opposite direction, litmus testing is a standard approach to understanding the semantics of hardware architectures [Alglave et al. 2011, 2014] so as to design faithful models around them. Here, given a candidate memory model and the observed execution of a litmus test, consistency checking verifies whether the execution is a counterexample to the model. Finally, consistency testing is also used as a separability criterion between different memory models [Kokologiannakis et al. 2023; Wickerson et al. 2017].

Consistency checks are also a widespread task in program verification and testing. In stateless model checking, the goal of the model checker is to enumerate-and-check the absence of errors in all program executions (typically up to some bound). To reduce the load of the verification task, an abstraction mechanism partitions the space of all behaviors into equivalence classes, each represented by an abstract execution. Instead of enumerating concrete executions, the model checker enumerates abstract executions, which yields an exponential reduction of the search space. Each candidate abstract execution undergoes a consistency check to ensure that the model checker does not diverge to unrealizable parts of the search space [Abdulla et al. 2023, 2019, 2018; Agarwal et al. 2021; Bui et al. 2021; Chalupa et al. 2017; Chatterjee et al. 2019; Kokologiannakis et al. 2022, 2019]. In runtime testing, predictive techniques aim to infer the presence of unobserved, erroneous executions from observed executions that are bug-free. Such techniques operate by constructing a candidate execution that manifests the bug (using the observed execution as a guide) and then applying a consistency check (explicitly or implicitly) to verify that the execution indeed represents

valid program behavior (hence the bug report is a true positive) [Huang et al. 2014; Kalhauge and Palsberg 2018; Kini et al. 2017; Luo and Demsky 2021; Mathur et al. 2020, 2021; Pavlogiannis 2019].

Owing to the widespread applicability, the computational complexity of consistency testing has been studied thoroughly for a wide variety of memory models and a wide variety of settings. The seminal work of Gibbons and Korach [1997] showed that the problem is NP-complete for SC, even when either the number of threads or the number of memory locations is bounded (but not both). Later, Cantin et al. [2005] proved that the problem remains NP-complete even with a single memory location (but with unboundedly many threads), which also implies NP-completness for all memory models that adhere to the "SC-per-location" property, such as TSO, PSO, RA, SRA and Relaxed. Gonthmakher et al. [2003] showed a similar NP-completeness for a Java memory model. Furbach et al. [2015] proposed a unified treatment of the consistency problem on many weak-memory models which led to similar NP-completeness results, while the NP-completeness of consistency testing under various causal-consistency models was proven in [Bouajjani et al. 2017].

A popular approach to tackle the intractability in consistency testing is via *parameterization*: an intractable problem becomes tractable when some of its input parameters, such as the number of threads, is bounded [Abdulla et al. 2018; Gibbons and Korach 1994; Mathur et al. 2020]. On the other hand, all existing results that establish the NP-hardness of consistency testing in all memory models rely on some input parameters being unbounded. For SC, this unboundedness is, in fact, a prerequisite for intractability: the problem becomes polynomial-time when both the number of threads and memory locations are bounded, a result that has led to efficient parameterized model checking [Agarwal et al. 2021]. For weak-memory, however, analogous results have thus far remained elusive. In particular, are there any efficient parameterized algorithms for consistency in weak memories? *How hard, actually, is weak-memory testing?*

Here we resolve this question by establishing a deep hardness result for many popular weak-memory models: consistency testing is NP-complete even in its *bounded* setting, where executions contain a constant number of threads, memory locations and values, and the size of the input is solely determined by the (unbounded) number of events. To our knowledge, this is the first result that fully exposes the hardness of weak-memory testing and proves that the problem admits no parameterization under standard input parameters. In turn, this implies that practical approaches to testing have to resort to heuristics, while model checkers might be more performant when exploring finer abstractions (such as reads-from [Abdulla et al. 2019; Chalupa et al. 2017; Tunç et al. 2023], or those based on executions graphs [Kokologiannakis et al. 2017; Lahav and Margalit 2019]).

Our contributions. We study the *bounded consistency testing problem* for many popular weak-memory models found in software, hardware, and distributed systems. The input is always an abstract execution $\overline{\mathsf{X}} = (\mathsf{E}, \mathsf{po})$ consisting of a set of events E and a program order po defining the order of execution of these events in each thread. The task is to determine whether $\overline{\mathsf{X}}$ is consistent in a given memory model. The *boundedness* of the problem refers to the number of threads, memory locations, and values accessed by $\overline{\mathsf{X}}$ being bounded (i.e., constant). It is easy to see that the problem is in NP in all the models we consider, and we will not be establishing this fact formally. We write $\mathcal{M}_1 \preccurlyeq \mathcal{M}_2$ to denote that memory model $\mathcal{M}_2$ is weaker than memory model $\mathcal{M}_1$, i.e., any execution that is consistent in $\mathcal{M}_1$ is also consistent in $\mathcal{M}_2$.

We begin with release-acquire semantics, as popularized by C11. We consider the Release-Acquire model (RA), as well as its Strong (SRA) and Weak variants (WRA) [Lahav and Boker 2022]. In addition, we consider Relaxed-Acyclic, the standard Relaxed semantics of C11 equipped with the common assumption of causal acyclicity (aka $(\mathsf{po} \cup \mathsf{rf})$ acyclicity). This assumption is often used as

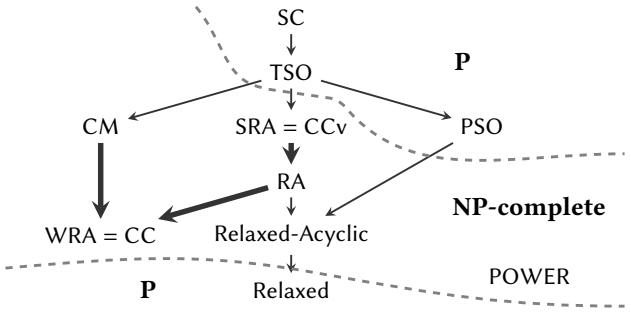


Fig. 1. The complexity landscape of bounded weak-memory testing. An arrow $\mathcal{M}_1 \to \mathcal{M}_2$ means that $\mathcal{M}_1$ is stronger than $\mathcal{M}_2$. Thick arrows represent range-hardness for all models between the endpoints. The complexity of bounded consistency checking for all models except for SC are established in this paper.

an additional axiom [Margalit and Lahav 2021; Norris and Demsky 2013] as it has been argued that $(\mathrm{po} \cup \mathrm{rf})$-cycles do not arise in practice [Lee et al. 2023]. We prove the following theorem.

Theorem 1.1. *Consistency testing for bounded inputs is NP-complete for* Relaxed-Acyclic *as well as for any memory model* $\mathcal{M}$ *such that* $\mathsf{SRA} \preccurlyeq \mathcal{M} \preccurlyeq \mathsf{WRA}$*, even in their atomic read-modify-write (RMW)-free fragment.*

Note that Theorem 1.1 establishes hardness for Relaxed-Acyclic, WRA, RA, SRA as well as the whole range of models between SRA and WRA. This result improves existing results on consistency checking, for which hardness relied on an unbounded domain of threads and/or memory locations [Cantin et al. 2005; Furbach et al. 2015; Gibbons and Korach 1994].

Next, we turn our attention to popular causal-consistency models [Fidge 1988; Lamport 1978]. There have been several efforts to formalize various aspects of causal consistency, out of which have emerged three well-accepted models, namely Causal Consistency CC [Bouajjani et al. 2017; Hutto and Ahamad 1990], Causal Convergence CCv [Bouajjani et al. 2017; Burckhardt 2014; Perrin et al. 2016], and Causal Memory CM [Ahamad et al. 1995; Bouajjani et al. 2017; Perrin et al. 2016]. It was recently shown that CC coincides with WRA while CCv coincides with SRA [Lahav and Boker 2022]. Thus Theorem 1.1 extends to CC and CCv. We prove that the problem is also hard for CM, thereby establishing hardness for the ranges defined by the three main models.

Theorem 1.2. *Consistency testing for bounded inputs is NP-complete for any memory model* $\mathcal{M}$ *such that (i)* $\mathsf{CCv} \preccurlyeq \mathcal{M} \preccurlyeq \mathsf{CC}$ *or (ii)* $\mathsf{CM} \preccurlyeq \mathcal{M} \preccurlyeq \mathsf{CC}$.

Next, we turn our attention to the POWER architecture. Lahav et al. [2016] show that SRA captures precisely the guarantees of POWER for programs that are compiled from the release-acquire fragment of C/C++. Thus Theorem 1.1 extends to the following corollary.

Corollary 1.3. *Consistency testing for bounded inputs is NP-complete for* POWER.

Continuing with hardware models, we study Total Store Order (TSO) as employed in x86 architectures (aka x86-TSO) and its extension to Partial Store Order (PSO). It turns out that, in the bounded setting, consistency checks become tractable in these models.

Theorem 1.4. *Consistency testing for bounded threads and memory locations is in polynomial time for* TSO *and* PSO.

One natural, final question concerns the vanilla Relaxed model, i.e., if we remove the acyclicity condition from Relaxed-Acyclic. In this case, the problem becomes polynomial-time, which is a corollary of the corresponding result for SC [Agarwal et al. 2021].

Corollary 1.5. *Consistency testing for bounded threads is in polynomial time for* Relaxed.

Although Corollary 1.5 is technically straightforward, it is conceptually interesting under the following realization. For all of our previous results (as well as for SC), the hardness of consistency coincides with whether the corresponding model exhibits multi-copy atomicity. In contrast, Relaxed is non-multi-copy atomic, yet consistency testing is in polynomial time.

Following the results of this paper, Fig. 1 pictorially presents the full landscape of the tractability and the hardness in testing weak memories.

High-level intuition. Our proofs exploit complex combinatorial properties that arise in weak memory. Although it is hard to pinpoint one key insight that fully explains our hardness results, our proofs rely on the fact that most of the models we consider (i) are causally consistent, and (ii) allow $(\mathsf{po}' \cup \mathsf{rf} \cup \mathsf{fr})$-cycles, where po' is the standard program order restricted to instructions of the same type (read-read and write-write orderings) on different locations. In contrast, the polynomial-time models SC and TSO forbid (ii), while PSO allows (ii) but also fails (i).

Outline. The rest of the paper is organized as follows.

- In Section 2, we define our problem setting and the memory models we consider based on C/C++ atomics. We also develop relevant notation that will be helpful in later sections.
- In Section 3, we prove Theorem 1.1 for Relaxed-Acyclic. For readability, we prove a weaker version of Theorem 1.1 in which the inputs use boundedly many threads and locations but manipulate unboundedly many values. Later in Section 6, we explain how to perform simple modifications to our reduction to make it work even for bounded values.
- In Section 4, we prove Theorem 1.1 for all models SRA $\preccurlyeq \mathcal{M} \preccurlyeq$ WRA. Similarly to the previous case, our reduction uses unboundedly many values, while the modifications described in Section 6 also apply to this model, to arrive at the final result.
- In Section 5, we establish Theorem 1.2, Corollary 1.3, Theorem 1.4 and Corollary 1.5.
- Finally, in Section 6, we present the modifications in the reductions of Section 3 and Section 4 that fully establish Theorem 1.1.

Due to space restrictions, the full paper appears as a technical report in [Chakraborty et al. 2023].

2 PRELIMINARIES

This section defines the axiomatic semantics of the SRA, RA, WRA, and Relaxed memory models. As these are standard concepts, our exposition follows recent work on the topic (e.g., [Lahav and Boker 2022; Margalit and Lahav 2021; Tunç et al. 2023]). In axiomatic semantics, program executions consist of sets of events and relations between them. Given an integer i, we let $[i] = \{1, 2, \ldots, i\}$.

Events. An event is a tuple $\langle \mathsf{id}, \mathsf{tid}, \mathsf{lab} \rangle$ where id, tid, lab denote a unique identifier, thread identifier, and the label respectively. The label is of the form $\mathsf{lab} = \langle \mathsf{op}, \mathsf{loc}, \mathsf{Val}, \mathsf{ord} \rangle$ where op, loc, Val, ord respectively denote a read (r) or write (w) memory operation, accessed memory location, read or written value, and memory order respectively. For the SRA, RA, and WRA models, reads and writes are of *acquire* and *release* orders respectively. For the Relaxed model, the read and write accesses have *relaxed* order. These memory orders are used to define the semantics of models like C11, but we will not be using them explicitly here. As we treat each model separately, all access orders are

Table 1. Variants of the coherence axioms.

Release-Acquire		Relaxed	
$\mathsf{irr}(\mathsf{mo}_x; \mathsf{hb})$	(write-coherence)	$\mathsf{irr}(\mathsf{mo}_x; \mathsf{po})$	(relaxed-write-coherence)
$\mathsf{acy}(\mathsf{hb} \cup \mathsf{mo})$	(strong-write-coherence)		
$\mathsf{irr}(\mathsf{rf}^{-1}; \mathsf{mo}_x; \mathsf{hb})$	(read-coherence)	$\mathsf{irr}(\mathsf{rf}^{-1}; \mathsf{mo}_x; \mathsf{rf}^?; \mathsf{po})$	(relaxed-read-coherence)
$\mathsf{irr}(\mathsf{hb}_x; [\mathsf{W}]; \mathsf{hb}_x; \mathsf{rf}^{-1})$	(weak-read-coherence)		

determined by the models and are never mixed. Hence, we will simply write $\mathtt{r}(t, x, v)/\mathtt{w}(t, x, v)$ to denote a read/write event of thread t, accessing location x and reading/writing value v. We occasionally omit x and/or v, when it is irrelevant or clear from the context, while we let $\mathsf{tid}(e)$ denote the thread of event e. We do not introduce fences or atomic read-modify-write (RMW) events, as all our hardness results hold even with only read/write events, while our positive results can be easily extended to handle fences and RMWs. Finally, we denote the set of read and write accesses by R and W respectively.

Notation on relations. Let B be a binary relation over a set of events E. The reflexive, transitive, reflexive-transitive closures, and inverse relations of B are denoted as $B^?$, B^+, B^*, and B^{-1}, respectively. We compose two relations B_1 and B_2 as $B_1; B_2$. $[A]$ denotes the identity relation on a set A. We write $\mathsf{irr}(B)$ and $\mathsf{acy}(B)$ to denote that B is irreflexive and acyclic, respectively. We occasionally write that there exists a B-edge $e \xrightarrow{B} e'$ to denote that $(e, e') \in B$. We naturally extend this notation to paths, so that a B-path $P\colon e \overset{B}{\rightsquigarrow} e'$ is a sequence of B-edges $e = e_1 \xrightarrow{B} e_2 \xrightarrow{B} \cdots \xrightarrow{B} e_i = e'$. Finally, we write B_x to restrict B on events accessing location x.

Executions and relations. An execution is a tuple $\mathsf{X} = \langle \mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo} \rangle$ where E is a set of events and po, rf, mo are binary relations over E. In particular, the *program order* ($\mathsf{po} \subseteq (\mathsf{E} \times \mathsf{E})$) is a strict total order on the events of each thread. The *reads-from* relation ($\mathsf{rf} \subseteq (\mathsf{W} \times \mathsf{R})$) relates a write and read event pair $(\mathtt{w}, \mathtt{r})$, denoting that $\mathtt{r}$ obtains its value from $\mathtt{w}$. Every read reads from exactly one write on the same memory location and having the same value (thus rf^{-1} is a function). The *modification order* ($\mathsf{mo} \subseteq \bigcup_x (\mathsf{W}_x \times \mathsf{W}_x)$) is a strict total order over same-location writes in an execution. Finally, the *happens-before* relation is defined as $\mathsf{hb} \triangleq (\mathsf{po} \cup \mathsf{rf})^+$. Fig. 2 shows examples of executions presented as *execution graphs*. In each execution graph the nodes represent events and the edges represent relations. We omit some relation-edges that are clear from the context [*].

Consistency Axioms. Consistency axioms capture different aspects or properties of an execution, such as coherence and causality cycles, under a memory model. These properties are interpreted differently in different memory models.

Coherence. In an execution, *coherence* enforces an ordering between same-location events. For events using the release-acquire memory orders, *write-coherence* requires that each mo_x order agrees with hb. A stronger variant is *strong-write-coherence*, which requires that mo agrees with hb, transitively. *Read coherence* enforces that a read $\mathtt{r}$ can read from a write $\mathtt{w}$ when there is no *intermediate* write $\mathtt{w}'$ on the same-location that happens-before $\mathtt{r}$. Depending upon how "intermediate" writes are treated, two variations of read coherence are popular — in standard *read-coherence*, $\mathtt{w}$ and $\mathtt{w}'$ are ordered by mo_x whereas in *weak-read-coherence* they are ordered by hb_x. Finally, we also have variants of write and read coherence when all accesses are relaxed. Here hb is replaced with po, as rf-edges do not contribute to hb between different memory locations.

[*]It is also common to define a *from reads* relation $\mathsf{fr} \triangleq \mathsf{rf}^{-1}; \mathsf{mo}$. However, we will not be using fr explicitly in this paper.

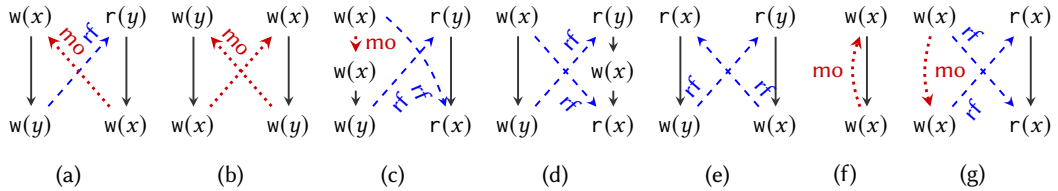


Fig. 2. Executions forbidden by (a) write-coherence, (b) strong-write-coherence, (c) read-coherence, (d) weak-read-coherence, (e) porf-acyclicity, (f) relaxed-write-coherence, (g) relaxed-read-coherence.

Table 2. The main weak-memory models based on C11 that we consider in this work.

	Release-Acquire		Relaxed
WRA	porf-acyclicity weak-read-coherence	Relaxed	relaxed-write-coherence relaxed-read-coherence
RA	write-coherence read-coherence	Relaxed-Acyclic	relaxed-write-coherence relaxed-read-coherence porf-acyclicity
SRA	strong-write-coherence read-coherence		

Causality Cycles. A causality cycle arises in the presence of relaxed accesses and consists of po and rf orderings. A causality cycle may result in 'out-of-thin-air' behavior in an execution. To avoid such 'out-of-thin-air' behavior, many consistency models and verification tools explicitly disallow such cycles [Luo and Demsky 2021; Margalit and Lahav 2021; Norris and Demsky 2013].

