The Store-Order Consistency Testing Problem for C-like Memory Models

GRACE TAN*, National University of Singapore, Singapore

1 INTRODUCTION

Consistency testing (henceforth "testing") is the problem of determining if an abstract execution of a concurrent program is *feasible* under a given memory model, i.e. does it have a concrete execution consistent with the memory model, and finds many applications in the testing and verification of concurrent software. Model checkers attempt to exhaustively search for buggy executions to prove or disprove the correctness of a program. Since the number of concrete executions can be exponential in the number of abstract ones, practical algorithms limit the search space to abstract executions. To this end, model checkers use consistency testers to check the feasibility of the abstract executions they enumerate [1, 3–5, 8, 11, 16, 17]. In the context of predictive concurrency testing [6, 15], consistency testers are also used to limit the search space to *sound* reorderings.

Consequently, the complexity of testing has been studied across several parametrizations, memory models, and levels of abstractions. In the most abstract setting, consistency testing is done for *reads-value* abstract executions (henceforth "*reads-value testing*"), where the input is the set of events E annotated with values, together with the program order po. For most memory models, reads-value testing is intractable [7, 9, 12–14, 20], so recent work has instead tackled the easier problem of testing *reads-from* abstract executions (henceforth "*reads-from testing*"), where one is additionally given a complete reads-from relation rf [1, 2, 4, 10, 20], enabling scalable model checkers [5, 16]. Most notably, reads-from testing under the RC20 memory model [19], an important formalization of the C/C++ memory model, can be done efficiently in time $O(n(k+d))$, where n, k, and d are the number of events, threads, and memory locations in the execution respectively [23].

Unfortunately, the problem of testing *store-order* abstract executions (henceforth "*store-order testing*"), where one is also given the store-order mo, has largely been understudied. Furthermore, the apparent difficulty of performing a thorough algorithmic investigation for it gives rise to a unique chicken-and-egg problem. Because of the apparent hardness, no applications that use a store-order tester have been developed. But on the other hand, lack of a solid application discourages an active investigation of this problem. However, this cycle is worth breaking. For instance, model checking programs that are very read heavy or have predictable write patterns may require enumerating much fewer store-order abstract executions than reads-from abstract executions. In such a setting, an efficient store-order tester can significantly lower the overall model checking cost.

In this work, we initiate the study of store-order testing by exploring its algorithmic and complexity-theoretic landscape under the memory models RC20, RA, SRA, Relaxed, and Relaxed-Acyclic [10], relevant due to their relation to C/C++, whose informal memory model has even been adapted to the Rust and Go languages [21, 22]. A summary of our results can be found in Table 1.

We show that store-order testing is intractable for (even the RMW-free fragments of) RC20, RA, SRA, and Relaxed-Acyclic. Specifically, for any memory model $\mathcal{M}$ whose strength lies between the RMW-free RA and SRA memory models, store-order testing under $\mathcal{M}$ is NP-complete. Store-order testing under RMW-free Relaxed-Acyclic is not only NP-complete, but also W[1]-hard when parameterized by the number of threads (so a fixed-parameter-tractable algorithm is unlikely).

*Category: Graduate. ACM#: 4838626. Advised by Umang Mathur (NUS), in collaboration with Shankaranarayanan Krishna (IIT Bombay) and Andreas Pavlogiannis (Aarhus University).

Author's address: Grace Tan, National University of Singapore, Singapore, Singapore, gracetan@comp.nus.edu.sg.

Memory model	Upper bound	Lower bound
RC20	$2^{poly(n)}$	NP-complete even for $k = 4$ (sec. 3.1)
SRA $\preceq \mathcal{M} \preceq$ RA	$2^{poly(n)}$	NP-complete (fig. 1a)
Relaxed-Acyclic	$2^{poly(n)}$	W[1]-hard (fig. 1c)
RC20-Ordered	$O(nk)$ (sec. 3.2)	No $O(n^{\omega/2-\epsilon})$ algorithm[1] (fig. 1b)
Relaxed	$\Theta(n)$ (sec. 3.2)	
Single-writer RC20/RC20-Ordered	$O(nk)$ (sec. 3.2)	No $O(n^{\omega/2-\epsilon})$ algorithm[1] (fig. 1b)
Single-writer RA/SRA	$O(nk)$ (sec. 3.2)	No $O(n^{\omega/2-\epsilon})$ algorithm[1] (fig. 1b)
Single-writer Relaxed-Acyclic	$\Theta(n)$ (sec. 3.2)	
Single-writer Relaxed	$\Theta(n)$ (sec. 3.2)	

Table 1. The complexity landscape of store-order consistency testing.

Finally, store-order testing under RMW-free RC20 is NP-complete even when the input is limited to only 4 threads, the strongest intractability result we have.

We next identify a practically motivated fragment for which the store-order testing problem becomes tractable. Specifically, the *single-writer* fragment, i.e. the fragment where every location is written to by exactly one thread, admits an efficient algorithm, running in time $O(nk)$, and in $O(n)$ for specific subfragments. We extend this result by generalizing the single-writer constraint to a weaker one, which can be cleanly formalized as a new memory model we call 'RC20-Ordered'. We also show that these algorithms are optimal or nearly optimal by establishing conditional super-linear lower bounds[1] that also apply to the respective RMW-free fragments.

We find that the RC20-Ordered memory model exhibits salient properties. Our proofs establish and heavily use a partial order on executions that compares their $\mathrm{mo}^?$; rf^2 relations. This led to the surprising result that there is a *unique* minimal consistent concrete execution in this partial order, hinting that the "Ordered" strengthening may be interesting to study in its own right.

2 PRELIMINARIES

Executions[3]**.** A (store-order) *abstract* execution is a tuple $\widetilde{G} \triangleq \langle E, \mathrm{po}, \mathrm{mo}\rangle$, where E is a set of events annotated with values, po is the program order, and mo is the store order. Given an abstract execution $\widetilde{G}$, a *concrete* execution (for $\widetilde{G}$) is a tuple $G \triangleq \langle E, \mathrm{po}, \mathrm{mo}, \mathrm{rf}\rangle$ where E, po, mo match those in $\widetilde{G}$ and the reads-from relation rf is consistent with the locations and values of each event.

