#286 Pomsets with Preconditions: A Simple Model of Relaxed Memory

✓ **Email notification**
Select to receive email on updates to reviews and comments.

PC conflicts
None

Accepted

Submission (694kB) 15 May 2020 11:20:54pm EDT · 77670fc0

▶ **Abstract**
Relaxed memory models must simultaneously achieve efficient implementability and thread-compositional reasoning. Is that why they have
[more]

▶ **Authors (blind)**
R. Jagadeesan, A. Jeffrey, J. Riely [details]

Artifact
No

▶ **Topics**

	OveMer	RevExp
Review #286A	B	X
Review #286B	C	Z
Review #286C	B	X
Review #286D	A	X

6 Comments: *Response (J. Riely)*; anonymous (2), Reviewer C; *Round-2-Submission Response (J. Riely)*; Reviewer A

You are an **author** of this submission.

Edit submission

Reviews and comments in plain text

Review #286A

Overall merit   **Reviewer expertise**

B. Weak Accept **X.** Expert

Paper summary

The paper presents an arguably simpler memory model formalization for programming languages. This formalization relies on a single ordering relation, instead of the collection of distinct orderings usually used. This works only because the model assumes "multi-copy atomicity", i.e. that writes become visible to other threads in a single global order, something that is not implementable on architectures such as ARMv7 or Power. This assumption allows all weakness in the memory ordering to be understood in terms of reordering actions within a thread.

The paper provides useful insights into OOTA behavior by providing a temporal-logic-based approach to prove the absence of certain OOTA behaviors.

Assessment

Strengths: Arguably a simpler memory model formulation. Maybe. Many worked-out litmus cases. Litmus cases seem to be resolved in a reasonable way.

Very general discussion of applicability of sequential transformations that I hadn't seen before.

The paper contains lots of careful comparisons to other models, and thus does a good job of of contrasting this work to other work, and even of indirectly contrasting the other work. Given the general confusion in this area, I think that's a valuable contribution.

Weaknesses: The fact that this requires multi-copy atomicity probably makes this a non-starter for most programming languages that aim to be portable, especially lower-level languages like C. In addition to ARMv7 and Power, I believe this also precludes Nvidia (and possibly other?) GPUs.

I found the paper very dense, and hard to follow for someone not already familiar with the formalism, casting some doubt on the claim that this is not "complicated", as implied by the introduction.

The emphasis on a different formalism, as opposed to a clear description of a change to an existing model, makes it hard to identify the crucial insight here, and would make something like this hard to adopt in an existing language standard.

The lack of machine-checked proofs always casts some doubt in this area.

I couldn't resolve some questions marked with (*) below. I'm hoping for answers in the author response.

Comments for author

Line 58: The explanation around line 58 seems incomprehensible, and I couldn't quickly understand the Lochbihler version either. The fact that s is meant to be local is not explained yet. No object of type C is ever assigned to a global. If correct, this seems to be an artifact of the way allocation is modelled in the Lochbihler paper?

C and C++ quite intentionally do not allow reordering of relaxed atomic loads from the same

C and C++ quite intentionally do not allow reordering of relaxed atomic loads from the same variable, since that we found to be needlessly error prone, e.g. atomically incremented counters can appear to decrease in value. The fact that this invalidates (some) CSE was not viewed as a significant cost, given the relative infrequency of relaxed accesses, and the fact that some forms of CSE, like moving loads out of loops, have dubious practical implications. I think that's the right decision for variables explicitly labelled atomic, as in C, but not for plain accesses in Java.

I don't understand examples like Internal2 at line 520, that have unmatched ra accesses, in this case the store to y. In my mental model, unmatched release acquire accesses should be equivalent to relaxed accesses. If they are not, I worry that implementing them with locks is no longer allowed. That implementation flexibility is essential for C and C++ that support arbitrary-sized atomics. (*) Is the ra here significant? (*) Is a lock-based implementation of atomics sound? Is there an ra access missing at line 532?

The paper addresses thread-inlining. (*) What about other non-thread-local compiler analyses like value-range analysis? If a variable is only ever assigned 1 or 2, is the compiler free to use that fact? I'm actually less concerned about value-range analysis (which seems rare) than I am about alias analysis (which seems common). But I strongly expect the answer to be the same.

This might be clearer as a longer, more self-contained journal paper. On the other hand, I think it would benefit from the conference exposure. And I think an audience-accessible talk is possible, probably by replacing the formalism entirely with examples.

Please informally describe Definition 7.1 before giving the formal definition. I couldn't make sense out of it. (*) If you can give give additional insight here without much added text, please include that in the rebuttal.

Details:

The claim that this "supports all reasonable sequential compiler optimizations" seems prone to misinterpretation. Section 7 correctly points out several kinds of sequential optimizations that are not correct in a concurrent setting.