- $\mathsf{acy}(\mathsf{po} \cup \mathsf{rf})$ (porf-acyclicity)

Fig. 2 shows examples of executions forbidden by different axioms. The write-coherence axiom forbids the execution in Fig. 2a as it violates the irreflexivity of $(\mathsf{mo}_x; \mathsf{hb})$. The $(\mathsf{hb} \cup \mathsf{mo})$ cycle in Fig. 2b is forbidden by strong-write-coherence. The execution in Fig. 2c violates irreflexivity of $(\mathsf{rf}^{-1}; \mathsf{mo}_x; \mathsf{hb})$ and thus fails read-coherence. In Fig. 2d, we have $(\mathtt{w}(x), \mathtt{w}(x)) \in \mathsf{hb}_x; [\mathtt{W}]$, $(\mathtt{w}(x), \mathtt{r}(x)) \in \mathsf{po} \subseteq \mathsf{hb}$, and $(\mathtt{r}(x), \mathtt{w}(x)) \in \mathsf{rf}^{-1}$, violating weak-read-coherence. The execution in Fig. 2e violates porf-acyclicity. Finally, the executions in Fig. 2f and Fig. 2g violate relaxed-write-coherence and relaxed-read-coherence, respectively.

Memory Models. We can now describe the main memory models we consider in this work, by listing the axioms that each execution needs to satisfy in the respective model Table 2.

Release-Acquire and variants. The release-acquire (RA) memory model is weaker than sequential consistency and is arguably the most well-understood fragment of C11. Here, the reads-from relation rf induces synchronization between thread threads, which is captured in the semantics by the happens-before relation hb. Following [Lahav and Boker 2022], we consider three variants of release-acquire models: Release-Acquire (RA), and its Strong (SRA) and Weak (WRA) variants.

SRA enforces strong-write-coherence on write accesses whereas RA enforces write-coherence. On the other hand, WRA does not place any ordering between same-location writes by mo_x. Instead, the only orderings considered between same-location writes are through the $[\mathtt{W}]; \mathsf{hb}_x; [\mathtt{W}]$ relation.

Relaxed. All accesses in the Relaxed model satisfy the corresponding coherence axioms relaxed-write-coherence and relaxed-read-coherence, which guarantee SC-per-location. The Relaxed-Acyclic model strengthens Relaxed by also requiring the acyclicity of $(\mathsf{po} \cup \mathsf{rf})$.

Based on the set of allowed behaviors, these models can be partially ordered as SRA $\preccurlyeq$ RA $\preccurlyeq$ {WRA, {Relaxed-Acyclic $\preccurlyeq$ Relaxed}}, where models towards the right allow more behaviors.

The consistency-testing problem. An execution X is *consistent* in a memory model $\mathcal{M}$, written $\mathsf{X} \models \mathcal{M}$, if it satisfies the axioms of $\mathcal{M}$. For example, the execution in Fig. 2b satisfies all axioms except strong-write-coherence, and hence it is consistent in RA, WRA, and Relaxed(-Acyclic).

When testing the behavior of a program within a memory model, one does not have access to *concrete* executions, but rather to *abstract* executions. The latter contains only information observed by the program, i.e., the events it executed and the values it read/wrote. Formally, an *abstract execution* $\overline{\mathsf{X}} = \langle \mathsf{E}, \mathrm{po} \rangle$ is a coarser object than concrete executions, missing the mo and rf relations, and a concrete execution $\mathsf{X} = \langle \mathsf{E}', \mathrm{po}', \mathrm{rf}', \mathrm{mo}' \rangle$ is said to be an extension of $\overline{\mathsf{X}}$ if $\mathsf{E}' = \mathsf{E}$ and $\mathrm{po}' = \mathrm{po}$. We call $\overline{\mathsf{X}}$ *consistent* in $\mathcal{M}$, written similarly as $\overline{\mathsf{X}} \models \mathcal{M}$, if there exists an rf and an mo such that the extension $\mathsf{X} = \langle \mathsf{E}, \mathrm{po}, \mathrm{rf}, \mathrm{mo} \rangle$ is an execution with $\mathsf{X} \models \mathcal{M}$. The problem of *consistency testing* in a memory model $\mathcal{M}$ is to determine whether $\overline{\mathsf{X}}$ is consistent in $\mathcal{M}$, i.e., whether there is a way to resolve rf and mo in $\mathcal{M}$ that would give rise to the observed behavior $\overline{\mathsf{X}}$ on the program level.

Conflicting triplets. In the coming sections, we use the notion of *conflicting triplets*. Given an abstract execution $\overline{\mathsf{X}} = (\mathsf{E}, \mathrm{po})$, we say that two events $e_1, e_2 \in \mathsf{E}$ *conflict* if they access the same location and at least one of them is a write. Given additionally a reads-from relation rf, a *conflicting triplet* (or *triplet*, for short) is a tuple $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ of pairwise conflicting events such that $(\mathtt{w}, \mathtt{r}) \in \mathrm{rf}$.

3 HARDNESS FOR RELAXED-ACYCLIC

We start with the Relaxed-Acyclic memory model and show that consistency testing is NP-complete under bounded threads and memory locations. This differs slightly from Theorem 1.1, which states that the problem remains hard even with bounded values. Since our proof is rather technical, we choose to present this intermediate result here. We will make the final step towards Theorem 1.1 in Section 6, which consists of a simple modification of the technique presented here. In Section 3.1, we present the hardness reduction and argue about its correctness in Sections 3.2 and 3.3.

3.1 Reduction

Our reduction is from Monotone 1-in-3 SAT which is known to be NP-complete [Garey and Johnson 1990]. The input is a monotone formula φ in conjunctive normal form, where each clause contains three literals, all of which are positive. The task is to determine if there exists a 1-in-3 truth assignment for φ, i.e., one that sets exactly one literal to *true* in each clause.

We remark that our reduction is combinatorially elaborate. We found that complex interactions between threads are necessary to expose the nuances that make the consistency problem for the Relaxed-Acyclic memory model (or for that matter, other memory models we consider) hard. Nevertheless, we assist the text with illustrations that help visualize and generalize the interaction patterns that are exploited in our reduction. To further enhance readability, we distinguish different memory locations with different colors (in both figures and main text).

Let $\varphi = \{C_i\}_{i \in [m]}$ be a monotone Boolean formula over n variables $\{s_j\}_{j \in [n]}$ and m clauses of the form $C_i = (s_j, s_k, s_\ell)$. We construct an abstract execution $\overline{\mathsf{X}} = (\mathsf{E}, \mathrm{po})$ such that $\overline{\mathsf{X}} \models$ Relaxed-Acyclic iff φ is satisfiable using a 1-in-3 assignment[†].

[†]We often use the phrase 'φ is satisfiable' to mean 'φ is satisfiable by a 1-in-3 assignment'.

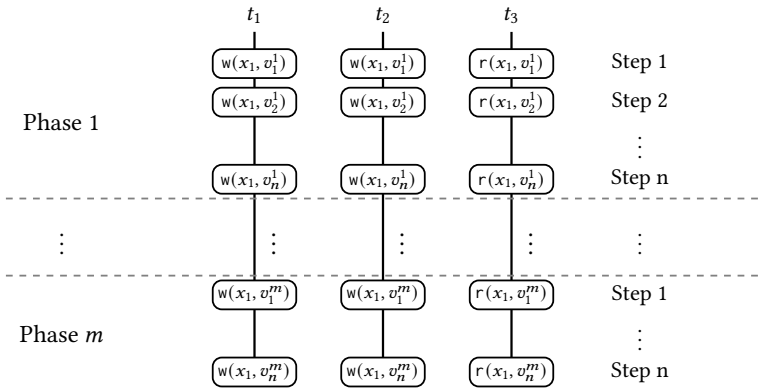


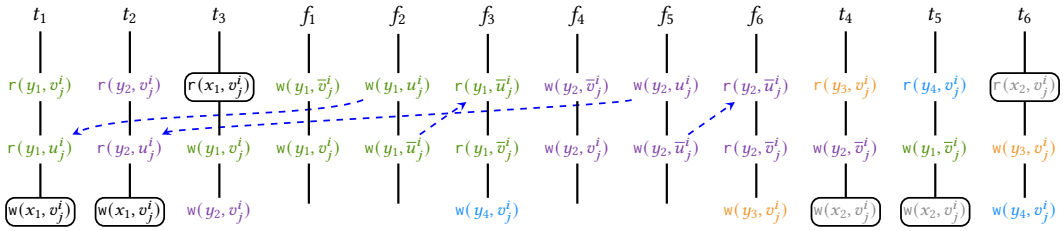
Fig. 3. The schematic reduction from a monotone formula φ to an abstract execution $\overline{\mathrm{X}}$.

High-level description. Our reduction constructs an abstract execution with $O(n \cdot m)$ events accessing $d = 14$ memory locations in $\kappa = 23$ threads, of which the three threads t_1, t_2 and t_3 form the core of the construction. Events appear in each of these threads in m *phases* (one phase per clause C_i), starting from phase 1 and going to larger phases as we go downwards in the threads. Each phase, in turn, consists of n *steps* (one step per variable s_j), again starting from step 1 and going to larger steps as we go downwards. In phase i and step j we have a read event $\mathtt{r}(t_3, x_1, v_j^i)$ that can read from either of two writes $\mathtt{w}(t_1, x_1, v_j^i)$ or $\mathtt{w}(t_2, x_1, v_j^i)$. The former case corresponds to the assignment $s_j = \bot$, while the latter case corresponds to the assignment $s_j = \top$. See Fig. 3 for an illustration. Our construction guarantees that the choices for the writer of $\mathtt{r}(t_3, x_1, v_j^i)$ are consistent across all phases i: either each $\mathtt{r}(t_3, x_1, v_j^i)$ reads from $\mathtt{w}(t_1, x_1, v_j^i)$, which corresponds to setting $s_j = \bot$ in φ, or each $\mathtt{r}(t_3, x_1, v_j^i)$ reads from $\mathtt{w}(t_2, x_1, v_j^i)$, which corresponds to setting $s_j = \top$ in φ. Moreover, for each clause $C_i = (s_j, s_k, s_\ell)$, our reduction guarantees that exactly one of $\mathtt{r}(t_3, x_1, v_j^i)$, $\mathtt{r}(t_3, x_1, v_k^i)$, and $\mathtt{r}(t_3, x_1, v_\ell^i)$ reads from thread t_2, which implies that the corresponding assignment on s_j, s_k and s_ℓ satisfies the 1-in-3 property. To achieve all these constraints, we introduce four *gadgets*, which consist of events on additional threads and memory locations, that guarantee the desired properties. In the following, we first describe each gadget separately, and then explain how to interleave them in order to obtain the abstract execution $\overline{\mathrm{X}}$.

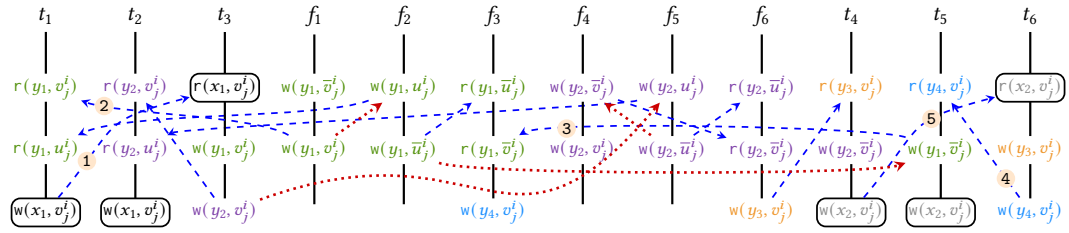
The copy gadget Copy_j^i. The main gadget in our construction is the copy gadget Copy_j^i, defined for each $i \in [m]$ and $j \in [n]$, and shown in Fig. 4. This gadget contains (i) the three focal events $\mathtt{w}(t_1, x_1, v_j^i)$, $\mathtt{w}(t_2, x_1, v_j^i)$ and $\mathtt{r}(t_3, x_1, v_j^i)$ that determine the truth value of s_j, (ii) three "mirror" events $\mathtt{w}(t_4, x_2, v_j^i)$, $\mathtt{w}(t_5, x_2, v_j^i)$ and $\mathtt{r}(t_6, x_2, v_j^i)$, and (iii) other events on memory locations y_1, y_2, y_3, y_4.

The gadget couples the writers of $\mathtt{r}(t_3, x_1, v_j^i)$ and $\mathtt{r}(t_6, x_2, v_j^i)$: if $\mathtt{r}(t_3, x_1, v_j^i)$ reads from thread t_1 then $\mathtt{r}(t_6, x_2, v_j^i)$ reads from thread t_4 (see Fig. 4b), while if $\mathtt{r}(t_3, x_1, v_j^i)$ reads from thread t_2 then $\mathtt{r}(t_6, x_2, v_j^i)$ reads from thread t_5 (see Fig. 4c).

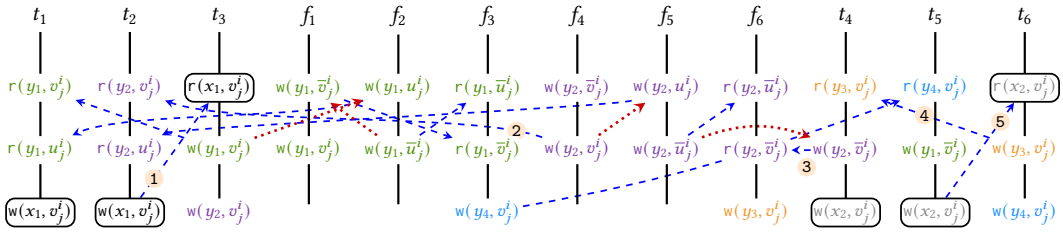
The copy-down gadget $\overline{\mathrm{Copy}}_j^i$. We use a copy-down gadget $\overline{\mathrm{Copy}}_j^i$, defined for $i \in [m-1]$ and $j \in [n]$, with structure identical to Copy_j^i, and shown in Fig. 5. This gadget contains (i) the three focal events $\mathtt{w}(t_1, x_1, v_j^{i+1})$, $\mathtt{w}(t_2, x_1, v_j^{i+1})$ and $\mathtt{r}(t_3, x_1, v_j^{i+1})$, (ii) the three mirror events $\mathtt{w}(t_4, x_2, v_j^i)$, $\mathtt{w}(t_5, x_2, v_j^i)$ and $\mathtt{r}(t_6, x_2, v_j^i)$, and (iii) other events on memory locations z_1, z_2, z_3, z_4.



(a) The copy gadget Copy^i_j.



(b) Choosing $\mathtt{r}(t_3, x_1, v^i_j)$ to read from t_1 forces the sequence of rf and mo edges shown.

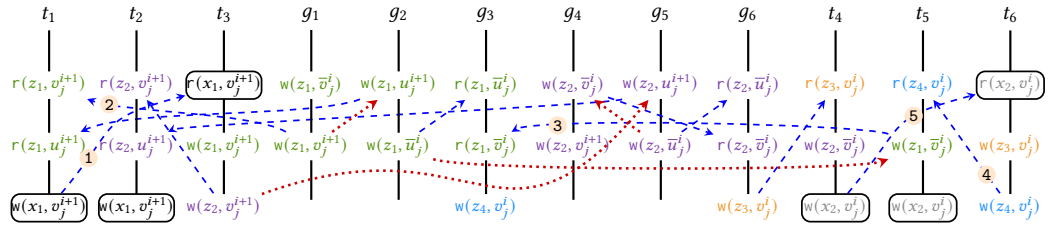


(c) Choosing $\mathtt{r}(t_3, x_1, v^i_j)$ to read from t_2 forces the sequence of rf and mo edges shown.

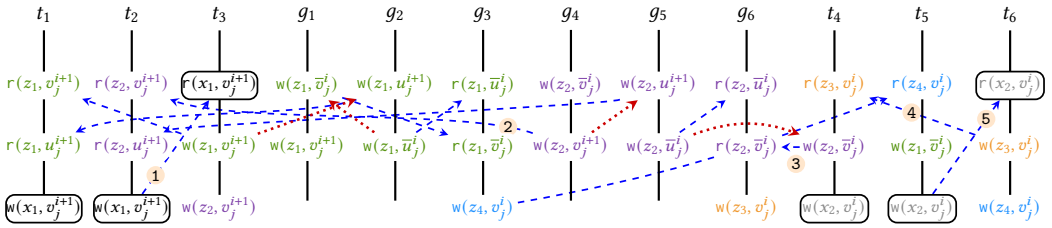
Fig. 4. The copy gadget Copy^i_j (a) captures the Boolean assignment to variable s_j in phase i. There are two ways to realize this gadget, by choosing which of the two writes $\mathtt{w}(x_1, v^i_j)$ the read $\mathtt{r}(x_1, v^i_j)$ observes. Choosing the write of t_1 (b) corresponds to setting $s_j = \bot$ and also forces $\mathtt{r}(x_2, v^i_j)$ to read from t_4. Choosing the write of t_2 (c) corresponds to setting $s_j = \top$ and also forces $\mathtt{r}(x_2, v^i_j)$ to read from t_5. This rf coupling is formalized in Lemma 3.1. The edge numbers specify the order in which rf-edges are inferred.

While Copy^i_j couples the writers of the focal and mirror read events — $\mathtt{r}(t_3, x_1, v^i_j)$ and $\mathtt{r}(t_6, x_2, v^i_j)$ — belonging to the same phase, $\overline{\mathrm{Copy}}^i_j$ couples the writers of the focal and mirror read events — $\mathtt{r}(t_3, x_1, v^{i+1}_j)$ and $\mathtt{r}(t_6, x_2, v^i_j)$ — of consecutive phases. If $\mathtt{r}(t_3, x_1, v^{i+1}_j)$ reads from thread t_1, then $\mathtt{r}(t_6, x_2, v^i_j)$ reads from thread t_4, while if $\mathtt{r}(t_3, x_1, v^{i+1}_j)$ reads from thread t_2, then $\mathtt{r}(t_6, x_2, v^i_j)$ reads from thread t_5. Together, Copy^i_j and $\overline{\mathrm{Copy}}^i_j$ couple the writers of the focal reads $\mathtt{r}(t_3, x_1, v^i_j)$ and $\mathtt{r}(t_3, x_1, v^{i+1}_j)$ across consecutive phases — either both read from thread t_1 or both read from t_2.

The at-most-one-true gadgets $\mathrm{AMOne}[c]^i_{j,k}$. Consider a clause C_i, and for each $c \in [3]$, let (s_j, s_k) be the c-th pair of literals that appear in C_i (according to some arbitrary but fixed total ordering on pairs of propositional variables). The at-most-one-true gadget $\mathrm{AMOne}[c]^i_{j,k}$ is shown in Fig. 6a and contains (i) the six focal events $\mathtt{w}(t_1, x_1, v^i_j)$, $\mathtt{w}(t_2, x_1, v^i_j)$ and $\mathtt{r}(t_3, x_1, v^i_j)$; and $\mathtt{w}(t_1, x_1, v^i_k)$, $\mathtt{w}(t_2, x_1, v^i_k)$ and $\mathtt{r}(t_3, x_1, v^i_k)$, and (ii) other events on memory location a_c. The gadget guarantees that at most



(a) Choosing $\mathsf{r}(t_3, x_1, v_j^i)$ to read from t_1 forces the sequence of rf and mo edges shown.