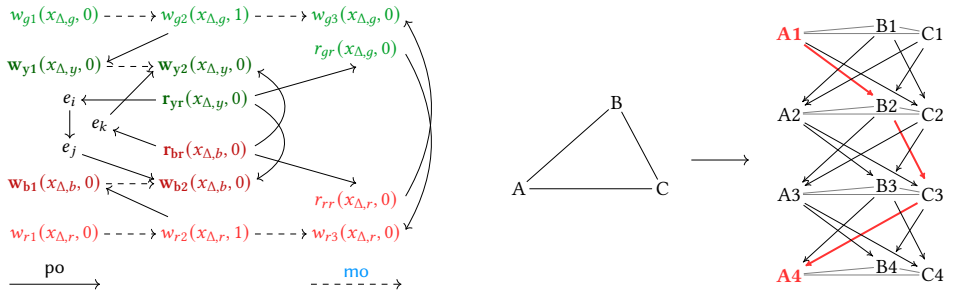
Memory models. An (axiomatic) *memory model* is a set of constraints $\mathcal{M}$ that define whether a given concrete execution is consistent. For example, the porf-acyclicity axiom is written $acy(\mathrm{po} \cup \mathrm{rf})$ and states that the relation po $\cup$ rf must be acyclic. A concrete execution G *satisfies* or is *consistent under* a memory model $\mathcal{M}$ (denoted $G \models \mathcal{M}$) if G satisfies all the constraints of $\mathcal{M}$. An abstract execution $\widetilde{G}$ is *feasible* under $\mathcal{M}$ if there exists some concrete execution G for $\widetilde{G}$ that satisfies $\mathcal{M}$. The formalizations of the memory models pertaining to this work are presented in Appendix A.

Comparing strengths of memory models and defining fragments. $\mathcal{M}_1$ is *stronger* than $\mathcal{M}_2$ (denoted $\mathcal{M}_1 \preceq \mathcal{M}_2$) if for all concrete executions G, $G \models \mathcal{M}_1 \implies G \models \mathcal{M}_2$. $\mathcal{M}_1$ is a *fragment* of $\mathcal{M}_2$ if it only adds constraints, i.e. $\mathcal{M}_1 = \mathcal{M}_2 \cup \mathcal{C}$ for some set of constraints $\mathcal{C}$. Note that the term "fragment" tends to refer to *syntactical* restrictions in the literature, e.g. constraints placed on the program itself. For example, the Relaxed-Acyclic fragment of RC20 requires all operations to have memory order Relaxed. This nuance is not important in this work, so our definition is simplified.

[1]There is no algorithm that runs in time $O(n^{\omega/2-\epsilon})$ for any $\epsilon > 0$, where ω is the matrix multiplication constant for Boolean Matrix Multiplication (believed to be > 2).

[2]$R; S$, $R^?$, R^+ denote the composition of relations R and S, reflexive and transitive closures of relations R and S respectively.

[3]To be precise, there are additional promises placed on po, mo, etc., which are not relevant in this extended abstract.



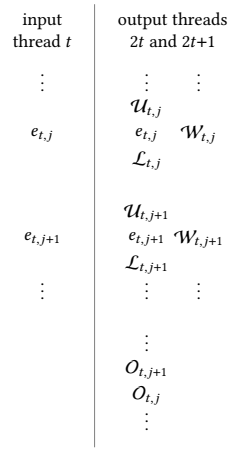
(a) Gadget for NP-hardness of SRA $\preceq \mathcal{M} \preceq$ RA (b) Idea for triangle hardness of single-writer RA/SRA

Fig. 1. Various parts of this work *not* covered in this extended abstract.

3 TECHNIQUES USED IN OUR RESULTS

3.1 Hardness of RC20

We show that the store-order testing problem under the RC20 memory model is NP-complete even when the input only has 4 threads. Our reduction from 3-SAT uses a novel technique based on *toggle limits*. A toggle happens on a thread t for a memory location x when the view of t for x changes value. The number of toggles can be restricted by simply limiting the number of writes to each location. We use two main ideas, that Relaxed reads with different values can be used to force toggles, and that Release-Acquire operations can be used to "join" threads' views together. These ideas are combined in our gadget such that the number of forced toggles is controlled based on the selected variable assignment, ensuring some thread runs out of toggles whenever a clause is falsified.



(c) Schematic for W[1]-hardness of Relaxed-Acyclic

3.2 An efficient algorithm for the single-writer fragments

We show runtime upper bounds on the store-order testing problem as listed in Table 1. Our algorithms are either optimal or nearly optimal due to our conditional super-linear lower bounds[1]. All of our upper bound results follow from the main algorithm for the new memory model RC20-Ordered that generalizes the single-writer fragment, which we describe next.

The RC20-Ordered memory model. RC20-Ordered is derived from RC20 with two modifications. First, we make release sequences unbounded by replacing rf^+ with $\mathsf{mo}^?;\mathsf{rf}$, changing the synchronizes-with and happens-before relations. Next, we replace porf-acyclicity with hbo^{RA}-acyclicity, similarly strengthening rf to $\mathsf{mo}^?;\mathsf{rf}$. Surprisingly, under this strengthening, the minimal consistent concrete execution is *unique*, a corollary which extends to the other single-writer models.

Testing algorithm for RC20-Ordered. Our algorithm starts with an partial execution with an empty reads-from relation $\mathsf{rf} = \emptyset$, and works in a greedy fashion, executing one event at a time. The execution order is controlled such that it's consistent with hbo^{RA} at all times. When executing reads, an appropriate edge must be added to rf. Among the candidate writes which don't break consistency, we pick the earliest candidate in mo. This greedy choice ensures that the execution graph minimizes the $\mathsf{mo}^?;\mathsf{rf}$ relation at all times. Our proofs of correctness and uniqueness of the minimal consistent execution heavily rely on this minimality.