The statement about relaxed accesses not requiring hardware synchronization could use more explanation. Relaxed accesses currently require special instructions on Itanium. They clearly require special instructions for accesses larger than natively supported by the hardware.

At line 51, it hasn't yet been explained that s is not shared.

Line 72: I don't believe that "undefined behavior" here is actually a weakness. This seems like just an instance of the observation that every substantial program has bugs. I also no longer believe that Java is really substantively different here. It does try to give defined semantics to language-level races. But it does not for racing standard (or other) library calls. For "substantial programs", the latter kind of races are not going to be that much less frequent than the former. Furthermore, the fact that low level races have defined behavior makes it harder to justify simple race detectors, which would usually also detect the latter. Java's security goals for untrusted code forced its approach, but those goals have generally not been met anyway.

I found the definition of substitution confusing. Please add an example or two

Line 132: Calling memory orders "fencing actions" seems weird to me. Fences order two large groups of accesses with respect to each other. These don't.

Line 153 is questionable. Whether plain and relaxed accesses are the same for race-free code depends on OOTA treatment, which isn't yet resolved. Consider if (x) y := 1 || if (y) x := 1 .

Line 527: "has to a jump through"

Line 745: "is preceded by a reading 1 for z": What's "a"?

Definition 7.1: "When it possible to"

Line 992: "??"

Comments for author after the author response

Re: "bait-and-switch". I'm somewhat familiar with the issue, but even to someone reasonably familiar with the Java model, it's still far from obvious how seemingly dead code can affect the outcome here. Please explain in the next version, or possibly use a simpler example.

I remain a bit concerned about the unmatched ra accesses in the examples. It was a goal of the C++ model that, with enough information to prove that, the compiler should be able to replace those with relaxed accesses. I'm not sure that was ever proven or disproven.

In general, this is great discussion that it would be good to see in the paper.

See https://dl.acm.org/doi/pdf/10.1145/3297858.3304043 for more discussion of Nvidia's memory model. It states among many other things: "The PTX model is not multi-copy atomic."

Review #286B

Overall merit	Reviewer expertise
C. Weak Reject	**Z.** Outsider

Paper summary

This paper proposes a new mathmatical model for relaxed memory, based on preconditions and pomsets. The paper gives a semantics to a core imperative language with concurrency, where programs denote sets of pomsets, and shows how these semantics validate (or not) several known subtle examples and litmus tests from the relaxed memory literature.

Assessment

- Pros:
  - Tackles an interesting and important problem

  - Aims to give a relaxed memory model based on well-understood concepts
- Cons:
  - Presentation is extremely convoluted, some parts appear circular
  - Very difficult to understand what the paper is trying to show
  - Seems to require deep knowledge of modern memory models in order to understand even a little bit of the contributions

Comments for author

General comments

I was quite excited by the abstract and introduction of the paper---defining a simple memory model that (a) matches observable behavior of real architectures, (b) validates common compiler optimizations, (c) supports logical reasoning, and (d) uses clean mathematical concepts like pomsets and preconditions would be an amazing achievement. However, the technical presentation of the paper was so convoluted that I could understand very little of the paper. Part of this is probably because I am not an expert in relaxed memory models (though I don't think I have less knowledge than the average member of the OOPSLA community), but the main problem is that the presentation is just very unclear. Piles of definitions are introduced and then seemingly forgotten until many pages later; some terms are not defined at all; example diagrams don't seem to correspond to what is being described in the text; and there is very little description of what the technical developments in each section are trying to accomplish. At points, I was so confused that I wondered if the results were circular. The goal and claims of this paper are impressive, but the presentation needs a lot of work.

Detailed comments

- 50 is this valid or not under Java 1.1?
- 53 "resulting" what did the JMM result from?
- 57 I could not understand this example. why is this split over two lines? and what exactly is the problem here, is this saying that x, y, d can (all?) refer to something of type C, even though this object is assigned to a different variable s?
- 61 I don't understand what the "lack of composability" is here
- 99 what does it mean to "compile" a formal model to an architecture?
- 191 "fulfilled" and "satisfied" are only defined several pages later.
- 199 what does validity of Hoare triples mean here? validity depends on the semantics of programs, but we are proposing a new pomset semantics. how are Hoare triples valid with respect to this pomset semantics?
- 205 does this mean that `r = 2` is true forever along this execution?
- 219 "unsatisfiable" and "termination": when are these terms used?
- 230 "the semantics is also closed with respect to ..." but the semantics hasn't even been defined yet.
- 252 "used define"
- 266 I really liked the diagrams. it would be worth clarifying that the depicted pomset is one element of the semantics of the given program (and that this semantics is still undefined at this point)

