#220 Predicate Transformers for Relaxed Memory: Sequential Composition for Concurrency Using Semantic Dependencies

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

PC conflicts
None

Submitted

Submission (329kB) 16 Apr 2021 4:00:23pm AoE · 9088d1e0

▶ **Abstract**

Program logics and semantics tell us that when executing $(S_1; S_2)$ starting in state s_0, we execute S_1 in s_0 to arrive at s_1, then execute S_2 in

[more]

▶ **Authors (blind)**

A. Jeffrey, J. Riely [details]

Attendance Preferences (potentially)

Yes

▶ **Topics**

	OveMer	RevCon
Review #220A	2	3
Review #220B	2	3
Review #220C	3	2

1 Comment: *R2 Response* (J. Riely)

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

Edit submission **Edit R2 response**

Reviews and comments in plain text

Review #220A

Overall merit

Review confidence

2. Weak reject **3.** High

Paper summary

In last year's OOPSLA, Jagadeesan et al. presented "Pomsets with Preconditions" (PwP), a simple semantics for concurrent programs that handles various relaxed memory optimisations. In this paper, the authors extend PwP with an associative sequential-composition operator. (The original PwP only has a prefixing operator; it cannot sequentially compose arbitrary programs.) To achieve this, they augment each event in their pomset not merely with a precondition, but with a family of Dijkstra-style predicate transformers. Then the semantics of `P1;P2` is defined so that `P1` predicate transformers are plugged into `P2`'s preconditions; which predicate transformer is selected depends on the *semantic* dependencies from `P1` to `P2`.

The advantage that proper sequential composition has over prefixing is that one can reason about program fragments, à la peephole optimisations. So PwP theorems like

```
[[x := 1; y := 1; S]] = [[y := 1; x := 1; S]]
```

that need to talk about the entire remaining program `S`, can be replaced with the rather neater

```
[[x := 1; y := 1]] = [[y := 1; x := 1]]
```

Strengths and Weaknesses

Strengths

- The paper is very well written and is a pleasure to read. I really enjoyed the conversational style.
- The paper represents an effort to sort out the "mess" of relaxed memory, and I think this is a laudable aim.
- I think the basic idea of putting predicate transformers into the execution graphs is quite neat (though of course there are an awful lot of details required to make that work, cf Figure 1).

Weaknesses

- Being able to reason about program fragments rather than entire program suffixes certainly marks an improvement over Jagadeesan et al.'s OOPSLA 2020 paper, but I'm not sure it's a substantial enough diff to merit an OOPSLA paper.
- The paper feels somewhat unfinished. For instance, the abstract claims that the new model has a local DRF theorem (line 18), but actually this is only conjectured (line 237) based on the new model's similarly to the prior PwP model. If the new model is so similar to the PwP model, then I suspect that it would not be an inordinate amount of work to actually prove properties like the DRF theorem. Also, the authors mention that an earlier version of the paper has been mechanised in Agda but that the mechanisation needs updating; I would feel more comfortable accepting this paper once that process is complete.

Comments for author

The paper's title is quite a mouthful! But I suppose it's sensible to keep all of its components. One thing that might help a little bit would be to put the part after the colon in a `\subtitle{...}`.

77: Might be worth clarifying that || binds less tightly than ;. You might also loosen the kerning around the || operator to indicate this even more clearly.

120: Visually, I wonder if it would look nicer to extend the `\mid` symbol so that it fills the height of the rectangle. This might help the reader to understand that the division between precondition and event should be parsed at the very lowest precedence.

124: You are assuming that `=>` binds less tightly than `/\` but this isn't standard. Parentheses might help to disambiguate (here and elsewhere).

174: I find the "linear" presentation of your pomsets quite unsatisfying. Too much is being packed into the horizontal dimension. What I mean is: in line 164 the horizontal dimension is being used to suggest sequential composition, but then in line 174, the same dimension is being used arbitrarily to spread out the predicate transformers. At a glance, it looks like ψ is sequenced before $Rx0$ and $(0 = r) \Rightarrow \psi$ is sequenced after it, but I don't think this is actually the case -- they could be flipped around without changing anything, right? So, I wonder if the various predicate transformers would be better off pulled out of the sequential flow, perhaps into a separate layer below the program events.

292: I wouldn't bother explaining that $\Rightarrow$ associates to the right -- anybody who is going to get anything out of this paper is going to be well aware of that. By the way: at this point you mention that $r{=}v \Rightarrow \psi$ should be parsed as $(r{=}v) \Rightarrow \psi$, which suggests that you could drop a lot of the parentheses you have put around equality expressions throughout the paper.

306: The definition of "delays" should come after the co/sync/sc relations that it depends on.