4 ONGOING AND FUTURE WORK

We aim to further refine the complexity landscape, starting with testing under SRA $\preceq \mathcal{M} \preceq$ RA parameterized by k, and either show its W[1]-hardness or find a FPT algorithm. Analogously, we aim to refine the complexity of testing under Relaxed-Acyclic in the bounded threads setting.

REFERENCES

[1] 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.

[2] Parosh Aziz Abdulla, Jatin Arora, Mohamed Faouzi Atig, and Shankara Narayanan Krishna. 2019. Verification of programs under the release-acquire semantics. In *PLDI 2019*, Kathryn S. McKinley and Kathleen Fisher (Eds.). ACM, 1117–1132. https://doi.org/10.1145/3314221.3314649

[3] 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

[4] 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

[5] 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.

[6] Zhendong Ang and Umang Mathur. 2024. Predictive Monitoring with Strong Trace Prefixes. In *Computer Aided Verification*, Arie Gurfinkel and Vijay Ganesh (Eds.). Springer Nature Switzerland, Cham, 182–204.

[7] 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

[8] 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

[9] 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

[10] 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 (Jan. 2024), 32 pages. https://doi.org/10.1145/3632908

[11] 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

[12] 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

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

[14] 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

[15] Shiyou Huang and Jeff Huang. 2016. Maximal causality reduction for TSO and PSO. *SIGPLAN Not.* 51, 10 (Oct. 2016), 447–461. https://doi.org/10.1145/3022671.2984025

[16] 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

[17] 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

[18] Ori Lahav, Viktor Vafeiadis, Jeehoon Kang, Chung-Kil Hur, and Derek Dreyer. 2017. Repairing Sequential Consistency in C/C++11. In *PLDI 2017*. 618–632. https://doi.org/10.1145/3062341.3062352 Technical Appendix Available at https://plv.mpi-sws.org/scfix/full.pdf.

[19] 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

[20] 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

[21] The Go language team. 2022. The Go Memory Model. https://go.dev/ref/mem. Accessed: 2024-11-22.

[22] The Rustonomicon authors. 2024. Atomics. https://doc.rust-lang.org/nomicon/atomics.html. Accessed: 2024-11-22 at commit https://github.com/rust-lang/nomicon/blob/0674321898cd454764ab69702819d39a919afd68/src/atomics.md.

[23] 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

A FORMALIZATIONS OF MEMORY MODELS USED

We now add details that were omitted from the extended abstract. The formalization presented here is consistent with [10] and [19] with the exception of the Relaxed memory model, which is different from [10] because we use a separate derivation.

However, the precise formalization has not been published in this exact form, to the author's knowledge. We use a different but more intuitive formalization of RC20 to motivate the derivation of RC20w (RC20 without porf-acyclicity), and RC20-Ordered (RC20 with "infinite" release sequences).

Our formalization of RC20 can be thought of as an adaptation of the technique used to formalize RC11 [18], but applied to the RC20 memory model. To further justify our formalization, we prove "common sense" intuitive properties that have not been formalized in the literature to the author's knowledge. For instance, we show that atomicity is equivalent to the irreflexivity of com, and that com is a strict weak order under atomicity. We also show that hb-acyclicity is in fact redundant, as it is implied by atomicity and coherence.

A.1 Detailed preliminaries

Events. An event is either a memory or fence event.

A *memory event* is a tuple $\langle \mathsf{id}, \mathsf{tid}, \mathsf{op}, \mathsf{loc}, \mathsf{ord}, \mathsf{rval}, \mathsf{wval} \rangle$ which denote the event id, thread id, operation, accessed memory location, memory order, value read, and value written respectively. A *fence event* is a tuple $\langle \mathsf{id}, \mathsf{tid}, \mathsf{ord} \rangle$ which denote the event id, thread id, and memory order respectively.

The purpose of the event id is merely to disambiguate distinct memory events that happen to share the same attributes, and as such, we will not place much emphasis on it. The operation is either a read, write, or read-modify-write (henceforth RMW). We use the letters `r`, `w`, and `rmw` to denote each type of operation. The memory orders are, arranged in increasing order of strength, $\mathsf{rlx} \sqsubseteq \mathsf{acq}, \mathsf{rel} \sqsubseteq \mathsf{acqrel}$. Given an event e, we allow accessing its properties with an object-notation style, e.g. $e.\mathsf{ord}$ is the memory order of e.

When the set of events is understood from context, we denote W, R, and F the writes, the reads, and the fence events of E respectively. RMWs count as both writes as well as reads, so we denote $M = W \cap R$ the set of RMWs of E.

Given a set of events or relation over events S and either a thread identifier t, a memory location x, or a memory order ord, we denote S_t, S_x, S^{ord} the set or relation restricted to events on thread t, accessing location x, or having memory order ord respectively. We similarly denote $S^{\sqsupseteq \mathsf{ord}} \triangleq \{e \in S \mid e.\mathsf{ord} \sqsupseteq \mathsf{ord}\}$ the set of events whose memory order is at least as strong as ord.

Executions. An *abstract* execution is a tuple $\widetilde{G} \triangleq \langle E, \mathsf{po}, \mathsf{mo} \rangle$, where E is a set of events annotated with values, and po, mo are binary relations over E. The program order po is a strict total order on the events of each thread. The store-order relation mo (also known as modification order) is a strict total order on the writes to each location.

Given an abstract execution $\widetilde{G}$, a *concrete* execution (for $\widetilde{G}$) is a tuple $G \triangleq \langle E, \mathsf{po}, \mathsf{mo}, \mathsf{rf} \rangle$ where E, po, mo match those in $\widetilde{G}$, such that the reads-from relation rf is a relation from writes to reads $\mathsf{rf} \subseteq W \times R$, rf is consistent with the locations and values of each event, i.e. for all $(e_1, e_2) \in E$, e_1's location and value written are equal to e_2's location and value read, and finally its inverse rf^{-1} is a function, i.e. every read reads from at most one write.

A summary of the relations defined in each abstract or concrete execution is given in Fig. A1.