- 283 "top-level" where do we use this term? if only much later, it would help to define it right before we need it.
- 289 I almost missed this transition. please visually mark when you are defining semantics for each case
- 291 are these to be understood as running in parallel?
- 310 again, mark new cases in the definition. these are really hard to find at a glance, and it is very easy to get lost in 2.4 (which is essentially one enormous definition)
- 325 I found the implication notation to be very confusing
- 331 what is "eco"?
- 335 what happened to the write to `r`? also, why is there an arrow from `Rx0` to `Wx1`? brown arrow should mean "coherence", but the text says "fulfillment". neither coherence nor fulfillment seem to make sense here. or is the text referring to the green arrow (but then shouldn't that be a "write to read order", not a "read to write order"?)
- 360 "The semantics is again driven by Hoare logic" I could not understand this. the semantics of Hoare logic should be defined in terms of the program semantics, but we are currently defining the program semantics. Unless the paper is referring to an existing semantics of Hoare logic? (this should be clearly cited or better, displayed in the paper).
- 393 "It is amazing how much this definition ..." hyperbole aside, have we finished the definition yet? it appears not.
- 413 why are there suddenly fence dependency arrows, when the paper only introduces fence dependencies in the next section? same with the rest of the diagrams in this section.
- 436 again, I do not understand this "justified by Hoare logic" comment.
- 568 "control dependencies" what color are these arrows?
- 571 what happened to the preconditions, are they all trivial even for this example with an if-then-else?
- 677 "a logic that rules out" shouldn't the model rule something out, not the logic?
- 690 what is the semantics of implication?
- 704-706 are these definitions or theorems? and if theorems, are there proofs?
- 709 is this proof rule claimed to be sound? what does "independent" mean?
- 729 satisfies $\neg W x 1$ what is the significance of this fact, in words? does it mean that no execution can write 1 to x?
- 771 "Preconditions provide a natural solution to working out these dependencies." perhaps so, but almost all of the examples don't have preconditions?
- 773 I could not understand this section at all. There is a theorem statement and proof in the appendix, but many of the terms are not defined or only referred to in other papers.
- 779 "eco" this is not discussed after Def. 2.5
- 797 I could also not follow this section. A high-level question: what are we trying to show in this section? What is the goal? This section introduces a ton of definitions, states a theorem and sketches a proof, but I don't understand what the goal is.
- 825 why is this theorem interesting? what is the intuitive meaning?
- 876 "roach motel laws" citation?
- 993 broken reference

Review #286C

Overall merit
B. Weak Accept

Reviewer expertise
X. Expert

Paper summary
This paper presents a new formalism for relaxed memory models that aims to balance implementation flexibility and ease of use. The foundation is partially ordered multisets, augmented with preconditions. While limited to multi-copy atomic models, the result is a formalism that requires fewer ordering relations than previous examples, and so is in principle easier to reason about. It also admits compositional reasoning, which enables simple proofs of out-of-thin-air properties and SC-DRF.

Assessment
Strengths:

- The model is indeed simpler than previous outings
- The simplicity of the OOTA argument is quite appealing

Weaknesses:

- While simpler than other work, the model is still frustratingly complex, so it's not clear how much we gain
- Not much in the way of applications

Comments for author
On the whole, I like this paper (despite some attempts to sabotage itself, which I'll get to). I'm not exactly confident that it will have any real impact -- it feels to me like another in the long line of memory consistency papers that have follow ups by the same authors a year later showing how the model was wrong. But it is creative, and not obviously incorrect, so I'm inclined to accept it.

I'm most interested in the out-of-thin-air results. Attacking this problem with temporal logic isn't an approach I've seen before, and the simplicity with which the proofs fall out endear me to this model.

At first blush, this model looks a lot like [Alglave 2010] but extended slightly with release/acquire operations instead of fences. It's true that this model somewhat realizes the simplicity of only needing to reason about a single execution relation. But this paper feels a little arbitrary in how it chooses to extend that idea: it focuses extensively on preconditions, but not on loops; it focuses on common subexpression elimination, but not on other code motion.

The other major slight against this paper is that it's incredibly dense, and yet cavalier about it. I can't let the lame excuse at line 1102 for not having any formal verification for this model slide. Mechanization is not that heavy a lift for work like this (Batty et al 2011, for example, is mechanized in ~1200 lines of Lem). Moreover, it helps reviewers and readers understand the model. The memory model literature has become much more approachable (relatively speaking) in the last few years precisely because of mechanization and automation efforts. At this point, it's table stakes.

At line 94, I'm not sure I follow the implication that satisfying this Hoare triple suffices to conclude "the write of y is independent of any code that precedes it in program order". The triple also holds for C_4 = `{true} x=1; y=x {y=1}`, but the write of `y` is certainly not independent of the code before it.