311: I find it odd to call $\bowtie_{sc}$ an "order" because it is clearly symmetric. It only really becomes an "order" when it is used in the context of (s7c) on line 361, which further imposes that the pairs of events are separated by sequential composition. Perhaps you can invent a different name for $\bowtie_{sc}$?

328: I'm a bit bothered by the way that the τ symbol is used both to denote a family of predicate transformers and as an abbreviation for the *specific* predicate transformer τ^E. It feels a bit confusing. Actually: do you even use that abbrevation having defined it on line 328? Perhaps you can just drop it, because you introduce it again at the start of Section 2.10.

336: $rf : E \to E$ should be $rf \subseteq E \times E$.

338: $\leq : E \times E$ should be $\leq \subseteq E \times E$.

344: $R_1 : E_1 \times E_1$ should be $R_1 \subseteq E_1 \times E_1$ (similar for R_2).

377: (w1) feels like a strangely convoluted way of saying $|E| \leq 1$.

Regarding Figure 1: one thing that was a little tricky for me was the fact that you tacitly assume that

the components of pomset P are called E, κ, etc. (And that the components of pomset P_1 are called E_1, κ_1, etc.) Other works in this area have adopted the convention of writing things like $P.E$ or E^P for the E component of pomset P. I imagine that doing so might make your semantics too verbose. Nevertheless, you might consider adding a note to explain your convention to the reader.

Lemma 2.5: You might add a note (perhaps in the proof) to remind the reader why they shouldn't expect $skip||P = P$ (i.e. because P's local state would be thrown away).

421: I *think* the reason for mapping reads/writes/fences to pomsets with *at most one* event (rather than *exactly* one event) is so you can validate transformations that remove unnecessary reads or redundant writes. If that's correct, I think it would be worth explaining that at this point of the paper.

499: Can you lift the superscript e a little further, so it's clearly attached to the node rather than the arrow?

595: It took me a while to work out what ! and !! are supposed to mean. Please clarify that they are intended as logical negation (if that's correct).

705: The beginning of Section 3 is really terse -- it reads more like draft notes! Needs an introductory sentence or two, and the Arm model needs a citation to explain where you're getting all these lobs and EGCs from.

1106: The phrase "compositional semantics for sequential composition" is a bit inelegant but I guess there's not too much you can do about it.

Minor things

- 17: s/the prior/prior
- 18: s/data race freedom theorem/data-race-freedom theorem
- 40: s/side-effect free expressions/side-effect-free expressions
- 87: s/which/that
- 200: s/have take/take
- 301: s/least/smallest (feels more grammatical?)
- 313: s/which/that
- 505: s/write/write,
- 523: s/left/right
- 633: s/equivalent/equivalent to
- 702: s/which will/which we will
- 702: s/not case/not the case
- 781: s/therefore/therefore,
- 786: s/not/is not

Questions for authors' response

- You don't explicitly state whether your model is intended to capture C11, Java, both, or neither. It looks pretty similar to C11 but not quite the same. Could you clarify?

- On line 503, I couldn't get my head around why the independent transformer is $(x{=}r \vee 1{=}r) \Rightarrow \psi$. Could you explain the intuition behind that? (I can see that it comes from R4b but I don't see why there is a disjunction.)

Review #220B

Overall merit
2. Weak reject

Review confidence
3. High

Paper summary
The paper extends the "Pomset with Preconditions" model of Jagadeesan et al. with predicate transformers so that it provides a compositional semantics to sequential composition.

Strengths and Weaknesses
The goal of this work (to provide a compositional semantics for shared-memory concurrency with weak consistency that allows efficient implementation) is relevant and important for programming languages, compilers, runtime systems, etc.

Yet, I have mixed feelings about the approach taken.

On the one hand, I appreciate the achievement of defining a compositional concurrency semantics. Defining such a semantics and ensuring that it supports efficient compilation to hardware and the correctness of compiler optimisations is really challenging.

On the other hand, the model is way too complicated, and does not appear particularly useful for verification, not even for the simple task of enumerating all possible outcomes of a litmus test. The various subtleties in the formal definition are almost impossible to understand by a mere mortal, and even by people that have spent years on related topics/models. I suspect that the main problem is the huge space of objects used to define the semantics: for a program with N statements, a single execution in its semantics has not only N logical assertions (one for each action as in the precursor model), but also 2^N further assertions (predicate transformers). Contrast that with the Arm model, which only has the N actions and a (fairly complex) partial order over them.

In part due to the complexity of the model, I cannot reasonably evaluate the correctness of the lemmas stated in the paper. I am willing to trust the author(s), but I think more evidence is necessary, since, e.g., the "Pomset with Preconditions" model failed to validate Redundant Read Elimination (Appendix, Line 1361). Similar problems occur with many other models in the literature. In the conclusion, the authors mention the possibility of a mechanised proof. I would urge the author(s) to carry it out, as it would largely alleviate my concerns regarding correctness.