Notation for relations. We denote $R; S$ the composition of R and S, denote $R^?, R^+, R^*$ the reflexive, transitive, and reflexive-transitive closures of R, and denote R^{-1} the inverse relation of R. Given a

$$\mathrm{po} \subseteq \bigsqcup_{\text{thread } t} E_t \times E_t \qquad \mathrm{mo} \subseteq \bigsqcup_{\text{location } x} E_x \times E_x \qquad \mathrm{rf} \subseteq \bigsqcup_{\text{location } x} W_x \times R_x$$

$$\mathrm{po}_t \text{ is a total order} \qquad \mathrm{mo}_x \text{ is a total order} \qquad \mathrm{rf}^{-1} \text{ is a function}$$

Fig. A1. Summary of the types of each relation in an abstract or concrete execution.

set of events S, we denote $[S]$ the identity relation on S. We also allow infix notation for relations, e.g. $e_1 \ \mathrm{mo} \ e_2 \ \mathrm{rf} \ e_3 \iff (e_1, e_2) \in \mathrm{mo} \wedge (e_2, e_3) \in \mathrm{rf}$.

Given a relation R, we denote $irr(R) \triangleq \forall e.((e,e) \notin R)$ the statement that R is irreflexive and denote $acy(R) \triangleq irr(R^+)$ the statement that R is acyclic.

Memory models. An (axiomatic) *memory model* is a set of statements $\mathcal{M}$ with free variables E, po, mo, rf. A concrete execution G *satisfies* or is *consistent* under a memory model $\mathcal{M}$ (denoted $G \models \mathcal{M}$) if all its statements are true when E, po, mo, rf are interpreted according to G. An abstract execution $\widetilde{G}$ is *feasible* under $\mathcal{M}$ if there exists some concrete execution G such that $G \models \mathcal{M}$.

Comparing strengths of memory models and defining fragments. $\mathcal{M}_1$ is *stronger* than $\mathcal{M}_2$ (denoted $\mathcal{M}_1 \preceq \mathcal{M}_2$) if for all concrete executions G, $G \models \mathcal{M}_1 \implies G \models \mathcal{M}_2$. $\mathcal{M}_1$ is a *fragment* of $\mathcal{M}_2$ if it only adds constraints, i.e. $\mathcal{M}_1 = \mathcal{M}_2 \cup \mathcal{C}$ for some set of axioms $\mathcal{C}$. Note that in the literature, the term "fragment" tends to refer to *syntactical* restrictions, i.e. constraints placed on the program itself. For example, the Relaxed-Acyclic fragment of RC20 requires all operations to have memory order Relaxed. This nuance is not important in this work, so our definition is simplified.

A.2 Derived relations

We now define relevant relations derived from po, mo, rf.

We denote the *from-reads* relation $\mathrm{fr} \triangleq (\mathrm{rf}^{-1}; \mathrm{mo}) \setminus [E]$, which relates reads and writes $\mathrm{fr} \subseteq R \times W$. Intuitively, $r \ \mathrm{fr} \ w$ means that w modifies some location *"after"* r reads it.

We denote the *coherence-order* relation $\mathrm{com} \triangleq (\mathrm{mo} \cup \mathrm{rf} \cup \mathrm{fr})^+$. This relates all events accessing the same location $\mathrm{com} \subseteq \bigsqcup_{\text{location } x} E_x \times E_x$. Intuitively, the coherence order on a location com_x is the notion of "time" as viewed from the perspective of x, so $e_1 \ \mathrm{com} \ e_2$ intuitively means that e_2 accesses some location *"after"* e_1 accesses it. The coherence axiom (described later) is used to make this intuitive notion correct.

We denote the *release-sequence* relation (for RC20) $\mathrm{rs} \triangleq \mathrm{rf}^*; [W]$. If $w_1 \ \mathrm{rs} \ w_2$, we say that w_2 participates in w_1's release sequence. We can say that each write w_1 is associated with a release sequence, a sequence of writes starting with the w_1 itself.

We denote the *synchronizes-with* relation (for RC20)

$$\mathrm{sw} \triangleq [E^{\sqsupseteq \mathrm{rel}}]; ([F]; \mathrm{po})^?; \mathrm{rs}; \mathrm{rf}; (\mathrm{po}; [F])^?; [E^{\sqsupseteq \mathrm{acq}}].$$

It is helpful to think of sw as an extension of the simpler relation for the fence-free fragment, under which $\mathrm{sw} = [E^{\sqsupseteq \mathrm{rel}}]; \mathrm{rs}; \mathrm{rf}; [E^{\sqsupseteq \mathrm{acq}}]$. Under this fragment, we have $w \ \mathrm{sw} \ r$ iff w is tagged `rel` or stronger, r reads from some write in w's release sequence, and r is tagged `acq` or stronger. The full definition for sw indicates how fence events participate in the synchronizes-with relation, but we do not go into detail for its intuition as it is not required for this work.

Since $\mathrm{rs} \triangleq \mathrm{rf}^*$ in RC20, it is much more common to formalize sw using the equivalent definition

$$\mathrm{sw} \triangleq [E^{\sqsupseteq \mathrm{rel}}]; ([F]; \mathrm{po})^?; \mathrm{rf}^+; (\mathrm{po}; [F])^?; [E^{\sqsupseteq \mathrm{acq}}].$$

We denote the happens-before relation (for RC20) $\mathrm{hb} \triangleq (\mathrm{po} \cup \mathrm{sw})^+$. This intuitively captures the notion of time as viewed from any particular event.

We denote the *release-sequence-ordered* relation (for RC20-Ordered) $\mathrm{rso} \triangleq \mathrm{mo}^?$. This means that w_2 participates in w_1's release sequence simply if w_2 is not mo-ordered before w_1 — release sequences "last forever".

We denote the synchronizes-with-ordered relation (for RC20-Ordered)

$$\mathsf{swo} \triangleq [E^{\sqsupseteq \mathsf{rel}}]; ([F]; \mathsf{po})^?; \mathsf{rso}; \mathsf{rf}; (\mathsf{po}; [F])^?; [E^{\sqsupseteq \mathsf{acq}}].$$

This strengthens the notion of release sequences from $\mathsf{rs} = \mathsf{rf}^*$ to $\mathsf{rso} = \mathsf{mo}^?$.