At line 278, I don't understand what this relaxed version of coherence buys us compared to the usual total-order notion. Page 10 doesn't really illuminate the situation.

I struggled in some places with typesetting. Line 282 is a good example. Eventually I figured out the intention is that this is meant to be read as if the previous line ends with "Then we define:", but for a long while these just seemed like dangling equations. Perhaps this just reflects badly on me, though. (It would be much clearer in a mechanized model! :-))

Section 2.5 seems out of place. It feels like this is the place where the architecture-specific nature of the model should appear. But the first thing we do is rule out a sequentially consistent model. One of the nice things about [Alglave 2010] that this paper lacks is clarity about what needs to be "plugged in" to vary the model. Here, I guess the answer is that we vary what we include in $\leq$, but the paper only shows this (very briefly) for ARMv8, and never for TSO.

Minor stuff:

- Line 331: eco is never defined (same problem at line 789)
- Line 356: "old the"
- Line 609: ", require that"

Response

Author [James Riely] 15 Jul 2020

Thanks for reading our paper and providing such valuable comments.

A common theme in the reviews is the density of the presentation. We appreciate the detail and specificity of your comments regarding presentation. Although we only respond to content-based comments below, in revision we will also address the presentational comments---these changes will improve the accuracy and accessibility of the paper. In particular, we will look to see if section 2 can be structured differently to improve readability.

Referee A:

> The fact that this requires multi-copy atomicity probably makes this a non-starter for most programming languages that aim to be portable, especially lower-level languages

> like C. In addition to ARMv7 and Power, I believe this also precludes Nvidia (and possibly other?) GPUs.

We are primarily targeting languages that have type-safety as a goal.

Our knowledge of GPU models is limited to [Alglave, et al, ASPLOS 2015]. The Nvidia GPUs discussed there have a memory model similar to RMO, which is a "global-time"/MCA model. There is some hope that our model might compile efficiently to some of these GPUs. We leave the full exploration of this to future work.

It is worth noting that the Nvidia GPUs don't observe load-load coherence, as witnessed by the CoRR litmus test in [Alglave, et al, ASPLOS 2015]. Thus the model of coherence in the C11 model is too strong to compile efficiently to Nvidia.

On lines 1074-1081, we mention extension to non-MCA architectures as future work. There are tradeoffs here, since non-MCA models are necessarily more complex.

> Line 58: The explanation around line 58 seems incomprehensible, and I couldn't quickly understand the Lochbihler version either. The fact that s is meant to be local is not explained yet. No object of type C is ever assigned to a global. If correct, this seems to be an artifact of the way allocation is modelled in the Lochbihler paper?

Pugh calls this kind of example "bait-and-switch". It is an artifact of the committing semantics of the JMM, not the modeling of memory allocation. Lochbihler assumes only that the memory allocator returns an address. In this example, the address could be the same in each branch of the conditional, leading to the surprising outcome.

In "5.4. Out-of-Thin-Air Values and the Java Security Architecture", Lochbihler discusses a variant, where both branches allocate an array of characters, which one branch means to be immutable. He concludes that the "example shows that the [JMM's] out-of-thin-air guarantee is too weak to support the [Java] security architecture."

"bait-and-switch" can be specified as a temporal property. Section 4 is a first attempt to reasoning about such properties formally.

> C and C++ quite intentionally do not allow reordering of relaxed atomic loads from the same variable, since that we found to be needlessly error prone, e.g. atomically incremented counters can appear to decrease in value. The fact that this invalidates (some) CSE was not viewed as a significant cost, given the relative infrequency of relaxed accesses, and the fact that some forms of CSE, like moving loads out of loops, have dubious practical implications. I think that's the right decision for variables explicitly labelled atomic, as in C, but not for plain accesses in Java.

It is simple to modify our model to include load-load coherence in pomset order. In this case, however, RR (line 989) requires the additional assumption that $x \neq y$, and thus CSE requires alias analysis. We would also lose the potential to compile efficiently to some Nvidia GPUs (see discussion above).

> I don't understand examples like Internal2 at line 520, that have unmatched ra accesses, in this case the store to y. In my mental model, unmatched release acquire accesses should be equivalent to relaxed accesses. If they are not, I worry that implementing them with locks is no longer allowed. That implementation flexibility is essential for C and C++ that support arbitrary-sized atomics. (*) Is the ra here significant? (*) Is a lock-based implementation of atomics sound? Is there an ra access missing at line 532?

Without the RA annotation, the read on `x` would be unordered w.r.t. the operations on `y` in thread 1. Therefore the execution would clearly be allowed.

Because of the RA annotation, however, the read on `x` must be ordered before the operations on `y`. If order was subsequently enforced from `Ry1` to `Wz1`, the execution would be *disallowed* .