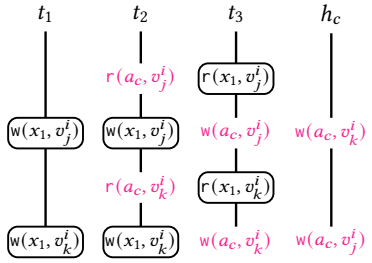
(b) Choosing $\mathsf{r}(t_3, x_1, v_j^i)$ to read from t_2 forces the sequence of rf and mo edges shown.

Fig. 5. The copy-down gadget $\overline{\text{Copy}}_j^i$ is very similar to Copy_j^i. Choosing $\mathsf{r}(x_1, v_j^{i+1})$ to read from t_1 (a) corresponds to setting $s_j = \bot$ and also forces $\mathsf{r}(x_2, v_j^i)$ to read from t_4. Choosing $\mathsf{r}(x_1, v_j^{i+1})$ to read from t_2 (b) corresponds to setting $s_j = \top$ and also forces $\mathsf{r}(x_2, v_j^i)$ to read from t_5. This rf coupling is formalized in Lemma 3.1. The edge numbers specify the order in which rf-edges are inferred.

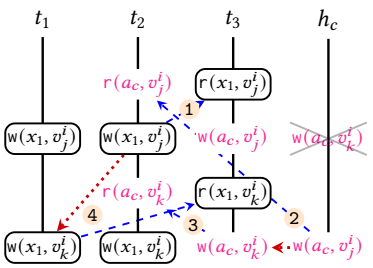
one of $\mathsf{r}(t_3, x_1, v_j^i)$ and $\mathsf{r}(t_3, x_1, v_k^i)$ reads from thread t_2, which corresponds to assigning $\top$ to at most one of s_j and s_k (Figs. 6b to 6d).

The at-least-one-true gadget $\text{ALOne}_{j,k,\ell}^i$. Consider a clause $C_i = (s_j, s_k, s_\ell)$. The at-least-one-true gadget $\text{ALOne}_{j,k,\ell}^i$ is shown in Fig. 7a and contains (i) the nine focal events $\mathsf{w}(t_1, x_1, v_j^i)$, $\mathsf{w}(t_2, x_1, v_j^i)$ and $\mathsf{r}(t_3, x_1, v_j^i)$; $\mathsf{w}(t_1, x_1, v_k^i)$, $\mathsf{w}(t_2, x_1, v_k^i)$ and $\mathsf{r}(t_3, x_1, v_k^i)$; and $\mathsf{w}(t_1, x_1, v_\ell^i)$, $\mathsf{w}(t_2, x_1, v_\ell^i)$ and $\mathsf{r}(t_3, x_1, v_\ell^i)$, and (ii) other events on memory location b. The gadget guarantees that at least one of $\mathsf{r}(t_3, x_1, v_j^i)$, $\mathsf{r}(t_3, x_1, v_k^i)$ and $\mathsf{r}(t_3, x_1, v_\ell^i)$ reads from thread t_2, which corresponds to assigning $\top$ to at least one of s_j, s_k and s_ℓ (shown in Figs. 7b to 7d). Note that this gadget, by itself, allows for two or even all three of the literals to be assigned to $\top$, however, these cases are not shown as they are prohibited by the at-most-one-true gadgets above.

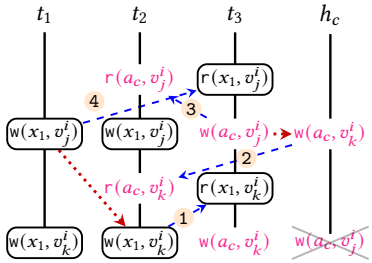
Putting the gadgets together. We serially connect all gadgets in their common threads by po. In particular, $\text{Copy}_{j_1}^{i_1}$ appears before $\text{Copy}_{j_2}^{i_2}$ if $i_1 < i_2$ or $i_1 = i_2$ and $j_1 < j_2$; $\overline{\text{Copy}}_{j_1}^{i_1}$ appears before $\overline{\text{Copy}}_{j_2}^{i_2}$ if $i_1 < i_2$ or $i_1 = i_2$ and $j_1 < j_2$; each $\text{AMOne}[c]_{j_1,k_1}^{i_1}$ appears before $\text{AMOne}[c]_{j_2,k_2}^{i_2}$ if $i_1 < i_2$, and finally $\text{ALOne}_{j_1,k_1,\ell_1}^{i_1}$ appears before $\text{ALOne}_{j_2,k_2,\ell_2}^{i_2}$ if $i_1 < i_2$. As various gadgets have common threads and events, besides connecting them, we also need to specify the interleaving between them. However, this interleaving can be arbitrary and we will not fix it here. Finally, we have indeed used $\kappa = 23$ threads and $d = 14$ memory locations.



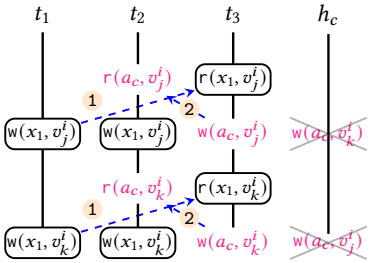
(a) The at-most-one-true gadget $\mathrm{AMOne}[c]^i_{j,k}$.



(b) Resolving with $s_j = \top$ and $s_k = \bot$.



(c) Resolving with $s_j = \bot$ and $s_k = \top$.



(d) Resolving with $s_j = \bot$ and $s_k = \bot$.

Fig. 6. The at-most-one-true gadget $\mathrm{AMOne}[c]^i_{j,k}$ parameterized by $c \in [3]$, and the three ways to resolve it depending on the boolean assignment to s_j and s_k (b, c, d). These rf constraints are formalized in Lemma 3.2. The edge numbers specify the order in which rf-edges are inferred. The crossed-out events are ignored in our analysis for simplicity (see Lemma 3.5).

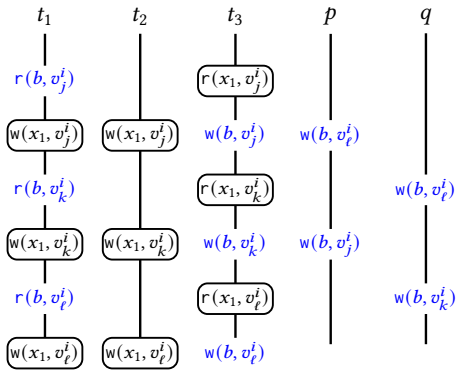
3.2 Soundness

We first establish the soundness of the reduction, i.e., if $\overline{\mathsf{X}}$ is consistent (using an extension X) within Relaxed-Acyclic, then φ has a satisfying assignment. In this direction, we will establish some intermediate lemmas. Recall that we obtain the satisfying assignment for φ by assigning $s_j = \top$ if $(\mathtt{w}(t_2, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$ for all $i \in [m]$, and $s_j = \bot$ if $(\mathtt{w}(t_1, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$ for all $i \in [m]$. The first lemma is based on the copy gadgets Copy^i_j and $\overline{\mathrm{Copy}}^i_j$, and states that each phase of X makes consistent choices for the writer of $\mathtt{r}(t_3, x_1, v_j^i)$ (i.e., whether it reads from t_1 or t_2), which makes the above assignment well-defined. It follows from the high-level description of these two gadgets and the accompanying Fig. 4 and Fig. 5.

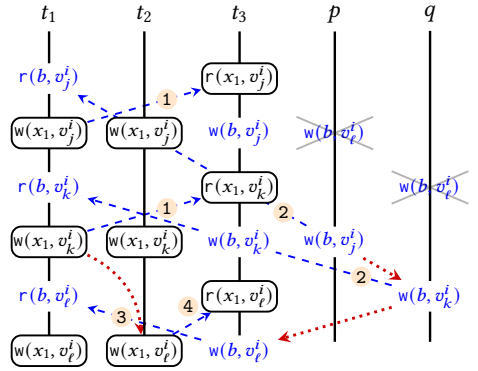
Lemma 3.1. *Let* $\mathsf{X} = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo})$ *be a concrete extension of* $\overline{\mathsf{X}}$ *with* $\mathsf{X} \models$ Relaxed-Acyclic. *For all* $i_1, i_2 \in [m]$, $j \in [n]$, *we have that* $(\mathtt{w}(t_1, x_1, v_j^{i_1}), \mathtt{r}(t_3, x_1, v_j^{i_1})) \in \mathsf{rf}$ *iff* $(\mathtt{w}(t_1, x_1, v_j^{i_2}), \mathtt{r}(t_3, x_1, v_j^{i_2})) \in \mathsf{rf}$.

Proof. We argue by induction that for every $i \in [m-1]$, we have $(\mathtt{w}(t_1, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$ iff $(\mathtt{w}(t_1, x_1, v_j^{i+1}), \mathtt{r}(t_3, x_1, v_j^{i+1})) \in \mathsf{rf}$. First, note that if $(\mathtt{w}(t_1, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$, then the copy gadget Copy^i_j forces $(\mathtt{w}(t_4, x_2, v_j^i), \mathtt{r}(t_6, x_2, v_j^i)) \in \mathsf{rf}$. Indeed, we have the following inferred sequence of rf edges (see 1 - 5 in Fig. 4b, where 1 represents $(\mathtt{w}(t_1, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$).

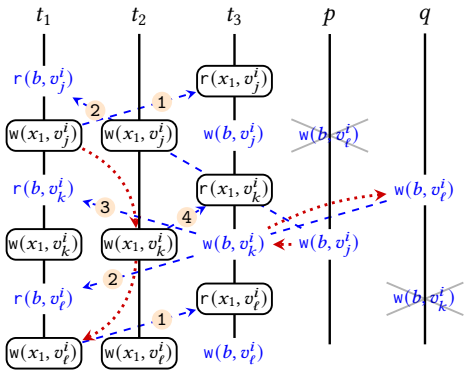
(1) We have $(\mathtt{r}(t_1, y_1, v_j^i), \mathtt{w}(t_3, y_1, v_j^i)) \in (\mathsf{po} \cup \mathsf{rf})^+$ and thus $(\mathtt{w}(t_3, y_1, v_j^i), \mathtt{r}(t_1, y_1, v_j^i)) \notin \mathsf{rf}$ due to porf-acyclicity. Thus $\mathtt{r}(t_1, y_1, v_j^i)$ is forced to read from the only other available write of the same value, i.e., $(\mathtt{w}(f_1, y_1, v_j^i), \mathtt{r}(t_1, y_1, v_j^i)) \in \mathsf{rf}$, depicted as 2 in Fig. 4b.



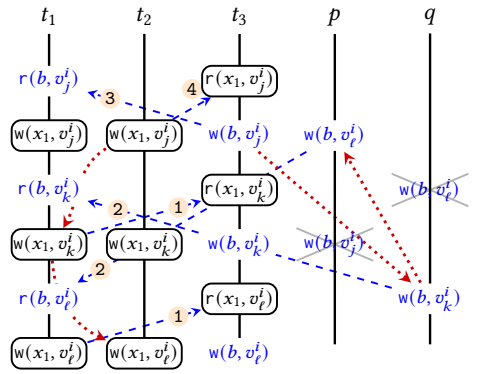
(a) The at-least-one-true gadget $\mathrm{ALOne}^i_{j,k,\ell}$.



(b) Resolving with $s_j = \bot$, $s_k = \bot$ and $s_\ell = \top$.



(c) Resolving with $s_j = \bot$, $s_k = \top$ and $s_\ell = \bot$.



(d) Resolving with $s_j = \top$, $s_k = \bot$ and $s_\ell = \bot$.

Fig. 7. The at-least-one-true gadget $\mathrm{ALOne}^i_{j,k,\ell}$ (a) and the three ways to resolve it depending on the boolean assignment to s_j, s_k and s_ℓ (d, c, b). These rf constraints are formalized in Lemma 3.3. The edge numbers specify the order in which rf-edges are inferred. The crossed-out events are ignored in our analysis for simplicity (see Lemma 3.5).

(2) We now have $(\mathtt{w}(f_1, y_1, v_j^i), \mathtt{r}(t_1, y_1, u_j^i)) \in (\mathrm{po}_{y_1} \cup \mathrm{rf}_{y_1})^+$, and since $(\mathtt{w}(f_2, y_1, u_j^i), \mathtt{r}(t_1, y_1, u_j^i)) \in \mathrm{rf}$, we have $(\mathtt{w}(f_1, y_1, v_j^i), \mathtt{w}(f_2, y_1, u_j^i)) \in \mathrm{mo}$ due to relaxed-read-coherence. Observe that now $(\mathtt{w}(f_1, y_1, \overline{v}_j^i), \mathtt{w}(f_2, y_1, \overline{u}_j^i)) \in (\mathrm{po}_{y_1} \cup \mathrm{mo}_{y_1})^+$, thus due to relaxed-write-coherence, we have $(\mathtt{w}(f_1, y_1, \overline{v}_j^i), \mathtt{w}(f_2, y_1, \overline{u}_j^i)) \in \mathrm{mo}_{y_1}$. Since $(\mathtt{w}(f_2, y_1, \overline{u}_j^i), \mathtt{r}(f_3, y_1, \overline{v}_j^i)) \in (\mathrm{po}_{y_1} \cup \mathrm{rf}_{y_1})^+$, we have $(\mathtt{w}(f_1, y_1, \overline{v}_j^i), \mathtt{r}(f_3, y_1, \overline{v}_j^i)) \notin \mathrm{rf}$ due to relaxed-read-coherence. Thus $\mathtt{r}(f_3, y_1, \overline{v}_j^i)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(t_5, y_1, \overline{v}_j^i), \mathtt{r}(f_3, y_1, \overline{v}_j^i)) \in \mathrm{rf}$, depicted by 3.

(3) We now have $(\mathtt{r}(t_5, y_4, v_j^i), \mathtt{w}(f_3, y_4, v_j^i)) \in (\mathrm{po} \cup \mathrm{rf})^+$ and thus $(\mathtt{w}(f_3, y_4, v_j^i), \mathtt{r}(t_5, y_4, v_j^i)) \notin \mathrm{rf}$, due to porf-acyclicity. Thus $\mathtt{r}(t_5, y_4, v_j^i)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(t_6, y_4, v_j^i), \mathtt{r}(t_5, y_4, v_j^i)) \in \mathrm{rf}$, depicted by 4.

(4) We now have $(\mathtt{r}(t_6, x_2, v_j^i), \mathtt{w}(t_5, x_2, v_j^i)) \in (\mathrm{po} \cup \mathrm{rf})^+$ and thus $(\mathtt{w}(t_5, x_2, v_j^i), \mathtt{r}(t_6, x_2, v_j^i)) \notin \mathrm{rf}$ due to porf-acyclicity. Thus $\mathtt{r}(t_6, x_2, v_j^i)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(t_4, x_2, v_j^i), \mathtt{r}(t_6, x_2, v_j^i)) \in \mathrm{rf}$, depicted by 5.

On the other hand, if $(\mathtt{w}(t_2, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$, then the copy gadget Copy_j^i forces $(\mathtt{w}(t_5, x_2, v_j^i), \mathtt{r}(t_6, x_2, v_j^i)) \in \mathsf{rf}$, by a similar analysis (see Fig. 4c, depicted by 1 - 5).

Finally, a similar analysis on the copy-down gadget $\overline{\mathrm{Copy}}_j^i$ (see Fig. 5, and the forced rf edges 1 - 5) concludes that $(\mathtt{w}(t_1, x_1, v_j^{i+1}), \mathtt{r}(t_3, x_1, v_j^{i+1})) \in \mathsf{rf}$ iff $(\mathtt{w}(t_4, x_2, v_j^i), \mathtt{r}(t_6, x_2, v_j^i)) \in \mathsf{rf}$, and hence we have $(\mathtt{w}(t_1, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$ iff $(\mathtt{w}(t_1, x_1, v_j^{i+1}), \mathtt{r}(t_3, x_1, v_j^{i+1})) \in \mathsf{rf}$, as desired. □

The next lemma is based on the at-most-one-true gadgets $\mathrm{AMOne}[c]_{j,k}^i$, and it is used to show that for every clause C_i and for each of the three pairs of literals (s_j, s_k) in C_i, at most one of them is assigned to true. Again, it follows by a direct analysis of the accompanying figure, Fig. 6.

Lemma 3.2. *Let* $\mathsf{X} = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo})$ *be a concrete extension of* $\overline{\mathsf{X}}$ *with* $\mathsf{X} \models$ Relaxed-Acyclic. *For every* $i \in [m]$ *and* $j, k \in [n]$ *such that* s_j *and* s_k *appear in clause* C_i*, we have* $(\mathtt{w}(t_2, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \notin \mathsf{rf}$ *or* $(\mathtt{w}(t_2, x_1, v_k^i), \mathtt{r}(t_3, x_1, v_k^i)) \notin \mathsf{rf}$.

Proof. The statement follows by analyzing the at-most-one-true gadget $\mathrm{AMOne}[c]_{j,k}^i$, where $c \in [3]$ is such that (s_j, s_k) is the c-th pair of variables in C_i (see Fig. 6).

First, if $(\mathtt{w}(t_2, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$ holds (marked 1 in Fig. 6b) then $(\mathtt{w}(t_2, x_1, v_k^i), \mathtt{r}(t_3, x_1, v_k^i)) \notin \mathsf{rf}$. Indeed, we have the following sequence of rf edges (see 1 - 4, Fig. 6b).

(1) We have $(\mathtt{r}(t_2, a_c, v_j^i), \mathtt{w}(t_3, a_c, v_j^i)) \in (\mathsf{po} \cup \mathsf{rf})^+$ and thus $(\mathtt{w}(t_3, a_c, v_j^i), \mathtt{r}(t_2, a_c, v_j^i)) \notin \mathsf{rf}$ due to porf-acyclicity. Thus $\mathtt{r}(t_2, a_c, v_j^i)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(h_c, a_c, v_j^i), \mathtt{r}(t_2, a_c, v_j^i)) \in \mathsf{rf}$, depicted by 2.
(2) We now have $(\mathtt{w}(h_c, a_c, v_j^i), \mathtt{r}(t_2, a_c, v_k^i)) \in (\mathsf{po}_{a_c} \cup \mathsf{rf}_{a_c})^+$, and due to relaxed-write-coherence, we also have $(\mathtt{w}(h_c, a_c, v_k^i), \mathtt{w}(h_c, a_c, v_j^i)) \in \mathsf{mo}_{a_c}$. In turn, this implies $(\mathtt{w}(h_c, a_c, v_k^i), \mathtt{r}(t_2, a_c, v_k^i)) \notin \mathsf{rf}$ due to relaxed-read-coherence. Thus $\mathtt{r}(t_2, a_c, v_k^i)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(t_3, a_c, v_k^i), \mathtt{r}(t_2, a_c, v_k^i)) \in \mathsf{rf}$, depicted by 3.
(3) We now have $(\mathtt{r}(t_3, x_1, v_k^i), \mathtt{w}(t_2, x_1, v_k^i)) \in (\mathsf{po} \cup \mathsf{rf})^+$ and thus $(\mathtt{w}(t_2, x_1, v_k^i), \mathtt{r}(t_3, x_1, v_k^i)) \notin \mathsf{rf}$ due to porf-acyclicity. Thus $\mathtt{r}(t_3, x_1, v_k^i)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(t_1, x_1, v_k^i), \mathtt{r}(t_3, x_1, v_k^i)) \in \mathsf{rf}$, depicted by 4.