We denote the happens-before-ordered relation (for RC20-Ordered), $\mathsf{hbo} \triangleq (\mathsf{po} \cup \mathsf{swo})^+$. This also captures the notion of time as viewed from an event, but under the RC20-Ordered memory model.

Under the Release-Acquire fragment where every event's memory order is as strong as possible ($R = E^{\sqsupseteq \mathsf{acq}}, W = E^{\sqsupseteq \mathsf{rel}}, M = E^{\mathsf{acqrel}}$), the happens-before and happens-before-ordered relations can be simplified. We denote these simplifications $\mathsf{hb}^{RA} \triangleq (\mathsf{po} \cup \mathsf{rf}^+)^+$ and $\mathsf{hbo}^{RA} \triangleq (\mathsf{po} \cup (\mathsf{mo}^?; \mathsf{rf}))^+$ respectively. Observe that in the case of hb^{RA}, the transitive rf^+ edge can be simplified to get the alternative definition $\mathsf{hb}^{RA} \triangleq (\mathsf{po} \cup \mathsf{rf})^+$, and for consistency with the literature, with refer to hb^{RA} with the more common name $\mathsf{porf} \triangleq (\mathsf{po} \cup \mathsf{rf})^+$.

Formally, we can say that under the Release-Acquire fragment, $\mathsf{hb} = \mathsf{hb}^{RA}$, porf and $\mathsf{hbo} = \mathsf{hbo}^{RA}$. However, note that we still use these relations outside of the Release-Acquire fragment.

The full list of derived relations is given in Fig. A2

$$\begin{aligned}
\mathsf{fr} &\triangleq (\mathsf{rf}^{-1}; \mathsf{mo}) \setminus [E] \\
\mathsf{com} &\triangleq (\mathsf{mo} \cup \mathsf{rf} \cup \mathsf{fr})^+ \\
\mathsf{rs} &\triangleq \mathsf{rf}^* \\
\mathsf{rso} &\triangleq \mathsf{mo}^? \\
\mathsf{sw} &\triangleq [E^{\sqsupseteq \mathsf{rel}}]; ([F]; \mathsf{po})^?; \mathsf{rs}; \mathsf{rf}; (\mathsf{po}; [F])^?; [E^{\sqsupseteq \mathsf{acq}}] \\
\mathsf{swo} &\triangleq [E^{\sqsupseteq \mathsf{rel}}]; ([F]; \mathsf{po})^?; \mathsf{rso}; \mathsf{rf}; (\mathsf{po}; [F])^?; [E^{\sqsupseteq \mathsf{acq}}] \\
\mathsf{hb} &\triangleq (\mathsf{po} \cup \mathsf{sw})^+ \\
\mathsf{hbo} &\triangleq (\mathsf{po} \cup \mathsf{swo})^+ \\
\mathsf{porf}, \mathsf{hb}^{RA} &\triangleq (\mathsf{po} \cup \mathsf{rf})^+ \\
\mathsf{hbo}^{RA} &\triangleq (\mathsf{po} \cup (\mathsf{mo}; \mathsf{rf}))^+
\end{aligned}$$

Fig. A2. The full list of derived relations used.

A.3 Consistency axioms

We now define the statements that will be used in the various memory models. These statements are often named consistency axioms or simply axioms in the context of memory models. We give these axioms names as the various memory models share many of their axioms.

A.3.1 Core consistency axioms. The most important axiom is *atomicity*, which states that every RMW must read from the immediately preceding write. We formalize this with the axiom $\mathit{irr}(\mathsf{com})$, the *irreflexivity of* com. It can be shown that this is equivalent to the original definition: so long as mo and rf are well-formed, every RMW reads from the immediately preceding write iff com_x is a strict weak order iff com is irreflexive. This justifies the intuition that com_x is the "notion of time" as viewed from the perspective of x, as it ensures there are no cycles in com_x.

Analogously, another important axiom is *acyclicity of happens-before*, to justify the intuition that hb (or hbo) is a valid "notion of time" by ensuring that there are no cycles. This is formalized with the statement $\mathit{irr}(\mathsf{hb})$ (or $\mathit{irr}(\mathsf{hbo})$).

The next axiom is *coherence*, which intuitively states that each variable's notion of time is consistent with each of its events' notion of time. This is formalized (for RC20) with the statement $irr(\mathsf{hb};\mathsf{com})$. Intuitively, if an event e_1 happens before another e_2, this axiom states that e_2 cannot be coherence ordered before e_1. For RC20-Ordered, we use the appropriate notions of time, and formalize it with the statement $irr(\mathsf{hbo};\mathsf{com})$.

The above axioms formalize the majority of the C/C++ memory model, not accounting for SC accesses (which we did not even formalize a memory order for), or parts of the standard that are informally specified (memory fairness, out of thin air behaviors). As such, these form the core of any memory model we consider.

RC20 proposes the axiom porf-acyclicity to formalize the prohibition of out of this air behaviors, $irr(\mathsf{porf})$. In RC20-Ordered, the analogous axiom is hbo^{RA}-acyclicity $irr(\mathsf{hbo}^{RA})$, replacing RC20's notion of time with RC20-Ordered's.

A summary of the core consistency axioms is in Fig. A3.

$$\begin{aligned}
\text{atomicity} &\triangleq irr(\mathsf{com})\\
\mathsf{hb}\text{-acyclicity} &\triangleq irr(\mathsf{hb})\\
\mathsf{hbo}\text{-acyclicity} &\triangleq irr(\mathsf{hbo})\\
\text{coherence} &\triangleq irr(\mathsf{hb};\mathsf{com})\\
\text{coherence-ordered} &\triangleq irr(\mathsf{hbo};\mathsf{com})\\
\mathsf{porf}\text{-acyclicity} &\triangleq irr(\mathsf{porf})\\
\mathsf{hbo}^{RA}\text{-acyclicity} &\triangleq irr(\mathsf{hbo}^{RA})
\end{aligned}$$

Fig. A3. The core consistency axioms.