Although our model forces some ordering within the thread that includes the RA access, threads that lack RA access are free to reorder. Perhaps this gives the desired effect. For example

$$(y := 1; x^{ra} := 1) || (s := x; z := 1)$$

allows the execution $(Wz1) \leq (Wy1) \leq (W^{ra}x1) \leq (Rx1)$.
(Here, we are using augmentation to add order from $Wz1$ to $Wy1$.)

We do expect that lock-based implementation of atomics should be sound, if locks were introduced as first class entities, but we have not explored this. Intuitively, lock elision can be validated by using roach motel (lines 998-999) to move the entire program into the scope of the lock.

Back to line 532, with the RA annotations as written. Armv8 *allows* this execution, so it is important that we allow it. Armv8 treats "internal reads" as a special case, dropping the order from `Wy1` to `Ry1`. We initially assumed a special case in the read rule would be required in our semantics as well. We were delighted to discover that no special case was necessary.

This is one subtle way that we believe our work simplifies prior approaches. We hope that rules 5a-5d (lines 398-401) are understandable as general principles, rather than simply as the intersection of a series of relations. It is true that there is the cost of understanding the role of substitution at lines 403-404, and in particular that the read rule performs `[x/r]` rather than `[v/r]`. But it appears that no "simplification" in the world of relaxed memory models comes for free.

> The paper addresses thread-inlining. (*) What about other non-thread-local compiler analyses like value-range analysis? If a variable is only ever assigned 1 or 2, is the compiler free to use that fact? I'm actually less concerned about value-range analysis (which seems rare) than I am about alias analysis (which seems common). But I strongly expect the answer to be the same.

Yes! This is the importance of TC1/TC9 (lines 481-498). Note that TC9 also allows collusion by the scheduler! Thus, invariant-based analyses (such as types) would fail for TC9, but our semantics handles TC9 correctly.

In section 3, we model memory locations as values. Thus our model should permit optimizations based on alias analysis.

> Please informally describe Definition 7.1 before giving the formal definition. I couldn't make sense out of it. (*) If you can give give additional insight here without much added text, please include that in the rebuttal.

We will move lines 915-919 above the definition and reword. The idea is to capture postconditions as writes (E_Y). Some preconditions are captured as reads (D_X) and others remain preconditions on the write events (χ). The pomset may also include an arbitrary collection of conflicting writes (C_Y), which are ordered before the "final" writes of E_Y.

Referee B:

> - 50 is this valid or not under Java 1.1?

No, it is not valid.

> - 57 ... what exactly is the problem here, is this saying that x, y, d can (all?) refer to something of type C, even though this object is assigned to a different variable s?

Yes. Worse, they can be assigned C even if the guard on the statement that allocates the C is false.

> - 199 what does validity of Hoare triples mean here? validity depends on the semantics of programs, but we are proposing a new pomset semantics. how are Hoare triples valid with respect to this pomset semantics?

We will clarify that we mean validity w.r.t. the standard sequential semantics.

> - 205 does this mean that `r = 2` is true forever along this execution?

Yes, the precondition `r = 2` will be imposed on every event that follows these events in pomset order.

> - 219 "unsatisfiable" and "termination": when are these terms used?

We tried to separate concerns about the data model from the pomset model and our particular language. This does mean that use of terms is spread out. So that "termination" is defined as an action on line 135, lifted to events on 219, and then used to define the terminal event in our language on 312-318. Likewise "satisfiable" appears at 142, 200, 213, and in section 7.

> - 291 are these to be understood as running in parallel?

These are separate threads. We consider their parallel composition in the following paragraph.

> - 325 I found the implication notation to be very confusing

We will note that this is meant to evoke prefixing in process algebra.

> - 335 what happened to the write to `r`?

It is the Ry0 action. In reads from memory, we do not include the register name.

> also, why is there an arrow from `Rx0` to `Wx1`? brown arrow should mean "coherence", but the text says "fulfillment". neither coherence nor fulfillment seem to make sense here. or is the text referring to the green arrow (but then shouldn't that be a "write to read order", not a "read to write order"?)

We will expand the text here. Since `Rx0` reads `Wx0`, condition 4 of fulfillment (line 257) requires that all writes to `x` are either ordered before `Wx0` or after `Rx0`. Either way, you get a cycle.

> - 393 "It is amazing how much this definition ..." hyperbole aside, have we finished the definition yet? it appears not.

Yes, we should say "this candidate". It is amazing, though! Our initial definition had special cases to handle the examples on lines 477-533 --- every other semantics does! We were shocked to discover that all this special casing could simply be dropped.

> - 413 why are there suddenly fence dependency arrows, when the paper only introduces fence dependencies in the next section? same with the rest of the diagrams in this section.