Second, if $(\mathtt{w}(t_2, x_1, v_k^i), \mathtt{r}(t_3, x_1, v_k^i)) \in \mathsf{rf}$ marked 1 in Fig. 6c), then $(\mathtt{w}(t_2, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \notin \mathsf{rf}$. Indeed, we have the following forced sequence of rf edges (see Fig. 6c).

(1) We have $(\mathtt{r}(t_2, a_c, v_k^i), \mathtt{w}(t_3, a_c, v_k^i)) \in (\mathsf{po} \cup \mathsf{rf})^+$ and thus $(\mathtt{w}(t_3, a_c, v_k^i), \mathtt{r}(t_2, a_c, v_k^i)) \notin \mathsf{rf}$ due to porf-acyclicity. Thus $\mathtt{r}(t_2, a_c, v_k^i)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(h_c, a_c, v_k^i), \mathtt{r}(t_2, a_c, v_k^i)) \in \mathsf{rf}$, marked 2.
(2) Due to relaxed-write-coherence, we have $(\mathtt{w}(h_c, a_c, v_k^i), \mathtt{w}(h_c, a_c, v_j^i)) \in \mathsf{mo}_{a_c}$ We thus have $(\mathtt{w}(h_c, a_c, v_j^i), \mathtt{r}(t_2, a_c, v_j^i)) \notin \mathsf{rf}$, as this would imply that $(\mathtt{w}(h_c, a_c, v_j^i), \mathtt{r}(t_2, a_c, v_k^i)) \in (\mathsf{po}_{a_c} \cup \mathsf{rf}_{a_c})$, which would violate relaxed-read-coherence. Thus $\mathtt{r}(t_2, a_c, v_j^i)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(t_3, a_c, v_j^i), \mathtt{r}(t_2, a_c, v_j^i)) \in \mathsf{rf}$, marked 3.
(3) We now have $(\mathtt{r}(t_3, x_1, v_j^i), \mathtt{w}(t_2, x_1, v_j^i)) \in (\mathsf{po} \cup \mathsf{rf})^+$ and thus $(\mathtt{w}(t_2, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \notin \mathsf{rf}$ due to porf-acyclicity. Thus $\mathtt{r}(t_3, x_1, v_j^i)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(t_1, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$, depicted by 4. □

The third lemma is based on the at-least-one-true gadget $\mathrm{ALOne}^i_{j,k,\ell}$, and it is used to show that for every clause $C_i = (s_j, s_k, s_\ell)$, at least one of its literals is assigned to true. Again, it follows by a direct analysis of the accompanying figure, Fig. 7.

Lemma 3.3. *Let* $\mathsf{X} = (\mathsf{E}, \mathrm{po}, \mathrm{rf}, \mathrm{mo})$ *be a concrete extension of* $\overline{\mathsf{X}}$ *with* $\mathsf{X} \models$ Relaxed-Acyclic. *For every* $i \in [m]$ *and* $j, k, \ell \in [n]$ *such that* s_j, s_k *and* s_ℓ *appear in clause* C_i, *we have* $(\mathtt{w}(t_2, x_1, v^i_j), \mathtt{r}(t_3, x_1, v^i_j)) \in \mathrm{rf}$ *or* $(\mathtt{w}(t_2, x_1, v^i_k), \mathtt{r}(t_3, x_1, v^i_k)) \in \mathrm{rf}$ *or* $(\mathtt{w}(t_2, x_1, v^i_\ell), \mathtt{r}(t_3, x_1, v^i_\ell)) \in \mathrm{rf}$.

Proof. First, if $(\mathtt{w}(t_2, x_1, v^i_j), \mathtt{r}(t_3, x_1, v^i_j)) \notin \mathrm{rf}$ and $(\mathtt{w}(t_2, x_1, v^i_k), \mathtt{r}(t_3, x_1, v^i_k)) \notin \mathrm{rf}$ (hence both read from t_1, marked 1 in Fig. 7b) then $(\mathtt{w}(t_2, x_1, v^i_\ell), \mathtt{r}(t_3, x_1, v^i_\ell)) \in \mathrm{rf}$. Indeed, we have the following forced sequence of rf edges (see Fig. 7b, 1 - 4).

(1) We have $(\mathtt{r}(t_1, b, v^i_j), \mathtt{w}(t_3, b, v^i_j)) \in (\mathrm{po} \cup \mathrm{rf})^+$ and thus $(\mathtt{w}(t_3, b, v^i_j), \mathtt{r}(t_1, b, v^i_j)) \notin \mathrm{rf}$ due to porf-acyclicity. Thus $\mathtt{r}(t_1, b, v^i_j)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(p, b, v^i_j), \mathtt{r}(t_1, b, v^i_j)) \in \mathrm{rf}$. Similarly, we have $(\mathtt{r}(t_1, b, v^i_k), \mathtt{w}(t_3, b, v^i_k)) \in (\mathrm{po} \cup \mathrm{rf})^+$ and thus $(\mathtt{w}(t_3, b, v^i_k), \mathtt{r}(t_1, b, v^i_k)) \notin \mathrm{rf}$ due to porf-acyclicity. Thus $\mathtt{r}(t_1, b, v^i_k)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(q, b, v^i_k), \mathtt{r}(t_1, b, v^i_k)) \in \mathrm{rf}$. These two edges are depicted 2.

(2) We now have $(\mathtt{w}(p, b, v^i_j), \mathtt{r}(t_1, b, v^i_\ell)) \in (\mathrm{po}_b \cup \mathrm{rf}_b)^+$, and due to relaxed-write-coherence, we also have $(\mathtt{w}(p, b, v^i_\ell), \mathtt{w}(p, b, v^i_j)) \in \mathrm{mo}_b$. In turn, this implies $(\mathtt{w}(p, b, v^i_\ell), \mathtt{r}(t_1, b, v^i_\ell)) \notin \mathrm{rf}$ due to relaxed-read-coherence. Similarly, we now have $(\mathtt{w}(q, b, v^i_j), \mathtt{r}(t_1, b, v^i_\ell)) \in (\mathrm{po}_b \cup \mathrm{rf}_b)^+$, and due to relaxed-write-coherence, we also have $(\mathtt{w}(q, b, v^i_\ell), \mathtt{w}(q, b, v^i_j)) \in \mathrm{mo}_b$. In turn, this implies $(\mathtt{w}(q, b, v^i_\ell), \mathtt{r}(t_1, b, v^i_\ell)) \notin \mathrm{rf}$ due to relaxed-read-coherence. Thus $\mathtt{r}(t_1, b, v^i_\ell)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(t_3, b, v^i_\ell), \mathtt{r}(t_1, b, v^i_\ell)) \in \mathrm{rf}$, depicted 3.

(3) We now have $(\mathtt{r}(t_3, x_1, v^i_\ell), \mathtt{w}(t_1, x_1, v^i_\ell)) \in (\mathrm{po} \cup \mathrm{rf})^+$ and thus $(\mathtt{w}(t_1, x_1, v^i_\ell), \mathtt{r}(t_3, x_1, v^i_\ell)) \notin \mathrm{rf}$ due to porf-acyclicity. Thus $\mathtt{r}(t_3, x_1, v^i_\ell)$ is forced to read from the only other available write, i.e., $(\mathtt{w}(t_2, x_1, v^i_\ell), \mathtt{r}(t_3, x_1, v^i_\ell)) \in \mathrm{rf}$, depicted by 4.

A similar analysis establishes that if $(\mathtt{w}(t_2, x_1, v^i_j), \mathtt{r}(t_3, x_1, v^i_j)) \notin \mathrm{rf}$ and $(\mathtt{w}(t_2, x_1, v^i_\ell), \mathtt{r}(t_3, x_1, v^i_\ell)) \notin \mathrm{rf}$ then $(\mathtt{w}(t_2, x_1, v^i_k), \mathtt{r}(t_3, x_1, v^i_k)) \in \mathrm{rf}$ (see Fig. 7c), as well as that if $(\mathtt{w}(t_2, x_1, v^i_k), \mathtt{r}(t_3, x_1, v^i_k)) \notin \mathrm{rf}$ and $(\mathtt{w}(t_2, x_1, v^i_\ell), \mathtt{r}(t_3, x_1, v^i_\ell)) \notin \mathrm{rf}$ then $(\mathtt{w}(t_2, x_1, v^i_j), \mathtt{r}(t_3, x_1, v^i_j)) \in \mathrm{rf}$ (see Fig. 7d). □

Lemma 3.1 states that our truth assignment for φ is valid, while Lemma 3.2 and Lemma 3.3 guarantee that in every clause, exactly one literal is set to true. Hence we have the following corollary.

Corollary 3.4. *If* $\overline{\mathsf{X}} \models$ Relaxed-Acyclic *then* φ *is satisfiable.*

3.3 Completeness

We now turn our attention to the completeness property, i.e., if φ is satisfiable then $\overline{\mathsf{X}}$ is consistent in the Relaxed-Acyclic model. To this end, we construct a reads-from relation rf and a partial modification order $\overline{\mathrm{mo}}$ as follows.

(1) For each gadget, we insert rf-edges according to Figs. 4 to 7 and the truth assignments on literals s_j, s_k, s_ℓ involved in that gadget. In particular, this implies that, for each $i \in [m]$ and $j \in [n]$, we have $(\mathtt{w}(t_1, x_1, v^i_j), \mathtt{r}(t_3, x_1, v^i_j)) \in \mathrm{rf}$ if $s_j = \bot$ and $(\mathtt{w}(t_2, x_1, v^i_j), \mathtt{r}(t_3, x_1, v^i_j)) \in \mathrm{rf}$ if $s_j = \top$. Observe that rf is fully specified, i.e., every read has been assigned a write.

(2) For every two conflicting writes $\mathtt{w}_1, \mathtt{w}_2$ with (i) $\mathtt{tid}(\mathtt{w}_1) \neq \mathtt{tid}(\mathtt{w}_2)$, and (ii) there exist two reads $\mathtt{r}_1, \mathtt{r}_2$ with $(\mathtt{r}_1, \mathtt{r}_2) \in \mathrm{po}$ such that $(\mathtt{w}_i, \mathtt{r}_i) \in \mathrm{rf}$ for each $i \in [2]$, we have $(\mathtt{w}_1, \mathtt{w}_2) \in \overline{\mathrm{mo}}$.

We call a triplet $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ on location x *safe* in Relaxed-Acyclic if either $(\mathtt{w}', \mathtt{r}) \notin (\mathrm{po}_x \cup \mathrm{rf}_x)^+$ or $(\mathtt{w}', \mathtt{w}) \in (\mathrm{po}_x \cup \mathrm{rf}_x \cup \overline{\mathrm{mo}}_x)^+$. To prove the consistency of $\overline{\mathrm{X}}$ it suffices to prove the following:

(1) $(\mathrm{po} \cup \mathrm{rf})$ is acyclic, and
(2) $\overline{\mathrm{mo}}$ is *minimally coherent* for Relaxed-Acyclic, namely, that (i) $(\mathrm{po}_x \cup \mathrm{rf}_x \cup \overline{\mathrm{mo}}_x)$ is acyclic for every location x, and (ii) every triplet is safe.

Indeed, minimal-coherence guarantees that, for each location x, there exists a total extension mo_x of $\overline{\mathrm{mo}}_x$ that satisfies relaxed-write-coherence and relaxed-read-coherence [Tunç et al. 2023].

To simplify our analysis, we ignore the writes in threads $\{h_c\}_{c\in[3]}$, p and q that are not read in rf (crossed-out in Fig. 6 and Fig. 7). This allows us to make our formal statements more uniform, while this omission does not affect the correctness of the analysis. Indeed, let InactiveWr $= \{\mathtt{w} \in \mathrm{W} \mid \mathrm{tid}(\mathtt{w}) \in \{h_1, h_2, h_3, p, q\} \text{ and } \nexists \mathtt{r} \in \mathrm{R}.\ (\mathtt{w}, \mathtt{r}) \in \mathrm{rf}\}$. The following lemma is straightforward.

Lemma 3.5. *The following statements hold.*

(1) If there is a $(\mathrm{po} \cup \mathrm{rf})$*-cycle then there is such a cycle without any write in* InactiveWr.
(2) If there is a $(\mathrm{po}_x \cup \mathrm{rf}_x \cup \overline{\mathrm{mo}}_x)$*-cycle for some memory location* x *then there is such a cycle without any write in* InactiveWr.
(3) If there is an unsafe triplet, then there is such a triplet $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ *where* $\mathtt{w}' \notin$ InactiveWr.

Proof. We prove each item separately.

(1) We first observe that for any $\mathtt{w} \in$ InactiveWr, there is no outgoing rf or even $\overline{\mathrm{mo}}$ edge. Hence any cycle containing a write $\mathtt{w} \in$ InactiveWr must contain a sequence of edges $e_1 \xrightarrow{\mathrm{po}} \mathtt{w} \xrightarrow{\mathrm{po}} e_2$, which can be replaced by the single edge $e_1 \xrightarrow{\mathrm{po}} e_2$.
(2) Proof same as above.
(3) Consider a memory location x and an unsafe triplet $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ on x. This means that there is a $(\mathrm{po}_x \cup \mathrm{rf}_x)$-path $P\colon \mathtt{w}' \overset{\mathrm{po}_x \cup \mathrm{rf}_x}{\rightsquigarrow} \mathtt{r}$. Since the threads $\{h_c\}_{c\in[3]}$, p and q contain only same-location writes, P must be of the form $P\colon \mathtt{w}' \overset{\mathrm{po}_x}{\rightsquigarrow} \mathtt{w}'' \xrightarrow{\mathrm{rf}_x} \mathtt{r}'' \overset{\mathrm{po}_x \cup \mathrm{rf}_x}{\rightsquigarrow} \mathtt{r}$,where, $\mathtt{w}''$ and $\mathtt{r}''$ conflict with $\mathtt{w}'$. Note that $\mathtt{r}'' \neq \mathtt{r}$, otherwise $\mathtt{w}'' = \mathtt{w}$ since $(\mathtt{w}, \mathtt{r}) \in \mathrm{rf}$, implying that $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ is safe. Since $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ is unsafe, we have $(\mathtt{w}', \mathtt{w}) \notin (\mathrm{po}_x \cup \mathrm{rf}_x \cup \overline{\mathrm{mo}}_x)^+$ and hence, $(\mathtt{w}'', \mathtt{w}) \notin (\mathrm{po}_x \cup \mathrm{rf}_x \cup \overline{\mathrm{mo}}_x)^+$. Thus $(\mathtt{w}, \mathtt{r}, \mathtt{w}'')$ is also unsafe, while clearly $\mathtt{w}'' \notin$ InactiveWr. □

The completeness can now be established in three lemmas, one for each of the properties (1), (2i), and (2ii) above (i.e., $(\mathrm{po} \cup \mathrm{rf})$-acyclicity and minimal coherence). Before we proceed, we will use the following notation to make our analysis easier.

Notation. Given an event e, we say that e *appears in phase* i, written $\mathrm{phase}(e) = i$, if it writes/reads a value superscripted by i (i.e., of the form v^i or $\overline{v}^i$). Similarly, we say *that* e *appears in step* j, written $\mathrm{step}(e) = j$, if it writes/reads a value subscripted by j (i.e., of the form v_j or $\overline{v}_j$). We define a quasi order $\preceq$ on the event set E with $e_1 \preceq e_2$ if either (i) $\mathrm{phase}(e_1) < \mathrm{phase}(e_2)$ or (ii) $\mathrm{phase}(e_1) = \mathrm{phase}(e_2)$ and $\mathrm{step}(e_1) \leq \mathrm{step}(e_2)$. We also write $e_1 \prec e_2$ to denote that $e_1 \preceq e_2$ and $e_2 \not\preceq e_1$. Now consider an arbitrary path $P = e_1, \ldots, e_k$. We say that P *crosses* a thread t if $\mathrm{tid}(e_\ell)$ for some for some $\ell \in [k]$. We call P *monotonic* if it linearizes $\preceq$, i.e., $e_\ell \preceq e_{\ell+1}$ for all $\ell \in [k-1]$.

We first establish the safety of each triplet.

Lemma 3.6. *Every triplet* $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ *is safe.*

Proof. First, observe that the following hold by construction.

(1) There is no memory location that is both written and read by the same thread.
(2) For every two conflicting writes $\mathtt{w}_1, \mathtt{w}_2$, if there exist two reads $\mathtt{r}_1, \mathtt{r}_2$ such that $(\mathtt{w}_i, \mathtt{r}_i) \in \mathsf{rf}$ for each $i \in [2]$ and $(\mathtt{r}_1, \mathtt{r}_2) \in \mathsf{po}$, then $(\mathtt{w}_2, \mathtt{w}_1) \notin \mathsf{po}$.

Now, let x be the location accessed by a triplet $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$, and assume there exists a $(\mathsf{po}_x \cup \mathsf{rf}_x)$-path $P\colon \mathtt{w}' \overset{\mathsf{po}_x \cup \mathsf{rf}_x}{\rightsquigarrow} \mathtt{r}$. Due to Item (1), P must leave a thread and enter another at most once. Thus, P must be of the form $P\colon \mathtt{w}' \overset{\mathsf{po}_x^?}{\rightsquigarrow} \mathtt{w}'' \overset{\mathsf{rf}_x}{\longrightarrow} \mathtt{r}'' \overset{\mathsf{po}_x^?}{\rightsquigarrow} \mathtt{r}$. If $(\mathtt{w}'', \mathtt{w}) \in \mathsf{po}_x^?$, we are done. Otherwise, due to Item (2), we can't have that $(\mathtt{w}, \mathtt{w}'') \in \mathsf{po}_x$. Hence, $\mathtt{w}$ and $\mathtt{w}''$ are in different threads, and by the construction of $\overline{\mathsf{mo}}_x$, we have $(\mathtt{w}'', \mathtt{w}) \in \overline{\mathsf{mo}}_x$, implying that $(\mathtt{w}', \mathtt{w}) \in (\mathsf{po}_x \cup \overline{\mathsf{mo}}_x)^+$. Hence the triplet is safe, as desired. □

Next, we establish the second ingredient of completeness, i.e., the acyclicity of $(\mathsf{po} \cup \mathsf{rf})$.