A.3.2 Redundancy of acyclicity of happens-before. Interestingly, one of the core consistency axioms is completely redundant, in the sense that if atomicity and coherence are assumed, hb-acyclicity is automatically implied, and similarly for the ordered variants.

$$\begin{aligned}
\text{atomicity} \wedge \text{coherence} &\implies \mathsf{hb}\text{-acyclicity}\\
\text{atomicity} \wedge \text{coherence-ordered} &\implies \mathsf{hbo}\text{-acyclicity}
\end{aligned}$$

A.3.3 Alternative coherence axioms. It is common in other formalisms to split the coherence axiom into parts. At the most extreme, one can split the coherence axiom into all 4 combinations of write-read coherence, formalizing the constraint placed on com when a write happens-before a read, and similarly for the combinations read-read coherence, write-write coherence, and read-write coherence.

This follows the decomposition of com into each type of coherence, the correctness of which is a corollary of the proof that com is a strict weak order, assuming atomicity.

$$\mathsf{com} \triangleq \mathsf{comRW} \cup \mathsf{comRR} \cup \mathsf{comWW} \cup \mathsf{comWR}$$

$$\mathsf{comRW} \triangleq \mathsf{fr} \qquad \mathsf{comRR} \triangleq \mathsf{fr};\mathsf{rf} \qquad \mathsf{comWW} \triangleq \mathsf{mo} \qquad \mathsf{comWR} \triangleq \mathsf{mo}^{?};\mathsf{rf}$$

Decomposing the coherence axioms according to the above decomposition of com, we obtain the axioms WR-coherence(-ordered), RR-coherence(-ordered), WW-coherence(-ordered), and RW-coherence(-ordered). As the decomposition holds whenever com is a strict weak order (atomicity), the decomposed coherence axioms are equivalent to the original coherence axioms under atomicity.

Instead of fully decomposing each coherence axiom into all 4 combinations, it is also common to decompose the coherence axiom into 2 halves. Read-coherence combines WR-coherence and RR-coherence, while write-coherence combines WW-coherence and RW-coherence.

It is also common to formalize write coherence under porf-acyclicity. This allows us to drop the case $irr(\mathsf{hb}; \mathsf{rf})$ as it is implied by porf-acyclicity, giving us the axioms write-coherence(-ordered)-under-porf.

The decomposed coherence axioms are shown in Fig. A4 and their relationships are shown in Fig. A5.

$$\begin{aligned}
\text{WR-coherence} &\triangleq irr(\mathsf{hb}; \mathsf{fr}) \\
\text{RR-coherence} &\triangleq irr(\mathsf{hb}; \mathsf{fr}; \mathsf{rf}) \\
\text{WW-coherence} &\triangleq irr(\mathsf{hb}; \mathsf{mo}) \\
\text{RW-coherence} &\triangleq irr(\mathsf{hb}; \mathsf{mo}^?; \mathsf{rf}) \\
\text{read-coherence} &\triangleq irr(\mathsf{hb}; \mathsf{fr}; \mathsf{rf}^?) \\
\text{write-coherence} &\triangleq irr(\mathsf{hb}; (\mathsf{mo} \cup (\mathsf{mo}^?; \mathsf{rf}))) \\
\text{write-coherence-under-porf} &\triangleq irr(\mathsf{hb}; \mathsf{mo}; \mathsf{rf}^?) \\
\text{WR-coherence-ordered} &\triangleq irr(\mathsf{hbo}; \mathsf{fr}) \\
\text{RR-coherence-ordered} &\triangleq irr(\mathsf{hbo}; \mathsf{fr}; \mathsf{rf}) \\
\text{WW-coherence-ordered} &\triangleq irr(\mathsf{hbo}; \mathsf{mo}) \\
\text{RW-coherence-ordered} &\triangleq irr(\mathsf{hbo}; \mathsf{mo}^?; \mathsf{rf}) \\
\text{read-coherence-ordered} &\triangleq irr(\mathsf{hbo}; \mathsf{fr}; \mathsf{rf}^?) \\
\text{write-coherence-ordered} &\triangleq irr(\mathsf{hbo}; (\mathsf{mo} \cup (\mathsf{mo}^?; \mathsf{rf}))) \\
\text{write-coherence-ordered-under-porf} &\triangleq irr(\mathsf{hbo}; \mathsf{mo}; \mathsf{rf}^?)
\end{aligned}$$

Fig. A4. The decomposed variants of the coherence axiom.

A.3.4 Alternative atomicity axioms. Other formalisms similarly do not formalize the atomicity axiom the same way. As before, we split atomicity into forward-atomicity and backward-atomicity, where the first constraint states that an RMW must not read from a write the future, while the second constraint states that an RMW must not read from a write in the past. Observe that $\mathsf{comWW} = \mathsf{mo}$ is already irreflexive because it is a strict total order, and $\mathsf{comRW} = \mathsf{fr}$ is irreflexive by force, because the identity relation is subtracted away. So the two atomicity cases correspond to the irreflexivity of comWR and comRR respectively. These are shown in Fig. A6.

A.3.5 Axioms for defining fragments. We define the Release-Acquire, Relaxed, single-writer, and single-location fragments in Fig. A7. Note that these constraints are all syntactic restrictions on what's allowed in an abstract or concrete execution, so they are fragments in the nuanced sense as well.

The Release-Acquire and Relaxed fragments restrict which memory orders are allowed in the program. The Release-Acquire fragment states that every read has memory order at least acq or stronger and every write has memory order at least rel or stronger. This means RMWs have memory order acqrel. The Relaxed fragment states that every event must have memory order rlx.