Yes, we need a better term for "fence or synchronization". We chose the term because synchronization actions have the effect of fences in the implementation. We would very much welcome a suggestion for a better term.

> - 571 what happened to the preconditions, are they all trivial even for this example with an if-then-else?

Yes, they have all been satisfied using (4b) from line 347.

> - 677 "a logic that rules out" shouldn't the model rule something out, not the logic?

We mean "the logic is able to establish that OOTA3 is not possible but it is not strong enough to establish that OOTA1 is not possible". We believe that our Jeffrey/Riely semantics does forbid OOTA1, but we didn't provide a way to prove it.

> - 690 what is the semantics of implication?

This goes back to the data model on line 142. In the examples given in this section, all preconditions are "true", so we did not need to put any particular requirements on the logic.

> - 704-706 are these definitions or theorems? and if theorems, are there proofs?

Theorems. We have pen and paper proofs that we can include in an appendix.

> - 709 is this proof rule claimed to be sound? what does "independent" mean?

Yes, again we have a pen and paper proof. "Independent" again from the data model on line 139. (Sorry!)

> - 729 satisfies $\neg W x 1$ what is the significance of this fact, in words? does it mean that no execution can write 1 to x?

Yes.

> - 771 "Preconditions provide a natural solution to working out these dependencies." perhaps so, but almost all of the examples don't have preconditions?

We might better have said "independencies" in the case of RFUB.

> - 773 I could not understand this section at all.
> - 797 I could also not follow this section.

Indeed, sections 5 and 6 are intended only for specialists. We tried to keep them short! There is a laundry list of things that any memory model must satisfy and these two sections tick off two boxes. In a journal version, we can expand these so that the paper is more self contained.

Referee C:

> At first blush, this model looks a lot like [Alglave 2010] but extended slightly with release/acquire operations instead of fences. It's true that this model somewhat realizes the simplicity of only needing to reason about a single execution relation.

Yes, that is exactly what we have tried to do. Alglave starts with thirteen relations. We were curious to see what could be achieved with only one. We were surprised how far we got.

> But this paper feels a little arbitrary in how it chooses to extend that idea: it focuses extensively on preconditions, but not on loops; it focuses on common subexpression elimination, but not on other code motion.

It is true that we do not handle loops. But we've attempted to handle everything else at least at the state of the art. With section 3, we hope to have covered all of the concurrency primitives of C++. With section 7, we attempted a general account of code motion.

> The other major slight against this paper is that it's incredibly dense, and yet cavalier about it. I can't let the lame excuse at line 1102 for not having any formal verification for this model slide. Mechanization is not that heavy a lift for work like this (Batty et al 2011, for example, is mechanized in ~1200 lines of Lem). Moreover, it helps reviewers and readers understand the model. The memory model literature has become much more approachable (relatively speaking) in the last few years precisely because of mechanization and automation efforts. At this point, it's table stakes.

We will remove the lame excuse from the final revision, noting mechanization as future work.

We hope that the originality and scope of the work are sufficient to overcome this shortcoming. Note that most of the models we discuss in section 1 have been mechanically verified, either as part of the original work, or post-hoc (as Lochbihler did for the JMM). Nonetheless, each of these models either admits a form of thin-air behavior (as in OOTA1) or disallows basic reorderings.

> At line 94, I'm not sure I follow the implication that satisfying this Hoare triple suffices to conclude "the write of y is independent of any code that precedes it in program order". The triple also holds for C_4 = `{true} x=1; y=x {y=1}`, but the write of `y` is certainly not independent of the code before it.

We meant: "the write of y is independent of any code that precedes C_i in program order".

> At line 278, I don't understand what this relaxed version of coherence buys us compared to the usual total-order notion. Page 10 doesn't really illuminate the situation.

We do not know of a useful property or example that distinguishes models that totally order all writes from models that only totally order writes that are read from. Perhaps there is a GPU where it matters? We did not know that load-load coherence would prevent efficient implementation on GPUs until after the paper was submitted.

Our approach has been to add only things that arise naturally from the model. Fulfillment needs to be defined as it is*. And a consequence of that definition is that conflicting writes that are read from must be totally ordered. It is straightforward to impose the additional requirement that *all* conflicting writes are totally ordered. But it is not clear what is gained.

*In appendix B of [Disselkoen, et al 2019], we showed that item 4 (line 258) is required to ensure that the fulfillment required by location binding (lines 279-282) is stable with respect to parallel composition.

> Section 2.5 seems out of place. It feels like this is the place where the architecture-specific nature of the model should appear. But the first thing we do is rule out a sequentially consistent model. One of the nice things about [Alglave 2010] that this paper lacks is clarity about what needs to be "plugged in" to vary the model. Here, I guess the answer is that we vary what we include in $\leq$, but the paper only shows this (very briefly) for ARMv8, and never for TSO.