Lemma 3.7. *$(\mathsf{po} \cup \mathsf{rf})$ is acyclic.*

Proof. First, note that there is no $(\mathsf{po} \cup \mathsf{rf})$-cycle crossing any of g_2 and g_5, as these threads only contain writes, and hence there is no way to enter them by an rf-edge. Moreover, by construction, any $(\mathsf{po} \cup \mathsf{rf})$-path P that starts from a thread other than g_2 and g_5 is necessarily monotonic.

Indeed, all rf-edges connect events of the same phase and step, while the only po-edges $e_1 \overset{\mathsf{po}}{\longrightarrow} e_2$ with $\mathrm{phase}(e_1) < \mathrm{phase}(e_2)$ appear in g_2 and g_5. Finally, the only po-edges $e_1 \overset{\mathsf{po}}{\longrightarrow} e_2$ with $\mathrm{phase}(e_1) = \mathrm{phase}(e_2)$ and $\mathrm{step}(e_1) > \mathrm{step}(e_2)$ occur in threads h_1, h_2, h_3, p and q (see Fig. 6 and Fig. 7). However, in light of Lemma 4.5, these edges can be ignored in our analysis.

Thus, any potential $(\mathsf{po} \cup \mathsf{rf})$-cycle has to traverse events of the same phase and step. A straightforward analysis of each instantiation of each gadget (see Figs. 4b and 4c, Figs. 5a and 5b, Figs. 6b to 6d, Figs. 7b to 7d), and their combinations on common events, establishes that no $(\mathsf{po} \cup \mathsf{rf})$-cycle exists, as desired. □

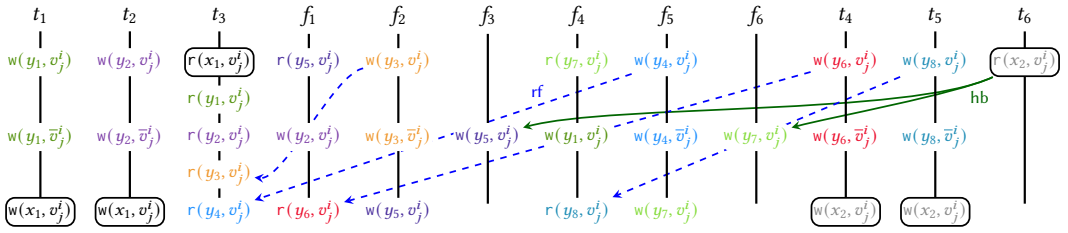
Finally, we establish the acyclicity of $(\mathsf{po}_x \cup \mathsf{rf}_x \cup \overline{\mathsf{mo}}_x)$.

Lemma 3.8. *$(\mathsf{po}_x \cup \mathsf{rf}_x \cup \overline{\mathsf{mo}}_x)$ is acyclic for all locations x.*

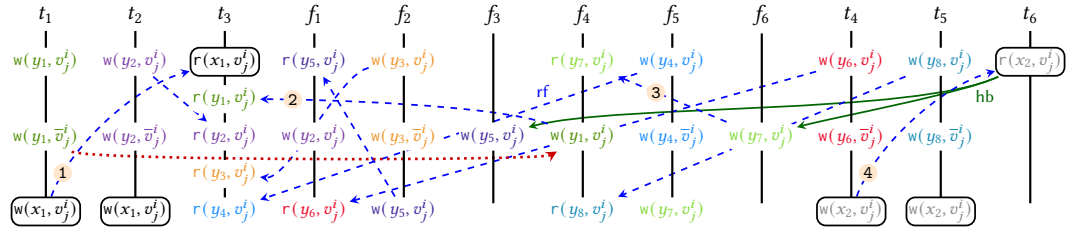
Proof. Observe that no memory location is both read and written by the same thread in any of the gadgets. Hence, any $(\mathsf{po}_x \cup \mathsf{rf}_x \cup \overline{\mathsf{mo}}_x)$-cycle C following an rf-edge $\mathtt{w} \overset{\mathsf{rf}}{\longrightarrow} \mathtt{r}$ enters a thread that it cannot exit. This means that C can only contain events of the same thread, which is forbidden by Lemma 3.7. Thus we only need to reason about the absence of $(\mathsf{po}_x \cup \overline{\mathsf{mo}}_x)$ cycles.

Observe that for every memory location x except z_1 and z_2, every $(\mathsf{po}_x \cup \overline{\mathsf{mo}}_x)$-path is monotonic. Hence any potential $(\mathsf{po}_x \cup \overline{\mathsf{mo}}_x)$-cycle must traverse paths of the same phase and step, and thus be contained in one of the gadgets. The absence of such cycles can be easily established by looking at the instantiations of these gadgets (Figs. 4 to 7).

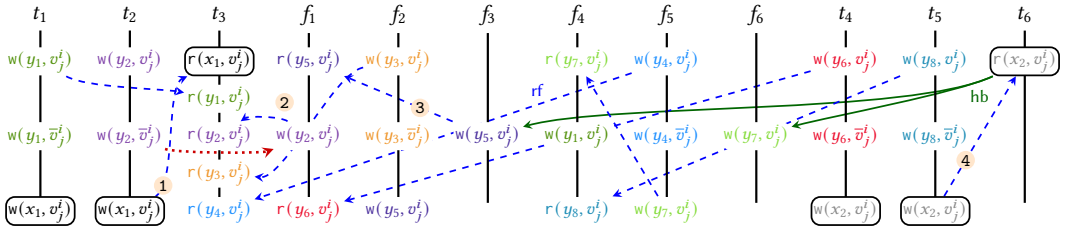
On the other hand, consider the case $x = z_1$ (the analysis is similar for $x = z_2$). Let C be a shortest $(\mathsf{po}_{z_1} \cup \overline{\mathsf{mo}}_{z_1})$-cycle. If C contains only events of the same phase and step, it can be dismissed again by looking at the gadget in Fig. 5. Otherwise, C is non-monotonic and must thus traverse the non-monotonic edge $\mathtt{w}(g_2, z_1, u_j^{i+1}) \overset{\mathsf{po}_{z_1}}{\longrightarrow} \mathtt{w}(g_2, z_1, \overline{u}_j^{i})$. From there it can continue to threads g_1



(a) The copy gadget Copy^i_j.



(b) Choosing $\mathsf{r}(t_3, x_1, v^i_j)$ to read from t_1 forces the sequence of rf and mo edges shown.



(c) Choosing $\mathsf{r}(t_3, x_1, v^i_j)$ to read from t_2 forces the sequence of rf and mo edges shown.

Fig. 8. The copy gadget Copy^i_j (a) captures the Boolean assignment to variable s_j in phase i. There are two ways to realize this gadget, by choosing which of the two writes $\mathsf{w}(x_1, v^i_j)$ the read $\mathsf{r}(x_1, v^i_j)$ observes. Choosing the write of t_1 (b) corresponds to setting $s_j = \bot$ and also forces $\mathsf{r}(x_2, v^i_j)$ to read from t_4. Choosing the write of t_2 (c) corresponds to setting $s_j = \top$ and also forces $\mathsf{r}(x_2, v^i_j)$ to read from t_5. This rf coupling is formalized in Lemma 4.1. The edge numbers specify the order in which rf-edges are inferred.

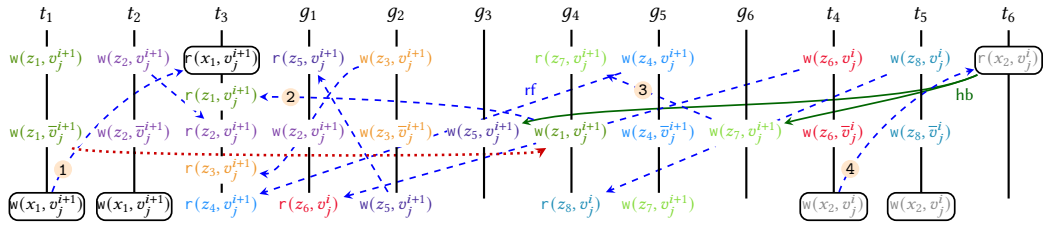
and thread t_5, following events e with $\mathsf{w}(g_2, z_1, \overline{u}^i_j) \preceq e$. It can be easily verified on Fig. 5 that the next time C crosses g_2, it is on an event e' with $(\mathsf{w}(g_2, z_1, u^{i+1}_j), e') \in \text{po}_{z_1}$, which contradicts the fact that C is a shortest cycle. The desired result follows. □

Lemma 3.6, Lemma 3.7 and Lemma 3.8 imply the completeness of the reduction.

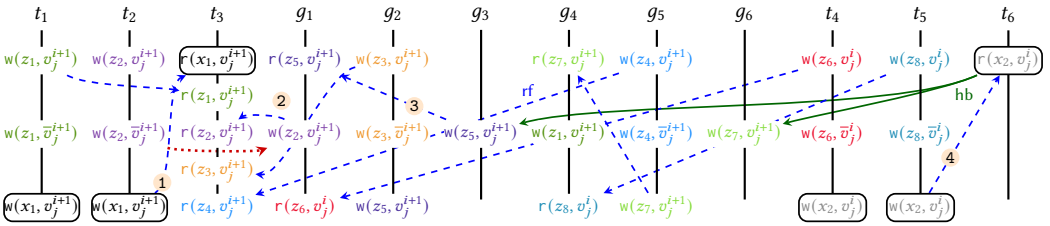
Corollary 3.9. *If φ is satisfiable then $\overline{\text{X}} \models$ Relaxed-Acyclic.*

4 HARDNESS FOR WRA, RA AND SRA

In this section we prove Theorem 1.1 for the range of models SRA $\preccurlyeq \mathcal{M} \preccurlyeq$ WRA, which also implies hardness specifically for SRA, RA, and WRA. Similarly to Section 3, our reduction uses unbounded values. Again, we will make the final step towards Theorem 1.1 in Section 6, which consists of a simple modification of the technique presented here.



(a) Choosing $\mathsf{r}(t_3, x_1, v_j^{i+1})$ to observe from t_1 forces the sequence of rf and mo edges shown.



(b) Choosing $\mathsf{r}(t_3, x_1, v_j^{i+1})$ to read from t_2 forces the sequence of rf and mo edges shown.

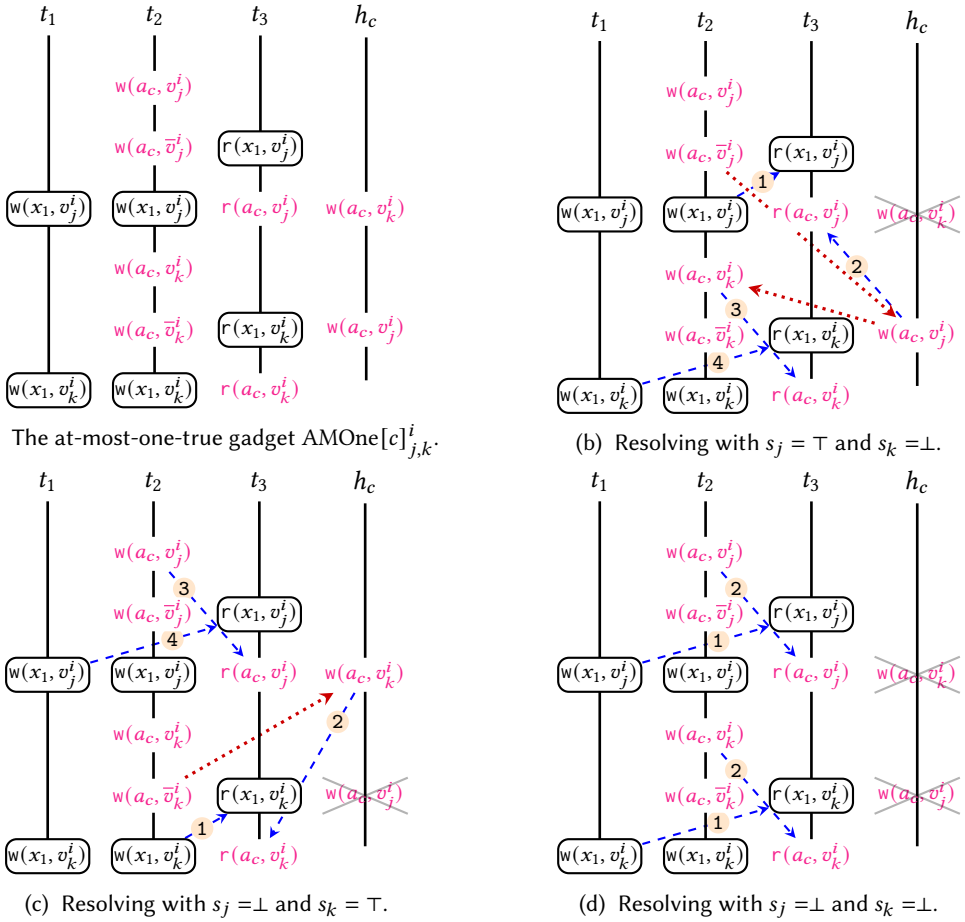
Fig. 9. The copy-down gadget $\overline{\text{Copy}}_j^i$ is very similar to Copy_j^i. Choosing $\mathsf{r}(x_1, v_j^{i+1})$ to read from t_1 (a) corresponds to setting $s_j = \bot$ and also forces $\mathsf{r}(x_2, v_j^i)$ to read from t_4. Choosing $\mathsf{r}(x_1, v_j^{i+1})$ to read from t_2 (b) corresponds to setting $s_j = \top$ and also forces $\mathsf{r}(x_2, v_j^i)$ to read from t_5. This rf coupling is formalized in Lemma 4.1. The edge numbers specify the order in which rf-edges are inferred.

4.1 Reduction

Given a monotone formula $\varphi = \{C_i\}_{i\in[m]}$ over n variables $\{s_j\}_{j\in[n]}$, we construct an abstract execution $\overline{\mathsf{X}} = (\mathsf{E}, \mathsf{po})$ with $\kappa = 23$ threads and accessing $d = 26$ memory locations such that φ is satisfiable iff $\overline{\mathsf{X}} \models \mathcal{M}$, for any memory model $\mathcal{M}$ such that $\mathsf{SRA} \preccurlyeq \mathcal{M} \preccurlyeq \mathsf{WRA}$. $\overline{\mathsf{X}}$ follows the general scheme of Fig. 3. Again, in phase i and step j we have a read event $\mathsf{r}(t_3, x_1, v_j^i)$ that can read from either of two writes $\mathsf{w}(t_1, x_1, v_j^i)$ (corresponding to $s_j = \bot$) or $\mathsf{w}(t_2, x_1, v_j^i)$ (corresponding to $s_j = \top$). We use the same types of gadgets as in the previous section to force the desirable interaction patterns between threads. However, the contents of each gadget are different to account for the different memory models. In particular, these gadgets now rely on the weak-read-coherence axiom to couple the readers in threads t_3 and t_6 and force the 1-in-3 property in each clause C_i, as opposed to the porf-acyclicity and relaxed-read-coherence axioms used in Section 3 for Relaxed-Acyclic.

The copy gadget Copy_j^i. The copy gadget Copy_j^i (Fig. 8) guarantees that $\mathsf{r}(t_3, x_1, v_j^i)$ reads from thread t_1 iff $\mathsf{r}(t_6, x_2, v_j^i)$ reads from thread t_4. Besides x_1 and x_2, this gadget also uses locations y_1, y_2, y_3, y_4, y_5, y_6, y_7 and y_8. Moreover, the gadget also contains two hb-edges out of $\mathsf{r}(t_6, x_2, v_j^i)$. Though not shown explicitly, these hb-edges can be easily enforced by an rf-edge $\mathsf{w} \xrightarrow{\mathsf{rf}} \mathsf{r}$, where w and r access a new memory location. These events are independent to our analysis later, and will thus be ignored, i.e., we will only be considering the hb-edges as they appear in the gadget.

The copy-down gadget $\overline{\text{Copy}}_j^i$. The copy-down gadget $\overline{\text{Copy}}_j^i$ (Fig. 9), as before, has a similar structure to the copy gadget Copy_j^i and guarantees that $\mathsf{r}(t_3, x_1, v_j^{i+1})$ reads from thread t_1 iff $\mathsf{r}(t_6, x_2, v_j^i)$ reads from thread t_4. Together, the two copy gadgets Copy_j^i and $\overline{\text{Copy}}_j^i$ ensure that $\mathsf{r}(t_3, x_1, v_j^i)$ reads from thread t_1 iff $\mathsf{r}(t_3, x_1, v_j^{i+1})$ reads from thread t_1, guaranteeing a valid truth assignment on s_j. Besides x_1 and x_2, this gadget also uses locations z_1, z_2, z_3, z_4, z_5, z_6, z_7 and z_8.



(a) The at-most-one-true gadget $\text{AMOne}[c]^i_{j,k}$.

(b) Resolving with $s_j = \top$ and $s_k = \bot$.

(c) Resolving with $s_j = \bot$ and $s_k = \top$.

(d) Resolving with $s_j = \bot$ and $s_k = \bot$.

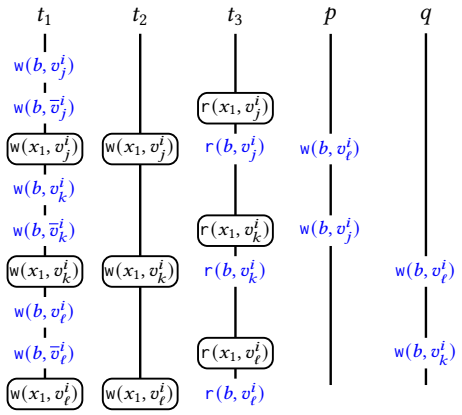
Fig. 10. The at-most-one-true gadget $\text{AMOne}[c]^i_{j,k}$ parameterized by $c \in [3]$, and the three ways to resolve it depending on the boolean assignment to s_j and s_k (b, c, d). These rf constraints are formalized in Lemma 4.2. The edge numbers specify the order in which rf-edges are implied. The crossed-out events are ignored in our analysis for simplicity (see Lemma 4.5).

Moreover, the gadget also contains two hb-edges out of $r(t_6, x_2, v^i_j)$, which, as argued above, will be ignored in our analysis.

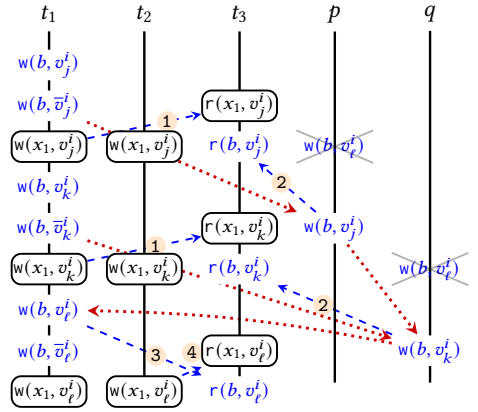
The at-most-one-true gadgets $\text{AMOne}[c]^i_{j,k}$. For each $c \in [3]$, the at-most-one-true gadget (Fig. 10) guarantees that for every clause C_i, the c-th pair of literals (s_j, s_k) in C_i is such that at most one of s_j and s_k is set to true. For each $c \in [3]$, the corresponding gadget contains one additional memory location a_c.