The single-writer fragment restricts where writes are performed, and states that every location is written to by exactly one thread. This is formalized by saying that if two writes share a location, they must also share the same thread.

$$
\begin{aligned}
& \text{atomicity} \wedge \text{coherence} \\
\iff\ & \text{atomicity} \wedge \text{WR-coherence} \\
& \wedge \text{RR-coherence} \\
& \wedge \text{WW-coherence} \\
& \wedge \text{RW-coherence} \\
& \text{atomicity} \wedge \text{coherence-ordered} \\
\iff\ & \text{atomicity} \wedge \text{WR-coherence-ordered} \\
& \wedge \text{RR-coherence-ordered} \\
& \wedge \text{WW-coherence-ordered} \\
& \wedge \text{RW-coherence-ordered} \\
\text{read-coherence} \iff\ & \text{WR-coherence} \wedge \text{RR-coherence} \\
\text{write-coherence} \iff\ & \text{WW-coherence} \wedge \text{RW-coherence} \\
\text{read-coherence-ordered} \iff\ & \text{WR-coherence-ordered} \wedge \text{RR-coherence-ordered} \\
\text{write-coherence-ordered} \iff\ & \text{WW-coherence-ordered} \wedge \text{RW-coherence-ordered} \\
& \text{porf-acyclicity} \wedge \text{write-coherence} \\
\iff\ & \text{porf-acyclicity} \wedge \text{write-coherence-under-porf} \\
& \text{porf-acyclicity} \wedge \text{write-coherence-ordered} \\
\iff\ & \text{porf-acyclicity} \wedge \text{write-coherence-ordered-under-porf}
\end{aligned}
$$

Fig. A5. Relationships between the decomposed coherence axioms.

$$
\begin{aligned}
\text{forward-atomicity} &\triangleq \mathit{irr}(\mathsf{mo}^?; \mathsf{rf}) \\
\text{backward-atomicity} &\triangleq \mathit{irr}(\mathsf{fr}; \mathsf{mo}) \\
\text{atomicity} &\iff \text{forward-atomicity} \wedge \text{backward-atomicity}
\end{aligned}
$$

Fig. A6. Alternative atomicity axioms.

$$
\begin{aligned}
\text{ra-fragment} &\triangleq (\forall e \in R, e.\mathsf{ord} \sqsubseteq \mathsf{acq}) \wedge (\forall e \in W, e.\mathsf{ord} \sqsubseteq \mathsf{rel}) \\
\text{rlx-fragment} &\triangleq \forall e \in E, e.\mathsf{ord} = \mathsf{rlx} \\
\text{single-writer} &\triangleq \forall \text{location } x, \forall w_1, w_2 \in W_x, w_1.\mathsf{tid} = w_2.\mathsf{tid} \\
\text{single-location} &\triangleq \forall e_1, e_2 \in E, e_1.\mathsf{loc} = e_2.\mathsf{loc}
\end{aligned}
$$

Fig. A7. Axioms for defining fragments.

Finally, the single-location fragment restricts the number of memory locations to just one. This is formalized by saying every event shares the same location.

A.3.6 Coherence axioms for Release-Acquire. Under the Release-Acquire fragment, hb = porf, which allows us to make some simplifications as optional $\mathsf{rf}^?$ edges may be absorbed into porf.

For the SRA memory model, we'll need a strengthening of write coherence, where mo must be *transitively* consistent with porf. These axioms are shown in Fig. A8.

$$\begin{aligned}\text{read-coherence-ra} &\triangleq \mathit{irr}(\text{porf}; \text{fr})\\ \text{write-coherence-ra} &\triangleq \mathit{irr}(\text{porf}; \text{mo})\\ \text{strong-write-coherence-ra} &\triangleq \mathit{acy}(\text{porf} \cup \text{mo})\end{aligned}$$

Fig. A8. Simplified coherence axioms under the Release-Acquire fragment.

A.3.7 Coherence axioms for Relaxed. Under the Relaxed fragment, hb = po, which allows us to make some simplifications as well. Since write coherence is unwieldy under this fragment, we use the split versions of write coherence there. These axioms are shown in Fig. A8.

$$\begin{aligned}\text{read-coherence-rlx} &\triangleq \mathit{irr}(\text{po}; \text{fr}; \text{rf}^?)\\ \text{write-coherence-rlx} &\triangleq \mathit{irr}(\text{po}; (\text{mo} \cup (\text{mo}^?; \text{rf})))\\ \text{WW-coherence-rlx} &\triangleq \mathit{irr}(\text{po}; \text{mo})\\ \text{RW-coherence-rlx} &\triangleq \mathit{irr}(\text{po}; \text{mo}^?; \text{rf})\end{aligned}$$

Fig. A8. Simplified coherence axioms under the Relaxed fragment.

A.4 Memory models

We finally define the memory models we use in this work, which are RC20, RC20w (RC20 without porf-acyclicity), RC20-Ordered (our generalization of the single-writer constraint), RA, SRA, Relaxed-Acyclic, and Relaxed.

$$\begin{aligned}\text{RC20w} &\triangleq \{\text{atomicity}, \text{coherence}\}\\ &= \{\mathit{irr}(\text{com}), \mathit{irr}(\text{hb}; \text{com})\}\\ \text{RC20} &\triangleq \{\text{porf-acyclicity}, \text{atomicity}, \text{coherence}\}\\ &= \{\mathit{irr}(\text{porf}), \mathit{irr}(\text{com}), \mathit{irr}(\text{hb}; \text{com})\}\\ \text{RC20-Ordered} &\triangleq \{\text{hbo}^{RA}\text{-acyclicity}, \text{atomicity}, \text{coherence-ordered}\}\\ &= \{\mathit{irr}(\text{hbo}^{RA}), \mathit{irr}(\text{com}), \mathit{irr}(\text{hbo}; \text{com})\}\end{aligned}$$

RA, Relaxed, and Relaxed-Acyclic are defined as fragments of RC20w and RC20. SRA is defined to be a strengthening of RA that enforces strong-write-coherence.