We will add discussion of how the hardware models (such as TSO, PSO, RMO, Alpha and ARMv8) weaken order relative to SC, and use this to motivate the relaxations in 2.6.

Because we are oriented toward language-level models, we did not think to include explicit material about how the model can be strengthened to more accurately capture these hardware models. This could be developed, perhaps along the lines of [Demange et al. 2013], who discuss a language level model for TSO.

Review #286D

Overall merit
A. Accept

Reviewer expertise
X. Expert

Paper summary

This paper takes an idea previously used to model information-flow atacks like Spectre -- pomsets with preconditions -- and applies it to the study of weak memory. The pomsets are used to represent executions in a similar way to an axiomatic model. The preconditions, applied to each memory event in an execution, diverge from the axiomatic scheme by capturing why the event has occurred. These preconditions capture dependence on other memory events, and they capture structural information about the program from conditionals. There is a clever set of rules that allow substitution and weakening of preconditions, while maintaining a notion of consistency over preconditions. The authors show that these rules account for thin-air behaviour and global value range analysis.

The semantics provides a good coverage of the usual C++-type primitives, including fences and read-modify-writes, but their scheme does not provide the single location coherence guarantees of C++ and nor does it account for non-multi-copy-atomic (non-MCA) behaviour. This is sufficient for x86 and ARM V8 targets, and they provide proofs of efficient implementability for those. They get substantial value out of these choices: their semantics only needs to keep track of a single ordering relation and it is compositional. This helps them establish a stronger local variant of the DRF-SC result. They provide a temporal logic and establish pomsets with preconditions as a basis for reasoning with Hoare triples, using them to validate rewriting rules. The paper uses examples effectively, and presents a great many of them, recalling in particular the OOTA1 example from previous work on Java, and using it to show that the leading Promising semantics does not support both type safety and memory reclamation.

Assessment

This thin-air solution strikes a very different compromise to existing models. The model takes great advantage of MCA to produce something quite elegant, and the treatment is comprehensive enough to see both the cost of the design, and its benefits in terms of reasoning.

The authors begin their paper by highlighting a problem in the predominant Promising semantics. This is valuable knowledge, but it also creates a space to explore an alternative approach to the thin air problem without resolving every detail, or mechanising everything.

The precondition mechanism that drives the semantics impressed me. It seems like a particularly explicit, natural and abstract way to think about dependencies, and I especially like the way it handles global value range assumptions (page 10) while remaining compositional. On the downside, the write elimination mechanism (Section 2.7) does feel a bit ad hoc, but no moreso than other thin-air solutions.

The paper has more than enough in the way of contributions paired with a terse style that is very clear, but occasionally omits too much (e.g. eco, from the literature, was never defined).

The lack of per-location C++ coherence, together with MCA, rules out the semantics as a fix for the C++ memory model, but that is not the point. This paper teaches us quite a bit about an unexplored part of the design space of weak memory models.

Comments for author

The authors should compare their model to the thin-air solution of "Modular Relaxed Dependencies in Weak Memory Concurrency" from ESOP 2020.

The authors say they deal with all Java causality tests (line 557). Tests 19 and 20 include join, but the language does not seem to provide the facility to join. How are tests 19 and 20 accounted for?

In section 5, does dropping parts of dependency-ordered-before produce a strictly weaker model of ARMv8? I want to be sure this is the proper compilation result.

Minor comments:

An example on page 5 uses pointer dereferencing before it is added to the language.

On page 5 and line 639, the use of the terms "states" and "events" confused me for a while.

The inverted delta subscript for downset was not introduced at the definition of downset.

Line 235: "a disjunct".

Line 252: "used define" --> "used to define"

Line 311: "that imply theta", I think you mean "implied by theta".

Line 317: should "theta'(e) = theta'(e)" be "theta'(e) = theta(e)"?

Line 331: eco not defined or referenced.

Lines 323 and 340: "trivial" and "general" expressions. These sound like keywords but are not defined.

Line 391: the final "5c" should probably be "5d".

@A1 17 Jul 2020

I have emailed review #D to the authors and given them until Tuesday July 21 (AoE) to provide a response via email if they desire to do so.

@A2 21 Jul 2020

The author response to review #D is appended:

> The authors should compare their model to the thin-air solution of "Modular Relaxed Dependencies in Weak Memory Concurrency" from ESOP 2020.

Yes, we will.

> The authors say they deal with all Java causality tests (line 557). Tests 19 and 20 include join, but the language does not seem to provide the facility to join. How are tests 19 and 20 accounted for?