The at-least-one-true gadget $\text{ALOne}^i_{j,k,\ell}$. The at-least-one-true gadget (Fig. 11) guarantees that for every clause C_i with three literals s_j, s_k, s_ℓ, at least one of them is set to true. It contains one additional memory location b.

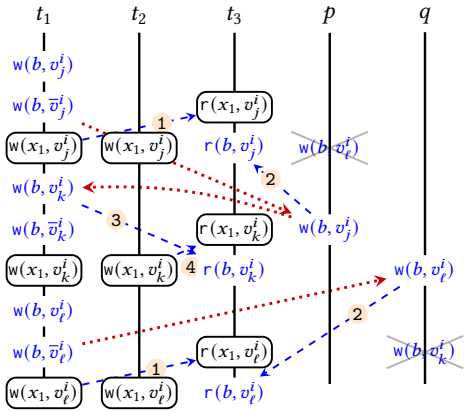
Putting the gadgets together. We obtain the abstract execution $\overline{\text{X}}$ by appropriately connecting all gadgets and specifying the interleaving of events in common threads.



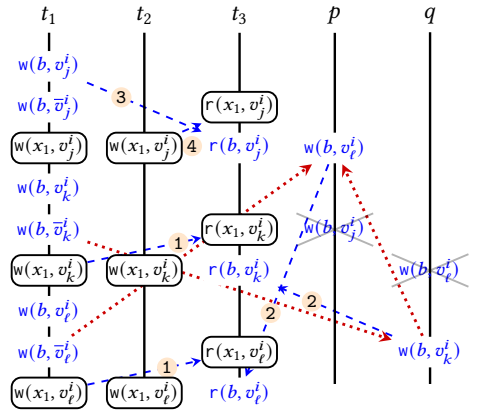
(a) The at-least-one-true gadget $\mathrm{ALOne}^i_{j,k,\ell}$.



(b) Resolving with $s_j = \bot$, $s_k = \bot$ and $s_\ell = \top$.



(c) Resolving with $s_j = \bot$, $s_k = \top$ and $s_\ell = \bot$.



(d) Resolving with $s_j = \top$, $s_k = \bot$ and $s_\ell = \bot$.

Fig. 11. The at-least-one-true gadget $\mathrm{ALOne}^i_{j,k,\ell}$ (a) and the three ways to resolve it depending on the boolean assignment to s_j and s_k (d, c, b). These rf constraints are formalized in Lemma 4.3. The edge numbers specify the order in which rf-edges are implied. The crossed-out events are ignored in our analysis for simplicity (see Lemma 4.5).

First, we serially connect all gadgets in their common threads by po. In particular, $\mathrm{Copy}^{i_1}_{j_1}$ appears before $\mathrm{Copy}^{i_2}_{j_2}$ if $i_1 < i_2$ or $i_1 = i_2$ and $j_1 < j_2$; $\overline{\mathrm{Copy}}^{i_1}_{j_1}$ appears before $\overline{\mathrm{Copy}}^{i_2}_{j_2}$ if $i_1 < i_2$ or $i_1 = i_2$ and $j_1 < j_2$; $\mathrm{AMOne}[c]^{i_1}_{j_1,k_1}$ appears before $\mathrm{AMOne}[c]^{i_2}_{j_2,k_2}$ if $i_1 < i_2$, and finally $\mathrm{ALOne}^{i_1}_{j_1,k_1,\ell_1}$ appears before $\mathrm{ALOne}^{i_2}_{j_2,k_2,\ell_2}$ if $i_1 < i_2$. Second, observe that the threads t_i, for $i \in [6]$, appear in multiple gadgets, thus we have to specify how to interleave their corresponding events. We do so by first fixing a total order on memory locations $\sigma = y_1, y_2, y_3, y_4, z_1, z_2, z_3, z_4, a_1, a_2, a_3, b$.

(1) The read events succeeding every $\mathsf{r}(t_3, x_1, v_j^i)$ in each gadget (i.e., those on locations $\{y_\ell, z_\ell\}_{\ell\in[4]}$, $\{a_c\}_{c\in[3]}$ and b) are po-ordered according to σ (note that reads on some locations, such as a_c, b, might not appear at all after $\mathsf{r}(t_3, x_1, v_j^i)$, e.g., if clause C_i does not contain variable s_j). Moreover, all these reads appear before any read on x_1 that is a po-successor of $\mathsf{r}(t_3, x_1, v_j^i)$ (in particular, reads on x_1 with phase $\geq i$ or reads with phase i and step $\geq j$).

(2) Similarly, the write events preceding every $\mathtt{w}(t_1, x_1, v_j^i)$ in each gadget are po-ordered according to σ. Moreover, all these writes appear after any write on x_1 that is a po-predecessor of $\mathtt{w}(t_1, x_1, v_j^i)$. Likewise for $\mathtt{w}(t_2, x_1, v_j^i)$, $\mathtt{w}(t_4, x_2, v_j^i)$ and $\mathtt{w}(t_5, x_2, v_j^i)$.

Finally, observe that we have indeed used $\kappa = 23$ threads and $d = 26$ memory locations. For the latter, we also count one extra memory location for each hb edge out of thread t_6, thus 4 in total.

4.2 Soundness

We start with the soundness of the reduction, in particular, if $\overline{\mathsf{X}} \models \mathsf{WRA}$ (and thus also $\overline{\mathsf{X}} \models \mathcal{M}$) then φ is satisfiable. We achieve this by proving a sequence of intermediate lemmas similar to Section 3, however, each lemma now requires reasoning about the semantics of WRA. Recall that we assign $s_j = \top$ if $(\mathtt{w}(t_2, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$ for all $i \in [m]$, and $s_j = \bot$ if $(\mathtt{w}(t_1, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$ for all $i \in [m]$. The first lemma is based on the copy gadgets Copy_j^i and $\overline{\mathrm{Copy}}_j^i$, and states that each phase of X makes consistent choices for the writer of $\mathtt{r}(t_3, x_1, v_j^i)$ (i.e., whether it reads from t_1 or t_2), which makes the above assignment well-defined. It follows from the high-level description of these two gadgets and the accompanying Fig. 8 and Fig. 9.

Lemma 4.1. *Let $\mathsf{X} = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo})$ be a concrete extension of $\overline{\mathsf{X}}$ that satisfies weak-read-coherence. For all $i_1, i_2 \in [m]$, $j \in [n]$, we have $(\mathtt{w}(t_1, x_1, v_j^{i_1}), \mathtt{r}(t_3, x_1, v_j^{i_1})) \in \mathsf{rf}$ iff $(\mathtt{w}(t_1, x_1, v_j^{i_2}), \mathtt{r}(t_3, x_1, v_j^{i_2})) \in \mathsf{rf}$.*

The next lemma is based on the at-most-one-true gadgets $\mathrm{AMOne}[c]_{j,k}^i$, and states that for every clause C_i and for each of the $c \in [3]$ pair of variables (s_j, s_k) in C_i, at most one of them is assigned to true. Again, it follows by a direct analysis on the accompanying Fig. 10.

Lemma 4.2. *Let $\mathsf{X} = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo})$ be a concrete extension of $\overline{\mathsf{X}}$ that satisfies weak-read-coherence. For every $i \in [m]$ and $j, k \in [n]$ such that s_j and s_k appear in clause C_i, we have $(\mathtt{w}(t_2, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \notin \mathsf{rf}$ or $(\mathtt{w}(t_2, x_1, v_k^i), \mathtt{r}(t_3, x_1, v_k^i)) \notin \mathsf{rf}$.*

The third lemma is based on the at-least-one-true gadget $\mathrm{ALOne}_{j,k,\ell}^i$, and it is used to show that for every clause $C_i = (s_j, s_k, s_\ell)$, at least one of its variables is assigned to true. Again, it follows by a direct analysis on the accompanying Fig. 11.

Lemma 4.3. *Let $\mathsf{X} = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo})$ be a concrete extension of $\overline{\mathsf{X}}$ that satisfies weak-read-coherence. For every $i \in [m]$ and $j, k, \ell \in [n]$ such that s_j, s_k and s_ℓ appear in clause C_i, we have $(\mathtt{w}(t_2, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$ or $(\mathtt{w}(t_2, x_1, v_k^i), \mathtt{r}(t_3, x_1, v_k^i)) \in \mathsf{rf}$ or $(\mathtt{w}(t_2, x_1, v_\ell^i), \mathtt{r}(t_3, x_1, v_\ell^i)) \in \mathsf{rf}$.*

Lemma 4.1 states that our truth assignment for φ is valid, while Lemma 4.2 and Lemma 4.3 guarantee that in every clause, exactly one literal is set to true. Hence we have the following corollary.

Corollary 4.4. *If $\overline{\mathsf{X}} \models \mathsf{WRA}$ then φ is satisfiable.*

4.3 Completeness

We now turn our attention to completeness property, i.e., if φ is satisfiable then $\overline{\mathsf{X}} \models \mathsf{SRA}$ (and thus also $\overline{\mathsf{X}} \models \mathcal{M}$). We use the notions of phase, step, and monotonicity as in Section 3. We construct a reads-from relation rf and a partial modification order $\overline{\mathsf{mo}}$ as follows.

(1) For each gadget, we insert rf and mo edges according to Figs. 8 to 11 and the truth assignments on variables s_j, s_k, s_ℓ involved in that gadget. In particular, this implies that, for each $i \in [m]$

and $j \in [n]$, we have $(\mathtt{w}(t_1, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$ if $s_j = \bot$ and $(\mathtt{w}(t_2, x_1, v_j^i), \mathtt{r}(t_3, x_1, v_j^i)) \in \mathsf{rf}$ if $s_j = \top$. Observe that rf is fully specified, i.e., every read has a write to read from. Note that for every pair $(\mathtt{w}, \mathtt{r}) \in \mathsf{rf}$ we have $\mathsf{phase}(\mathtt{w}) = \mathsf{phase}(\mathtt{r})$ and $\mathsf{step}(\mathtt{w}) = \mathsf{step}(\mathtt{r})$, thus $\mathtt{r} \preceq \mathtt{w}$.

(2) For every two conflicting writes $\mathtt{w}_1, \mathtt{w}_2$ such that (i) $\mathtt{w}_1 \prec \mathtt{w}_2$, and (ii) $(\mathtt{w}_1, \mathtt{w}_2) \notin \mathsf{hb}$, we have $(\mathtt{w}_1, \mathtt{w}_2) \in \overline{\mathsf{mo}}$. Thus, for any two conflicting writes $\mathtt{w}_1, \mathtt{w}_2$ with $\mathtt{w}_1 \prec \mathtt{w}_2$ we have $(\mathtt{w}_1, \mathtt{w}_2) \in (\mathsf{hb} \cup \overline{\mathsf{mo}})^+$.

We call a triplet $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ *safe* if either $(\mathtt{w}', \mathtt{r}) \notin \mathsf{hb}$ or $(\mathtt{w}', \mathtt{w}) \in (\mathsf{hb} \cup \overline{\mathsf{mo}})^+$. To prove the consistency of $\overline{\mathsf{X}}$, it suffices to argue that $\overline{\mathsf{mo}}$ is *minimally coherent*, namely, that (i) $(\mathsf{hb} \cup \overline{\mathsf{mo}})^+$ is acyclic, and (ii) every triplet $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ is safe. Indeed, these two conditions guarantee that $\overline{\mathsf{mo}}$ can be linearized to a total mo such that any extension X of $\overline{\mathsf{X}}$ satisfies $\mathsf{X} = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo}) \models \mathsf{SRA}$ [Tunç et al. 2023], which implies that also $\mathsf{X} \models \mathcal{M}$.

In order to simplify our analysis, we again ignore the writes in threads $\{h_c\}_{c \in [3]}$, p, and q that are not read in rf (crossed-out in Fig. 10 and Fig. 11). This allows us to make our formal statements more uniform, while it does not affect the correctness of the analysis. Indeed, let $\mathsf{InactiveWr} = \{\mathtt{w} \in \mathsf{W} : \mathsf{tid}(\mathtt{w}) \in \{h_1, h_2, h_3, p, q\} \text{ and } \nexists \mathtt{r} \in \mathsf{R}.\ (\mathtt{w}, \mathtt{r}) \in \mathsf{rf}\}$. The following lemma is straightforward.

Lemma 4.5. *The following statements hold.*

(1) If there is an $(\mathsf{hb} \cup \overline{\mathsf{mo}})$*-cycle then there is such a cycle without any write in* InactiveWr.
(2) If there is an unsafe triplet, then there is such a triplet $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ *where* $\mathtt{w}' \notin \mathsf{InactiveWr}$.

We start with condition (i) of minimal coherence, i.e., we need to show that $(\mathsf{hb} \cup \overline{\mathsf{mo}})$ is acyclic. Observe that each individual gadget is free from $(\mathsf{hb} \cup \overline{\mathsf{mo}})$-cycles, regardless of how we resolve the rf edges associated with it (see Figs. 8b and 8c, Figs. 9a and 9b, Figs. 10b to 10d, Figs. 11b to 11d). However, we have to also argue that the interleaving of these gadgets is free from $(\mathsf{hb} \cup \overline{\mathsf{mo}})$-cycles.

Our first key lemma states that hb paths between writes are, without loss of generality, monotonic. This is based on three observations. First, due to Lemma 4.5, we can ignore writes in the threads h_1, h_2, h_3, p and q, which contain po-edges that would violate this statement. Second, all rf-edges connect events of the same phase and step. Third, the only po-edges that are non-monotonic enter read events (in particular, a read $\mathtt{r}(v_j^i, z_6, g_1)$ or a read $\mathtt{r}(v_j^i, z_8, g_4)$ in $\overline{\mathsf{Copy}}_j^i$). Since the only possible continuation of an hb-path out of a read event is to take another po-edge, we can remove the first non-monotonic edge (as po is transitive) and obtain a new valid hb-path. Repeating this process results in a monotonic hb-path between the writes. Formally, we have the following.

Lemma 4.6. *For every two writes* $\mathtt{w}_1$, $\mathtt{w}_2$, *if* $(\mathtt{w}_1, \mathtt{w}_2) \in \mathsf{hb}$ *then there exists a monotonic* hb*-path* $\mathtt{w}_1 \overset{\mathsf{hb}}{\rightsquigarrow} \mathtt{w}_2$.

We can now prove the acyclicity condition of minimal coherence. Intuitively, any potential $(\mathsf{hb} \cup \overline{\mathsf{mo}})$-cycle C can be seen as a sequence of write events connected by hb and $\overline{\mathsf{mo}}$. By construction, every edge $\mathtt{w}_1 \xrightarrow{\overline{\mathsf{mo}}} \mathtt{w}_2$ is monotonic, while, due to Lemma 4.6, every subpath $\mathtt{w}_1 \overset{\mathsf{hb}}{\rightsquigarrow} \mathtt{w}_2$ of C is, without loss of generality, monotonic. Thus C is monotonic, and since it is a cycle, every event in C has the same phase and step. The absence of such cycles can then be directly established by inspecting the gadgets in Figs. 8 to 11.

Lemma 4.7. $(\mathsf{hb} \cup \overline{\mathsf{mo}})$ *is acyclic.*

Next, we turn our attention to the second condition of minimal coherence, i.e., we argue that every triplet is safe. We first prove a general statement that prohibits hb-paths to a read $\mathtt{r}$ from writes $\mathtt{w}'$ that are po-successors to the write $\mathtt{w}$ that $\mathtt{r}$ reads from. This will help us establish the safety of each triplet, and will also prove useful later in Section 5.1 when we address Causal Memory.

Lemma 4.8. *For every pair* $(\mathtt{w}, \mathtt{r}) \in \mathsf{rf}$ *and write* $\mathtt{w}'$ *with* $(\mathtt{w}, \mathtt{w}') \in \mathsf{po}$*, we have* $(\mathtt{w}', \mathtt{r}) \notin \mathsf{hb}$.

To realize Lemma 4.8, we first argue that any path $P\colon \mathtt{w}' \overset{\mathsf{hb}}{\rightsquigarrow} \mathtt{r}$ contains events of the same phase and step. Indeed, as no location is ever written and read by the same thread, P has the general form $P\colon \mathtt{w}' \overset{\mathsf{hb}^?}{\rightsquigarrow} \mathtt{w}'' \overset{\mathsf{rf}}{\longrightarrow} \mathtt{r}'' \overset{\mathsf{po}^?}{\longrightarrow} \mathtt{r}$ for some write $\mathtt{w}''$ and read $\mathtt{r}''$. Due to Lemma 4.6, the subpath $\mathtt{w}' \overset{\mathsf{hb}^?}{\rightsquigarrow} \mathtt{w}''$ is monotonic (wlog), while the last two edges of P are also monotonic (rf-edges are monotonic, while non-monotonic po-edges go from writes to reads). Hence P is monotonic. On the other hand, we have $\mathtt{w} \preceq \mathtt{w}'$ (as $(\mathtt{w}, \mathtt{w}') \in \mathsf{po}$), while, by construction, $\mathtt{r} \preceq \mathtt{w}$. Hence $\mathtt{r} \preceq \mathtt{w}'$, and since P is monotonic, it must contain only events of the same phase and step. In particular, P must be contained in the gadgets in Figs. 8 to 11. The absence of such paths P can then be established by a careful inspection of these gadgets.

We can now prove the safety of each triplet $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$. Intuitively, if $\mathtt{w}' \prec \mathtt{w}$, then we have $(\mathtt{w}', \mathtt{w}) \in \overline{\mathsf{mo}}$ by construction. On the other hand, if $\mathtt{w} \prec \mathtt{w}'$, Lemma 4.6 and Lemma 4.8 exclude the existence of hb-paths $\mathtt{w}' \overset{\mathsf{hb}}{\rightsquigarrow} \mathtt{r}$. Hence, it again suffices to only consider hb-paths $P\colon \mathtt{w}' \overset{\mathsf{hb}}{\rightsquigarrow} \mathtt{r}$ that are contained in the same gadget. Again, a careful inspection of Figs. 8 to 11 and the use of Lemma 4.8 show that $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ is indeed safe.

Lemma 4.9. *Every triplet* $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ *is safe.*

Lemma 4.7 and Lemma 4.9 show that $\overline{\mathsf{mo}}$ is indeed minimally coherent, which implies that $\mathsf{X} = (\mathsf{E}, \mathsf{po}, \mathsf{rf}, \mathsf{mo}) \models \mathrm{SRA}$. Thus we have the following corollary.