$$\begin{aligned}\text{RA} &\triangleq \text{RC20w} \cup \{\text{ra-fragment}\}\\ &\iff \text{RC20} \cup \{\text{ra-fragment}\}\\ \text{SRA} &\triangleq \text{RA} \cup \{\text{strong-write-coherence-ra}\}\\ \text{Relaxed} &\triangleq \text{RC20w} \cup \{\text{rlx-fragment}\}\\ \text{Relaxed-Acyclic} &\triangleq \text{RC20} \cup \{\text{rlx-fragment}\}\end{aligned}$$

Using the alternative coherence axioms covered earlier, we have following simplifications for RA, SRA, Relaxed, and Relaxed-Acyclic. These simplifications are useful in proofs as they allow one to more directly pinpoint the source of inconsistency.

$$
\begin{aligned}
\text{RA} &\iff \{\text{ra-fragment}, \text{porf-acyclicity}, \text{atomicity}, \\
&\qquad \text{read-coherence-ra}, \text{write-coherence-ra}\} \\
&= \{\text{ra-fragment}, \mathit{irr}(\text{porf}), \mathit{irr}(\text{com}), \mathit{irr}(\text{porf}; \text{fr}), \mathit{irr}(\text{porf}; \text{mo})\} \\
\text{SRA} &\iff \{\text{ra-fragment}, \text{porf-acyclicity}, \text{atomicity}, \\
&\qquad \text{read-coherence-ra}, \text{strong-write-coherence-ra}\} \\
&= \{\text{ra-fragment}, \mathit{irr}(\text{porf}), \mathit{irr}(\text{com}), \mathit{irr}(\text{porf}; \text{fr}), \mathit{acy}(\text{porf} \cup \text{mo})\} \\
\text{Relaxed} &\iff \{\text{rlx-fragment}, \text{atomicity}, \\
&\qquad \text{read-coherence-rlx}, \text{WW-coherence-rlx}, \text{RW-coherence-rlx}\} \\
&= \{\text{rlx-fragment}, \mathit{irr}(\text{com}), \mathit{irr}(\text{po}; \text{fr}; \text{rf}^?), \mathit{irr}(\text{po}; \text{mo}), \mathit{irr}(\text{po}; \text{mo}^?; \text{rf})\} \\
\text{Relaxed-Acyclic} &\iff \{\text{rlx-fragment}, \text{porf-acyclicity}, \text{atomicity}, \\
&\qquad \text{read-coherence-rlx}, \text{WW-coherence-rlx}, \text{RW-coherence-rlx}\} \\
&= \{\text{rlx-fragment}, \mathit{irr}(\text{porf}), \mathit{irr}(\text{com}), \\
&\qquad \mathit{irr}(\text{po}; \text{fr}; \text{rf}^?), \mathit{irr}(\text{po}; \text{mo}), \mathit{irr}(\text{po}; \text{mo}^?; \text{rf})\}
\end{aligned}
$$

A.5 Relationships between memory models under single-writer and single-location

Under the single-writer constraint, the fragments of RC20-Ordered are equivalent to the corresponding fragments of RC20, plus SRA. Under the single-location constraint, the fragments of RC20-Ordered are equivalent to the corresponding fragments of RC20 and RC20w, plus SRA.

$$
\begin{aligned}
&\{\text{single-writer}\} \cup \text{RC20-Ordered} \\
\iff &\{\text{single-writer}\} \cup \text{RC20} \\
&\{\text{single-writer}\} \cup \text{RC20-Ordered} \cup \{\text{ra-fragment}\} \\
\iff &\{\text{single-writer}\} \cup \text{RA} \\
\iff &\{\text{single-writer}\} \cup \text{SRA} \\
&\{\text{single-writer}\} \cup \text{RC20-Ordered} \cup \{\text{rlx-fragment}\} \\
\iff &\{\text{single-writer}\} \cup \text{Relaxed-Acyclic} \\
&\{\text{single-location}\} \cup \text{RC20-Ordered} \\
\iff &\{\text{single-location}\} \cup \text{RC20} \\
\iff &\{\text{single-location}\} \cup \text{RC20w} \\
&\{\text{single-location}\} \cup \text{RC20-Ordered} \cup \{\text{ra-fragment}\} \\
\iff &\{\text{single-location}\} \cup \text{RA} \\
\iff &\{\text{single-location}\} \cup \text{SRA} \\
&\{\text{single-location}\} \cup \text{RC20-Ordered} \cup \{\text{rlx-fragment}\} \\
\iff &\{\text{single-location}\} \cup \text{Relaxed-Acyclic} \\
\iff &\{\text{single-location}\} \cup \text{Relaxed}
\end{aligned}
$$

A.6 Comparison between RC20 and the original RC20 axioms

We compare our formalism with the original formalism by Margalit and Lahav [19] for RC20.

$$\begin{aligned}
\text{Margalit-Lahav-RC20} &\triangleq \{\text{RC20-write-coherence}, \text{RC20-read-coherence},\\
&\qquad \text{RC20-atomicity}, \text{RC20-porf-acyclicity}\}\\
\text{RC20-write-coherence} &\triangleq \mathit{irr}(\mathsf{mo}; \mathsf{rf}^?; \mathsf{hb}^?)\\
\text{RC20-read-coherence} &\triangleq \mathit{irr}(\mathsf{fr}; \mathsf{rf}^?; \mathsf{hb})\\
\text{RC20-atomicity} &\triangleq \mathit{irr}(\mathsf{fr}; \mathsf{mo})\\
\text{RC20-porf-acyclicity} &\triangleq \mathit{acy}(\mathsf{po} \cup \mathsf{rf})
\end{aligned}$$

RC20-porf-acyclicity is equivalent to porf-acyclicity, RC20-atomicity is equivalent to backward-atomicity, and RC20-read-coherence is equivalent to read-coherence. We then have

$$\text{RC20-write-coherence} \iff \text{forward-atomicity} \wedge \text{write-coherence-under-porf}$$

where the first case corresponds to not taking the hb edge, and the second case corresponds to taking the hb edge.

In total, we have porf-acyclicity directly, atomicity by forward-atomicity and backward-atomicity, and coherence by write-coherence-under-porf, porf-acyclicity, and read-coherence.

Thus, Margalit-Lahav-RC20 and RC20 are equivalent.