Yes, you are right. These examples are not expressible in the language. We don't foresee problems in being able to address a language with joins, eg by including an asymmetric form of parallel composition (C1 ||- C2), which preserves the local state of the right thread. (We had done this in some earlier edits of the paper, thus the mistake.)

We will also clarify that the "expected result" for TC16 is not the same as it is in Java. We disallow this execution due to the cycle

```
(r:=x; x:=1) || (s:=x; x:=2)

Rx2 -co-> Wx1 -rf-> Rx1 -co-> Wx2 -rf-> Rx2
```

We will include this example with an expanded discussion of coherence around examples Co2 and Co3 on page 10.

We will also emphasize that the semantics DOES allow load buffering:

```
(r:=x; y:=1) || (s:=y; x:=2)

Wy1 -rf-> Ry1     Wx2 -rf-> Rx2
```

> In section 5, does dropping parts of dependency-ordered-before produce a strictly weaker model of ARMv8? I want to be sure this is the proper compilation result.

Yes.

ARMv8 does not include internal reads in the global ordering relation, OB. In our semantics, this flexibility is matched by our treatment of internal reads by substitutions that do not cause dependencies; thus, preventing them from contributing to the pomset order. This permits us to only consider the global ordering of ARMv8 events that are not internal reads.

@A3 Reviewer C 29 Jul 2020

Congratulations on your conditional acceptance! Much of the reviewer feedback centres on improving the presentation to be more accessible both to experts and non-experts. We have no mandatory revisions to make, but we hope the 4 additional pages and the detailed reviewer feedback will help you to improve the final version.

Round-2-Submission Response

Author [James Riely] 15 Sep 2020

We dramatically rewrote and reorganized the paper in response to your feedback. Only the sections on "Thin air", "ARMv8", and "Extensions" are substantially the same. This is how the sections have moved:

```
OLD   NEW
 2    2-4    Model + Properties + Refinements
 3     5     Extensions
 4     6     Thin air
 5     7     ARMv8
 6     8     LDRF
7.3   3.2    Valid/Invalid rewrites
 8     9     Related
 9    10     Limitations
```

We refactored section the old section 2 into three sections

+ 2: a simple semantics
+ 3.1: litmus tests
+ 3.2: rewrites that hold for the simple semantics
+ 4: refinements to handle read/write elimination and case analysis

We eliminated the old 7.1-2 discussing Hoare logic. Unlike the other results in the paper, this result was both fragile and narrow. We decided make room to explain the results that are more robust and broadly applicable.

Other changes:

+ section 1: Intro
    - Added a starter memory model example (*)
+ section 2: Model
    - Organized the discussion around (*)
    - Added substitution example
    - Reorganized the presentation of prefixing
+ section 3.1: Litmus tests
    - corrected the discussion of JMM litmus tests
    - added load buffering
    - added TC16
    - added more discussion of coherence

  - add discussion of MCA1/MAC2
  - moved discussion of Sevcik thesis to the end of section 4

+ section 3.2: Rewrites/Optimizations

  - strengthened RL and CSE to get rid of read saturation
  - split CA to clarify which direction works in the simple model

+ section 4: Refinements to support read/write elimination and case analysis

  - cleanup of write prefixing (more precise than previous use of ||)
  - changed the semantics so that read elimination is now valid without appeal to "read saturation"
  - fixed bug in read-prefixing (See last paragraph of "Case Analysis")
  - strengthened SF to get rid of read saturation
  - added RE and RI

+ section 5: Extensions

  - cleanup address calculation

+ section 6: No-TAR

  - moved (*) to the intro
  - moved RFUB here

+ section 7: ARMv8

  - added one example at beginning to clarify which version of ARMv8 we are targeting

+ section 8: DRF-SC

  - added precise definition and detail in the proof so as to be self contained
  - added two new examples comparing to Dolan, et al: one for past races and one for future races

+ section 9: Related

  - added discussion of [Paviotti et al. 2020]

+ section 10: Limitations

  - removed argumentative bits
  - added a paragraph discussing java final field semantics w.r.t. read-read dependencies

main.pdf (725kB)

@A4 Reviewer A 4 Oct 2020

Some more suggestions for the final paper. Please consider:

1) Please state the assumption up front that all variables are initially zero before you use it in line 56. That would also abbreviate some examples, like at line 141. Currently that assumption seems to be inconsistently made. 2) [minor] I would not claim the statement that “every substantial C program has undefined behavior” is purely traceable to data races. Many other sources of undefined behavior contribute to that. 3) I still don't understand the initial explanation of OOTA1. Going back to the reference, I think the bad execution here arises because "new C" effectively allocates a type C object at a location that we had already previously committed to as containing a type D object. Without some explanation along these lines, this seems really hard to follow. It's crucial to look at the constituent pieces of allocation and not to view it as a single atomic operation. I don't believe that currently comes across.