Corollary 4.10. *If* φ *is satisfiable then* $\overline{\mathsf{X}} \models \mathrm{SRA}$.

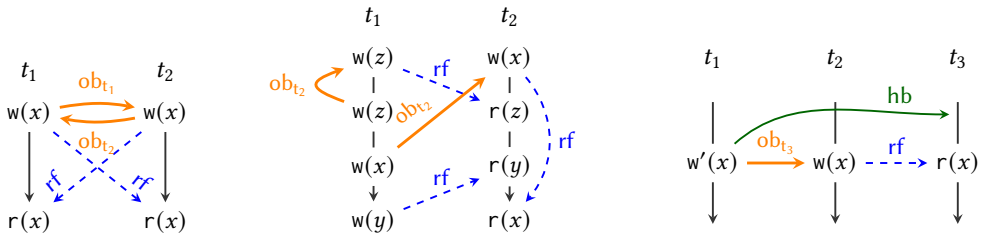
Together, Corollary 4.4 and Corollary 4.10 establish the correctness of the reduction, i.e., φ is satisfiable iff $\overline{\mathsf{X}} \models \mathcal{M}$ for any memory model $\mathrm{SRA} \preccurlyeq \mathcal{M} \preccurlyeq \mathrm{WRA}$.

5 IMPLICATIONS AND OTHER MEMORY MODELS

Our proof of Theorem 1.1 is strong enough to yield hardness on other popular memory models across different domains. In this section, we explore its implications.

Causal Consistency models. In a distributed setting, consistency commonly captures the concept of *causality*. Three of the most standard causal models are the basic Causal Consistency (CC), Causal Convergence (CCv), and Causal Memory (CM) [Bouajjani et al. 2017]. It was recently shown that CC coincides with WRA while CCv coincides with SRA [Lahav and Boker 2022]. Thus, Theorem 1.1 implies NP-completeness for all models between CCv and CC. In Section 5.1 we also establish NP-completeness for CM, by extending the proof of Theorem 1.1, thereby completing Theorem 1.2.

Hardware memory models. Next, we turn our attention to some popular hardware memory models, namely, for the POWER and x86-TSO architectures, as well as PSO. We show that Theorem 1.1 implies NP-completeness for POWER, but consistency checks for TSO/PSO run in polynomial time.



(a) CM-consistent (b) CM-inconsistent (c) The argument of Lemma 5.3

Fig. 12. (a) An execution consistent in CM, as each of ob_{t_1} and ob_{t_2} is acyclic, but inconsistent in SRA. (b) An execution inconsistent in CM, as ob_{t_2} creates a cycle, but consistent in SRA. (c) Illustration of the argument behind Lemma 5.3. Since $\mathrm{w}'(x)$ reaches an earlier event in t_3 than $\mathrm{w}(x)$, the edge $\mathrm{w}'(x) \xrightarrow{\mathrm{ob}_{t_3}} \mathrm{w}(x)$ does not create a new path from $\mathrm{w}'(x)$ to t_3.

The fully Relaxed model. Finally, we consider the Relaxed model which does not require $(\mathrm{po} \cup \mathrm{rf})$-acyclicity, and remark that this brings the problem in polynomial time. Interestingly, Relaxed is the only non-multi-copy atomic model in our list for which consistency testing is tractable.

5.1 Implications for Causal Memory

Here we prove the case (ii) of Theorem 1.2, i.e., the hardness of bounded consistency testing for any memory model $\mathcal{M}$ with $\mathrm{CC} \preccurlyeq \mathcal{M} \preccurlyeq \mathrm{CM}$. Instead of performing a separate reduction, we reuse our reduction from Section 4. Let $\varphi = \{C_i\}_{i \in [m]}$ be a Boolean formula over n variables $\{s_j\}_{j \in [n]}$ and m clauses of the form $C_i = (s_j, s_k, s_\ell)$, for which we have to solve Monotone 3SAT. Moreover, let $\overline{\mathrm{X}} = (\mathrm{E}, \mathrm{po})$ be the abstract execution as constructed in Section 4. Since WRA coincides with CC [Lahav and Boker 2022], we have the following soundness corollary.

Corollary 5.1. *If* $\overline{\mathrm{X}} \models \mathrm{CC}$ *then* φ *is satisfiable.*

Thus, to complete the theorem, it remains to show that if $\overline{\mathrm{X}} \models \mathrm{CM}$ then φ is satisfiable. Towards this, we next formalize CM and then prove the statement.

Causal Memory. Causal memory is similar to Causal Consistency but further requires that each thread has a locally-consistent view of the order in which different writes have been executed [Ahamad et al. 1995; Bouajjani et al. 2017]. This is made formal by introducing one additional relation for each thread, called the *observed-before* relation ob.

The observed-before relation ob. Given an event e, the *observed-before relation*[‡] for e is the smallest transitive relation $\mathrm{ob}_e \subseteq \mathrm{E} \times \mathrm{E}$ with the following properties.

(1) For every $(e_1, e_2) \in \mathrm{hb}$ such that $(e_i, e) \in \mathrm{hb}$ for each $i \in [2]$, we have $(e_1, e_2) \in \mathrm{ob}_e$.
(2) For every conflicting triplet $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ such that (i) $(\mathtt{w}', \mathtt{r}) \in \mathrm{ob}_e$ and (ii) $(\mathtt{r}, e) \in \mathrm{po}^?$, we have $(\mathtt{w}', \mathtt{w}) \in \mathrm{ob}_e$.

Intuitively, when t executed $\mathtt{r}$, it must have observed $\mathtt{w}$ after $\mathtt{w}'$, so that $\mathtt{r}$ indeed obtained its value from $\mathtt{w}$. The ob_e relation specifies that this ordering cannot change later when t executes e. Notice the fixpoint style of the above definition. As we order $(\mathtt{w}_1, \mathtt{w}_2) \in \mathrm{ob}_e$ (and since the relation is transitive and contains hb), more and more write-read pairs satisfy property (i) $(\mathtt{w}', \mathtt{r}) \in \mathrm{ob}_e$, triggering the addition of new orderings $(\mathtt{w}', \mathtt{w}) \in \mathrm{ob}_e$. For any two events e_1, e_2 with $(e_1, e_2) \in \mathrm{po}$,

[‡]Other works refer to this relation as "happened-before" for e. We avoid this term here to not confuse it with hb.

we have $\mathrm{ob}_{e_1} \subseteq \mathrm{ob}_{e_2}$, i.e., ob grows monotonically as we go downwards in each thread. The observed-before relation for a thread t is defined as $\mathrm{ob}_t = \mathrm{ob}_{e^{\max}}$, where $e^{\max}$ is the po-maximal event of t. The new axiom requires that ob_t is irreflexive [Bouajjani et al. 2017].

- ob_t is irreflexive for each thread t (ob-acyclicity)

In turn, CM is equal to WRA with ob-acyclicity as an extra axiom.

- (porf-acyclicity) $\wedge$ (weak-read-coherence) $\wedge$ (ob-acyclicity) [CM]

Observe that WRA $\preccurlyeq$ CM, but CM is incomparable with RA/SRA, i.e., CM allows executions that are inconsistent in RA/SRA and vice versa. See Fig. 12a and Fig. 12b for illustrations.

Next, we prove the completeness of the construction, i.e., if φ is satisfiable then $\overline{\mathsf{X}} \models \mathsf{CM}$. Consider the reads-from relation rf and the partial modification order $\overline{\mathrm{mo}}$ exactly as constructed in the completeness argument of Section 4 (i.e., following the gadgets in Figs. 8 to 11). Let $\mathsf{X} = (\mathsf{E}, \mathrm{po}, \mathrm{rf}, \mathrm{mo})$ be the execution witnessing the SRA-consistency of $\overline{\mathsf{X}}$ according to Corollary 4.10. Since SRA satisfies the porf-acyclicity and weak-read-coherence axioms, we only need to argue that X also satisfies ob-acyclicity to conclude that $\mathsf{X} \models \mathsf{CM}$. For this, we have to establish some additional lemmas.

Our first lemma stems from Lemma 4.8 and states an important property of rf: for every pair $(\mathtt{w}, \mathtt{r}) \in \mathrm{rf}$, $\mathtt{w}$ has no hb-path to po-predecessors of $\mathtt{r}$. In other words, the first event of the thread of $\mathtt{r}$ that $\mathtt{w}$ can reach by means of an hb-path is $\mathtt{r}$ itself via the rf-edge $\mathtt{w} \xrightarrow{\mathrm{rf}} \mathtt{r}$.

Lemma 5.2. *For every* $(\mathtt{w}, \mathtt{r}) \in \mathrm{rf}$ *and event* e *such that* $(e, \mathtt{r}) \in \mathrm{po}$*, we have* $(\mathtt{w}, e) \notin \mathrm{hb}$.

Next, we define a "one-hop" variant of ob. Given an event e, the *one-hop observed-before relation* for e is the smallest transitive relation $\mathrm{ob}^1_e \subseteq \mathsf{E} \times \mathsf{E}$ with the following properties.

(1) For every $(e_1, e_2) \in \mathrm{hb}$ such that $(e_i, e) \in \mathrm{hb}$ for each $i \in [2]$, we have $(e_1, e_2) \in \mathrm{ob}^1_e$.
(2) For every event e and conflicting triplet $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ such that (i) $(\mathtt{w}', \mathtt{r}) \in \mathrm{hb}$ and (ii) $(\mathtt{r}, e) \in \mathrm{po}^?$, we have $(\mathtt{w}', \mathtt{w}) \in \mathrm{ob}^1_e$.

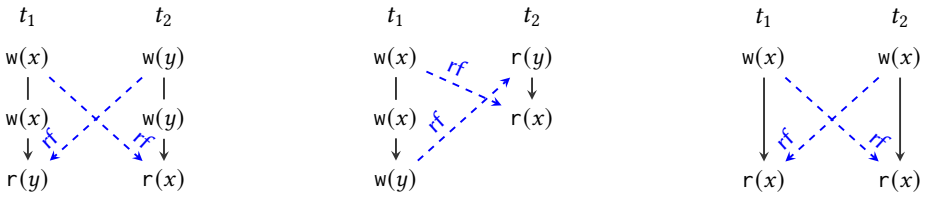
Contrasting ob^1_e to ob_e, the only difference is in condition (2i): ob_e checks whether $(\mathtt{w}', \mathtt{r}) \in \mathrm{ob}_e$, while ob^1_e checks the weaker condition $(\mathtt{w}', \mathtt{r}) \in \mathrm{hb}$. Thus ob^1_e does not have the fixpoint style of ob_e. Similarly to ob_t, we let $\mathrm{ob}^1_t = \mathrm{ob}^1_{e^{\max}}$, where $e^{\max}$ is the po-maximal event of thread t.

Our next lemma states that for our execution X, ob_e coincides with ob^1_e. In other words, ob_e reaches a fixpoint after only one iteration. This observation stems from Lemma 5.2. Intuitively, since for each triplet $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$, $\mathtt{w}$ cannot hb-reach any po-predecessor of $\mathtt{r}$, traversing an edge $\mathtt{w}' \xrightarrow{\mathrm{ob}_e} \mathtt{w}$ cannot lead to any events of the thread of $\mathtt{r}$ that weren't already reachable via the hb-path $\mathtt{w}' \overset{\mathrm{hb}}{\rightsquigarrow} \mathtt{r}$ that made us insert $(\mathtt{w}', \mathtt{w}) \in \mathrm{ob}_e$ in the first place (see Fig. 12c). Hence, adding such an ordering $(\mathtt{w}', \mathtt{w}) \in \mathrm{ob}_e$ cannot lead to further firings of condition (2i) of ob_e. Formally, we have the following.

Lemma 5.3. *For every thread* t*, we have* $\mathrm{ob}^1_t = \mathrm{ob}_t$.

Finally, observe that whenever we add $(\mathtt{w}', \mathtt{w}) \in \mathrm{ob}^1_e$, we have $(\mathtt{w}', \mathtt{r}) \in \mathrm{hb}$. Due to Lemma 4.9, the triplet $(\mathtt{w}, \mathtt{r}, \mathtt{w}')$ is safe, thus $(\mathtt{w}', \mathtt{w}) \in (\mathrm{hb} \cup \overline{\mathrm{mo}})^+$. Hence, the acyclicity of $\mathrm{ob}_t = \mathrm{ob}^1_t$ follows from the acyclicity of $(\mathrm{hb} \cup \overline{\mathrm{mo}})^+$ (Lemma 4.7). We thus have the following lemma, which, together with Corollary 5.1, completes the proof case (ii) of Theorem 1.2.

Lemma 5.4. *If* φ *is satisfiable, then* $\overline{\mathsf{X}} \models \mathsf{CM}$.



(a) TSO-consistent (b) PSO-consistent (c) TSO/PSO-inconsistent

Fig. 13. (a) An execution consistent in TSO, as well as in PSO and all other memory models we have considered, but not SC. (b) An execution consistent in PSO, as well as in Relaxed-Acyclic but not in TSO or even WRA. (c) An execution inconsistent in TSO/PSO, but consistent in CC/WRA.

5.2 Implications for POWER

The memory model of the POWER architecture is defined on load, store, atomic read-modify-write memory accesses, and various types of fences. POWER orders memory accesses based on fences, address, data, and control dependencies, while, again, coherence forces a total order on same-location accesses. In addition, POWER defines two global orderings, namely, *happens-before*, and *propagation*. The happens-before relation is based on dependencies, fences, and the rf relation across threads. The propagation relation captures the propagation of read and written values by combining fences, happens-before, rf, and mo. Based on these relations POWER defines its consistency axioms, which we will not present here; instead, we refer the interested readers to [Alglave et al. 2014].

Lahav et al. [2016] showed that SRA captures precisely the guarantees of POWER for programs that are compiled from the release-acquire fragment of C/C++. In turn, this implies that the result established in Section 4 for SRA transfers over to POWER. We thus have the following corollary.

Corollary 1.3. *Consistency testing for bounded inputs is NP-complete for* POWER.

5.3 What About x86-TSO and PSO?

Our results so far prove strong hardness for testing a variety of weak-memory models. In contrast, in this section, we outline that the problem is solvable in polynomial time for x86-TSO and PSO. Conceptually, this is less surprising for TSO, which diverges only a little from SC, but is more so for PSO, which allows for behaviors that are not even causally consistent.

Total Store Order. The TSO model deviates from SC by introducing a write-buffer for each thread, which acts as a FIFO queue [Sewell et al. 2010]. When a thread t executes a write $\mathtt{w}(t, x)$, this does not modify the shared memory immediately and is thus not visible to the other threads. Instead, $\mathtt{w}(t, x)$ is stored in the buffer of t. The buffer non-deterministically flushes some of its writes to the shared memory, at which point they become visible to the other threads. On the other hand, when a thread t executes a read $\mathtt{r}(t, x)$, it is forced to read from the most recent write $\mathtt{w}(t, x)$ in t's buffer. If no such write exists then $\mathtt{r}(t, x)$ reads from the shared memory. See Fig. 13a for an illustration.

For capturing the complexity of consistency-testing of an abstract execution $\overline{\mathsf{X}} = (\mathsf{E}, \mathrm{po})$ under TSO, it is helpful to switch to operational semantics. The semantics are defined by means of a labeled transition system $\mathcal{L}_{\mathrm{TSO}}$ In high level, a state in $\mathcal{L}_{\mathrm{TSO}}$ is a triplet $\langle \mathrm{P}, \mathrm{B}, \mathrm{M} \rangle$, where

(1) $\mathrm{P} \subseteq \mathsf{E}$ is the set of events that have been executed so far.
(2) $\mathrm{B} \colon \mathcal{T} \to (\mathsf{W})^*$ maps every thread t to a sequence of writes $\mathtt{w}(t, x_1), \mathtt{w}(t, x_2), \ldots, \mathtt{w}(t, x_i)$, which represents the state of the buffer of thread t.
(3) $\mathrm{M} \colon \mathcal{V} \to \mathsf{W}$ maps every memory location of the shared memory to the most recent write to it.

A counting argument shows that the size of $\mathcal{L}_{\text{TSO}}$ is bounded by $\kappa^d \cdot n^{O(\kappa^2)}$, for $n = |\mathsf{E}|$, κ threads, d locations, and thus becomes polynomial when $\kappa, d = O(1)$.

Partial Store Order (PSO). The PSO model [SPARC International 1994] is similar to TSO, with the difference that every thread has a different buffer for each location. This allows both write-read reorderings (like TSO) and write-write reorderings on different locations. This induces more behaviors than TSO, but is incomparable to some other models like WRA (see Fig. 13b and Fig. 13c). The operational semantics can be defined by means of an LTS $\mathcal{L}_{\text{PSO}}$ analogously to $\mathcal{L}_{\text{TSO}}$. A similar analysis shows that the size of $\mathcal{L}_{\text{PSO}}$ is bounded by $n^{O(\kappa(\kappa+d))}$. Hence we have the following theorem, which differentiates TSO/PSO from the other weak-memory models we have seen so far.

Theorem 1.4. *Consistency testing for bounded threads and memory locations is in polynomial time for* TSO *and* PSO.

5.4 A Final Note on Relaxed

Finally, we turn our attention to the Relaxed model. The only two axioms of this model are relaxed-write-coherence and relaxed-read-coherence, which concern individual memory locations, and guarantee per-location coherence, i.e., focusing on each location individually, the corresponding execution is SC-consistent. Given an abstract execution $\overline{\mathsf{X}}$, to decide whether $\overline{\mathsf{X}} \models$ Relaxed, it suffices to check whether $\overline{\mathsf{X}}_x \models$ SC for each location x, where $\overline{\mathsf{X}}_x$ occurs from $\overline{\mathsf{X}}$ by considering only events accessing x. As each consistency check $\overline{\mathsf{X}}_x \models$ SC takes polynomial time [Agarwal et al. 2021], and we clearly have polynomially many such checks, we arrive at Corollary 1.5.

Corollary 1.5. *Consistency testing for bounded threads is in polynomial time for* Relaxed.

6 HARDNESS WITH BOUNDED VALUES

For ease of presentation, our reductions in Section 3 and Section 4 use a bounded number of threads and memory locations but an unbounded value domain. Indeed, given the Boolean formula $\varphi = \{C_i\}_{i \in [m]}$ on m clauses and n variables, the value domain of the abstract execution $\overline{\mathsf{X}}$ has size $\Theta(n \cdot m)$. In this section we outline how to modify those reductions so that $\overline{\mathsf{X}}$ also uses a bounded value domain, thereby arriving at Theorem 1.1 and Theorem 1.2.

Intuition. Our two reductions are such that every read $\mathtt{r}$ can read from at most three writes, and these appear in the same gadget as $\mathtt{r}$. However, the values of these events are specific to the gadget, and in particular, specific to the phase i and the step j of the events (i.e., events are of the form $\mathtt{r}(x_1, t_3, v_j^i)$). Our strategy for decreasing the size of the value domain (of both reductions in Section 4 and Section 3) is by using repeating values which are not parameterized by the superscript i and subscript j (i.e., the events in the executions constructed now look like $\mathtt{r}(x_1, t_3, v)$ or $\mathtt{w}(x_1, t_1, v)$). This change does not affect completeness but threatens soundness, as now, some read events may read from write events in other gadgets that were previously forbidden simply because their values were not matching. To avoid this, we slightly modify our abstract executions $\overline{\mathsf{X}}$ by inserting a bounded number of auxiliary write and read events between consecutive gadgets, which also access a bounded number of values. The auxiliary write events write dummy values read by the auxiliary read events. The effect of these additional reads-from edges due to auxiliary events is to create $(\mathrm{po}_x \cup \mathrm{rf}_x)^+$ paths that once again forbid the original (i.e., non-auxiliary) read events of a gadget to access write events from other gadgets (while obeying the desired consistency axioms).

Construction. We now outline the construction. The process is similar for both the abstract executions of Section 3 and Section 4. For this reason, we describe it generically on an abstract

execution $\overline{\mathsf{X}}$. Our transformation is carried out in two steps, $\overline{\mathsf{X}} \rightsquigarrow \overline{\mathsf{X}}_1 \rightsquigarrow \overline{\mathsf{X}}_2$, where $\overline{\mathsf{X}}_1$ and $\overline{\mathsf{X}}_2$ have the same number of threads and locations as $\overline{\mathsf{X}}$, and $\overline{\mathsf{X}}_2$ additionally has a bounded value domain.

Step 1. We obtain $\overline{\mathsf{X}}_1$ by inserting various events in $\overline{\mathsf{X}}$ while keeping the threads and memory locations the same. We start by fixing a total order σ_1 on locations, and a total order σ_2 on threads.

$$\sigma_1 = x_1, x_2, y_1, \ldots, y_8, z_1, \ldots, z_8, a_1, \ldots, a_3, b$$
$$\sigma_2 = t_1, \ldots, t_6, f_1, \ldots, f_6, g_1, \ldots, g_6, h_1, \ldots, h_3, p, q$$

Fix a phase i and step j and let $\zeta = (i * m + j) \mod 2$. For a location x of $\overline{\mathsf{X}}$, different from a_1, a_2, a_3, b, we introduce auxiliary write and read events on x as follows: (i) if a thread t writes on x,we insert a write $\mathtt{w}(t, x, v^t_\zeta)$ after all events of phase i and step j in t, and (ii) if a thread t reads from x, we insert a sequence of read events $\mathtt{r}(t, x, v^{t^1}_\zeta), \mathtt{r}(t, x, v^{t^2}_\zeta), \ldots$ before all events of phase i and step $j+1$ (or phase $i+1$ and step 1, if $j = n$), where $t^1, t^2, \ldots$ is the subsequence of σ_2 of threads writing to x values read by thread t. We repeat this process for all locations $x \notin \{a_1, a_2, a_3, b\}$ in the order of appearance in the total order σ_1, placing the auxiliary writes before the auxiliary reads in each thread. Observe that each $\mathtt{r}(t, x, v^{t^\ell}_\zeta)$ event is forced to read from the respective $\mathtt{w}(t^\ell, x, v^{t^\ell}_\zeta)$.

Next, we turn our attention to the locations a_1, a_2, a_3 and b. The auxiliary events are positioned similarly, except for the detail about the step number j, because accesses to these locations span an entire phase (in the at-most-one-true and at-least-one-true gadgets). In particular, we have $\zeta = i \mod 2$, while auxiliary write events are placed in each thread after all events of phase i, and read events are placed before events of phase $i+1$.

Observe that since the number of threads and locations is bounded in $\overline{\mathsf{X}}$, the same holds for $\overline{\mathsf{X}}_1$, while the additional values accessed by the auxiliary events in $\overline{\mathsf{X}}_1$ are also bounded.

Step 2. In the second step, we transform $\overline{\mathsf{X}}_1$ to $\overline{\mathsf{X}}_2$ so that the latter only accesses a bounded number of values. In particular, we make $\overline{\mathsf{X}}_2$ identical to $\overline{\mathsf{X}}_1$ with the difference that, for every event of $\overline{\mathsf{X}}_1$ that also appears in $\overline{\mathsf{X}}$ (i.e., non-auxiliary events), we remove from its value the superscript of the phase and the subscript of the step of that event. For example, each write $\mathtt{w}(t_1, x_1, v^i_j)$ in $\overline{\mathsf{X}}_1$ becomes $\mathtt{w}(t_1, x_1, v)$ in $\overline{\mathsf{X}}_2$. It is straightforward to verify that $\overline{\mathsf{X}}_2$ has a bounded domain of threads, locations, and values. Moreover, $\overline{\mathsf{X}}_2$ is consistent in the respective memory model iff $\overline{\mathsf{X}}$ is, by repeating the arguments in Section 3 and Section 4, this time also accounting for the auxiliary events.

7 CONCLUSION

We have studied the standard problem of consistency-testing for various popular weak-memory models spanning across software, hardware, and distributed systems. We have shown that even the *bounded* version of consistency testing is NP-complete in most of these models, i.e., when every natural input parameter is bounded. This is a significant improvement over an abundance of prior hardness results which primarily stemmed from parameters such as the number of threads or memory locations being unbounded. Our results thus highlight the true intricacies of weak-memory testing. In particular, our results imply that the problem provably admits no parameterization with respect to natural input parameters. Interesting future work includes the possibility of extending our hardness to other memory models such as the one in ARM architectures, as well as recovering tractability by imposing further restrictions (such as context/view-switching).

ACKNOWLEDGMENTS

Andreas Pavlogiannis was partially supported by a research grant (VIL42117) from VILLUM FONDEN. S. Krishna was partially supported by the SERB MATRICS grant MTR/2019/000095. Umang Mathur was partially supported by a Singapore Ministry of Education (MoE) Academic Research Fund (AcRF) Tier 1 grant.

REFERENCES

Parosh Abdulla, Mohamed Faouzi Atig, S. Krishna, Ashutosh Gupta, and Omkar Tuppe. 2023. Optimal Stateless Model Checking for Causal Consistency. In *Tools and Algorithms for the Construction and Analysis of Systems*, Sriram Sankaranarayanan and Natasha Sharygina (Eds.). Springer Nature Switzerland, Cham, 105–125.

Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bengt Jonsson, Magnus Lång, Tuan Phong Ngo, and Konstantinos Sagonas. 2019. Optimal stateless model checking for reads-from equivalence under sequential consistency. *Proc. ACM Program. Lang.* 3, OOPSLA (2019), 150:1–150:29. https://doi.org/10.1145/3360576

Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bengt Jonsson, and Tuan Phong Ngo. 2018. Optimal Stateless Model Checking under the Release-Acquire Semantics. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 135 (oct 2018), 29 pages. https://doi.org/10.1145/3276505

Sarita V. Adve and Mark D. Hill. 1990. Weak Ordering&Mdash;a New Definition. In *ISCA '90*. 2–14. https://doi.org/10.1145/325164.325100

Pratyush Agarwal, Krishnendu Chatterjee, Shreya Pathak, Andreas Pavlogiannis, and Viktor Toman. 2021. Stateless Model Checking Under a Reads-Value-From Equivalence. In *CAV'21*. Springer International Publishing, 341–366.

Mustaque Ahamad, Gil Neiger, James E. Burns, Prince Kohli, and Phillip W. Hutto. 1995. Causal Memory: Definitions, Implementation, and Programming. *Distrib. Comput.* 9, 1 (mar 1995), 37–49. https://doi.org/10.1007/BF01784241

Jade Alglave, Will Deacon, Richard Grisenthwaite, Antoine Hacquard, and Luc Maranget. 2021. Armed Cats: Formal Concurrency Modelling at Arm. *ACM Trans. Program. Lang. Syst.* 43, 2, Article 8 (2021), 54 pages. https://doi.org/10.1145/3458926

Jade Alglave, Luc Maranget, Susmit Sarkar, and Peter Sewell. 2011. Litmus: Running Tests against Hardware. In *Tools and Algorithms for the Construction and Analysis of Systems*, Parosh Aziz Abdulla and K. Rustan M. Leino (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 41–44.

Jade Alglave, Luc Maranget, and Michael Tautschnig. 2014. Herding cats: modelling, simulation, testing, and data-mining for weak memory. *ACM Trans. Program. Lang. Syst.* 36, 2 (2014), 7:1–7:74. https://doi.org/10.1145/2627752

Mark Batty, Mike Dodds, and Alexey Gotsman. 2013. Library abstraction for C/C++ concurrency. In *POPL'13*. ACM, 235–248. https://doi.org/10.1145/2429069.2429099

Ahmed Bouajjani, Constantin Enea, Rachid Guerraoui, and Jad Hamza. 2017. On Verifying Causal Consistency. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages* (Paris, France) *(POPL '17)*. Association for Computing Machinery, New York, NY, USA, 626–638. https://doi.org/10.1145/3009837.3009888

Truc Lam Bui, Krishnendu Chatterjee, Tushar Gautam, Andreas Pavlogiannis, and Viktor Toman. 2021. The Reads-from Equivalence for the TSO and PSO Memory Models. *Proc. ACM Program. Lang.* OOPSLA (2021). https://doi.org/10.1145/3485541

Sebastian Burckhardt. 2014. Principles of Eventual Consistency. *Found. Trends Program. Lang.* 1, 1–2 (oct 2014), 1–150. https://doi.org/10.1561/2500000011

Jason F. Cantin, Mikko H. Lipasti, and James E. Smith. 2005. The Complexity of Verifying Memory Coherence and Consistency. *IEEE Trans. Parallel Distrib. Syst.* 16, 7 (jul 2005), 663–671. https://doi.org/10.1109/TPDS.2005.86

Soham Chakraborty, Shankaranarayanan Krishna, Umang Mathur, and Andreas Pavlogiannis. 2023. How Hard is Weak-Memory Testing? arXiv:2311.04302 [cs.PL]

Marek Chalupa, Krishnendu Chatterjee, Andreas Pavlogiannis, Nishant Sinha, and Kapil Vaidya. 2017. Data-Centric Dynamic Partial Order Reduction. *Proc. ACM Program. Lang.* 2, POPL, Article 31 (dec 2017), 30 pages. https://doi.org/10.1145/3158119

Krishnendu Chatterjee, Andreas Pavlogiannis, and Viktor Toman. 2019. Value-centric dynamic partial order reduction. *Proc. ACM Program. Lang.* 3, OOPSLA (2019), 124:1–124:29. https://doi.org/10.1145/3360550

Yunji Chen, Yi Lv, Weiwu Hu, Tianshi Chen, Haihua Shen, Pengyu Wang, and Hong Pan. 2009. Fast complete memory consistency verification. In *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. 381–392. https://doi.org/10.1109/HPCA.2009.4798276

C. J. Fidge. 1988. Timestamps in message-passing systems that preserve the partial ordering. *Proceedings of the 11th Australian Computer Science Conference* 10, 1 (1988), 56–66.

Florian Furbach, Roland Meyer, Klaus Schneider, and Maximilian Senftleben. 2015. Memory-Model-Aware Testing: A Unified Complexity Analysis. *ACM Trans. Embed. Comput. Syst.* 14, 4, Article 63 (sep 2015), 25 pages. https://doi.org/10.1145/2753761

Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., USA.

Phillip B. Gibbons and Ephraim Korach. 1994. On Testing Cache-Coherent Shared Memories. In *Proceedings of the Sixth Annual ACM Symposium on Parallel Algorithms and Architectures* (Cape May, New Jersey, USA) *(SPAA '94)*. Association for Computing Machinery, New York, NY, USA, 177–188. https://doi.org/10.1145/181014.181328

Phillip B. Gibbons and Ephraim Korach. 1997. Testing Shared Memories. *SIAM J. Comput.* 26, 4 (1997), 1208–1244. https://doi.org/10.1137/S0097539794279614

Alex Gonthmakher, Sergey Polyakov, and Assaf Schuster. 2003. Complexity of Verifying Java Shared Memory Execution. *Parallel Processing Letters* 13, 04 (2003), 721–733. https://doi.org/10.1142/S0129626403001628

Jeff Huang, Patrick O'Neil Meredith, and Grigore Rosu. 2014. Maximal Sound Predictive Race Detection with Control Flow Abstraction. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Edinburgh, United Kingdom) *(PLDI '14)*. ACM, New York, NY, USA, 337–348. https://doi.org/10.1145/2594291.2594315

P.W. Hutto and M. Ahamad. 1990. Slow memory: weakening consistency to enhance concurrency in distributed shared memories. In *Proceedings.,10th International Conference on Distributed Computing Systems*. 302–309. https://doi.org/10.1109/ICDCS.1990.89297

Christian Gram Kalhauge and Jens Palsberg. 2018. Sound Deadlock Prediction. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 146 (oct 2018), 29 pages. https://doi.org/10.1145/3276516

Dileep Kini, Umang Mathur, and Mahesh Viswanathan. 2017. Dynamic Race Prediction in Linear Time. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Barcelona, Spain) *(PLDI 2017)*. ACM, New York, NY, USA, 157–170. https://doi.org/10.1145/3062341.3062374

Michalis Kokologiannakis, Ori Lahav, Konstantinos Sagonas, and Viktor Vafeiadis. 2017. Effective Stateless Model Checking for C/C++ Concurrency. *Proc. ACM Program. Lang.* 2, POPL (2017). https://doi.org/10.1145/3158105

Michalis Kokologiannakis, Ori Lahav, and Viktor Vafeiadis. 2023. Kater: Automating Weak Memory Model Metatheory and Consistency Checking. *Proc. ACM Program. Lang.* 7, POPL, Article 19 (jan 2023), 29 pages. https://doi.org/10.1145/3571212

Michalis Kokologiannakis, Iason Marmanis, Vladimir Gladstein, and Viktor Vafeiadis. 2022. Truly Stateless, Optimal Dynamic Partial Order Reduction. *Proc. ACM Program. Lang.* 6, POPL (2022). https://doi.org/10.1145/3498711

Michalis Kokologiannakis, Azalea Raad, and Viktor Vafeiadis. 2019. Model Checking for Weakly Consistent Libraries. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 96–110. https://doi.org/10.1145/3314221.3314609

Ori Lahav and Udi Boker. 2022. What's Decidable About Causally Consistent Shared Memory? *ACM Trans. Program. Lang. Syst.* 44, 2, Article 8 (apr 2022), 55 pages. https://doi.org/10.1145/3505273

Ori Lahav, Nick Giannarakis, and Viktor Vafeiadis. 2016. Taming release-acquire consistency. In *POPL'16*. ACM, 649–662. https://doi.org/10.1145/2837614.2837643

Ori Lahav and Roy Margalit. 2019. Robustness against Release/Acquire Semantics. In *PLDI 2019*. 126–141. https://doi.org/10.1145/3314221.3314604

Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (jul 1978), 558–565. https://doi.org/10.1145/359545.359563

Sung-Hwan Lee, Minki Cho, Roy Margalit, Chung-Kil Hur, and Ori Lahav. 2023. Putting Weak Memory in Order via a Promising Intermediate Representation. *Proc. ACM Program. Lang.* 7, PLDI, Article 183 (jun 2023), 24 pages. https://doi.org/10.1145/3591297

Weiyu Luo and Brian Demsky. 2021. *C11Tester: A Race Detector for C/C++ Atomics.* Association for Computing Machinery, New York, NY, USA, 630–646. https://doi.org/10.1145/3445814.3446711

C. Manovit and S. Hangal. 2006. Completely verifying memory consistency of test program executions. In *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.* 166–175. https://doi.org/10.1109/HPCA.2006.1598123

Roy Margalit and Ori Lahav. 2021. Verifying Observational Robustness against a C11-Style Memory Model. *Proc. ACM Program. Lang.* 5, POPL, Article 4 (2021), 33 pages. https://doi.org/10.1145/3434285

Umang Mathur, Andreas Pavlogiannis, and Mahesh Viswanathan. 2020. The Complexity of Dynamic Data Race Prediction *(LICS '20)*. Association for Computing Machinery, New York, NY, USA, 713–727. https://doi.org/10.1145/3373718.3394783

Umang Mathur, Andreas Pavlogiannis, and Mahesh Viswanathan. 2021. Optimal Prediction of Synchronization-Preserving Races. *Proc. ACM Program. Lang.* 5, POPL, Article 36 (2021), 29 pages. https://doi.org/10.1145/3434317

Brian Norris and Brian Demsky. 2013. CDSChecker: Checking Concurrent Data Structures Written with C/C++ Atomics. In *OOPSLA'13*.

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

Andreas Pavlogiannis. 2019. Fast, Sound, and Effectively Complete Dynamic Race Prediction. *Proc. ACM Program. Lang.* 4, POPL, Article 17 (dec 2019), 29 pages. https://doi.org/10.1145/3371085

Matthieu Perrin, Achour Mostefaoui, and Claude Jard. 2016. Causal Consistency: Beyond Memory. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Barcelona, Spain) *(PPoPP '16)*. Association for Computing Machinery, New York, NY, USA, Article 26, 12 pages. https://doi.org/10.1145/2851141.2851170

S. Qadeer. 2003. Verifying sequential consistency on shared-memory multiprocessors by model checking. *IEEE Transactions on Parallel and Distributed Systems* 14, 8 (2003), 730–741. https://doi.org/10.1109/TPDS.2003.1225053

Peter Sewell, Susmit Sarkar, Scott Owens, Francesco Zappa Nardelli, and Magnus O. Myreen. 2010. X86-TSO: A Rigorous and Usable Programmer's Model for X86 Multiprocessors. *Commun. ACM* 53, 7 (jul 2010), 89–97. https://doi.org/10.1145/1785414.1785443

CORPORATE SPARC International, Inc. 1994. *The SPARC Architecture Manual (Version 9)*. Prentice-Hall, Inc., USA.

Hünkar Can Tunç, Parosh Aziz Abdulla, Soham Chakraborty, Shankaranarayanan Krishna, Umang Mathur, and Andreas Pavlogiannis. 2023. Optimal Reads-From Consistency Checking for C11-Style Memory Models. *Proc. ACM Program. Lang.* 7, PLDI, Article 137 (jun 2023), 25 pages. https://doi.org/10.1145/3591251

John Wickerson, Mark Batty, Tyler Sorensen, and George A. Constantinides. 2017. Automatically Comparing Memory Consistency Models. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages* (Paris, France) *(POPL '17)*. Association for Computing Machinery, New York, NY, USA, 190–204. https://doi.org/10.1145/3009837.3009838

Matt Windsor, Alastair F. Donaldson, and John Wickerson. 2022. High-coverage metamorphic testing of concurrency support in C compilers. *Software Testing, Verification and Reliability* (2022), e1812.

Received 2023-07-11; accepted 2023-11-07