Comments for author

While I appreciate the effort put into presenting the model in a stepwise fashion, I'm really put off by having 8 different variants/extensions of the model, and most results stated/proved about a different variant of the model. I wonder, is there one model that incorporates all the desired adaptations/extensions, that satisfies all the properties discussed in the paper?

The paper does not really discuss the programmability results, such as DRF theorems or the temporal logic from the "Pomset with Preconditions" paper, while the abstract mentions these results. Is it because of lack of space or are they straightforward?

Review #220C

Overall merit

3. Weak accept

Review confidence

2. Medium

Paper summary

This paper further develops a semantic-dependency account of weak memory.

Strengths and Weaknesses

- \+ further develops a plausible approach to weak memory semantics
- – still a work in progress

Comments for author

This paper extends Jagadeesan et al OOPSLA 2020 in several directions. but, as the paper notes, with many more developments still needed. The rationale for this approach is well explained. One might not agree that this is the most promising approach to weak-memory semantics, but it is surely defensible.

The detailed account is also well-explained, but massive enough that few people will carefully read all of it. This is OK, since some of this is destined to serve as reference material if the approach eventually succeeds. My only hesitance is that this paper, plus possibly follow-ons might be better suited to straight-to-journal submission.

Given the premises and goals, the general forms of the technical development seem almost definitionally sound, modulo errors in modeling/judgements based on observed processor weaknesses and oddities. I worry about the ARM-centric nature of some restrictions and constructs. It would improve confidence to also consider POWER, RISC-V, and/or others as a cross-check on too little vs too much generalization. I am generally familiar with ARM, but a few of the ARM oddities were new to me. I gather that the authors have checked these with ARM architects; if so they should be clearer about this.

I liked the technical ploy of first using single-use registers and then generalizing. As almost-noted in the paper, this mirrors actual register-renaming in processors as well as SSA compilation.

R2 Response

Author [James Riely] 15 Jun 2021 745 words

Thanks for reading our paper! All of the reviews are perfectly sensible. But it does seem like the big picture of the paper has gotten lost a bit in the grungy details of relaxed memory...

The paper includes a *novel* and *transferable* idea: families of predicate transformers can be used to calculate relaxed dependencies in a way that is *compositional* and *direct*.

This idea is not tied to the vagaries of ARM, or any particular compiler optimization, or DRF-SC, etc.

There has only been one previous attempt to compute dependencies in a relaxed memory model compositionally [Paviotti et al. 2020], and that semantics uses continuation passing to encode sequential composition. As an example of transferability: it would be relatively straightforward to adapt our technique to [Paviotti et al. 2020] in order to arrive a semantics of C11 that avoids thin air results. This would swap the event structures of [Paviotti et al. 2020] for preconditions and predicate transformers.

The other methods of computing dependencies are:

- not bothering (as in C11)
- using syntax (as in hardware models)
- using complex non-compositional operational models such as the JMM, promising semantics, weakestMO, etc

REFEREE A:

> You don't explicitly state whether your model is intended to capture C11, Java, both, or neither. It looks pretty similar to C11 but not quite the same. Could you clarify?

You could think of our model as a variant of C11 that allows more optimizations on relaxed atomics. (C11 fails to validate common-subexpression-elimination on relaxed atomics without alias analysis.) Java fails time-wise LDRF (races from the past can pollute the future).

> On line 503, I couldn't get my head around why the independent transformer is $(x{=}r \vee 1{=}r) \Rightarrow \psi$. Could you explain the intuition behind that? (I can see that it comes from R4b but I don't see why there is a disjunction.)

This was discussed in the PwP paper, using non-skolemized substitutions rather than implication (see appendix D of this paper). In this paper, we discussed JMM TC1 briefly at lines 564-569. In that example, the precondition of Wy1 is ((1=r ∨ 0=r) ⇒ r≥0). You need that (1=r ⇒ r≥0) to ensure

that the preceding read consumes an admissible value. You need that (0=r ⇒ r≥0) to ensure that the preceding write produces an admissible value.

REFEREE B:

> On the other hand, the model is way too complicated,

We were quite pleased that Fig 1 fit on a single page. Allocating another page for definitions 2.1-2.4, this still compares quite favorably to any other relaxed model.

> and does not appear particularly useful for verification

Work-in-progress suggests otherwise.

> I wonder, is there one model that incorporates all the desired adaptations/extensions, that satisfies all the properties discussed in the paper?

Yes. Pick the variant from section 3 that gives the compilation strategy you want to ARM. The additional features of section 4 are cumulative, so take the last model there and add RMWs.

> The paper does not really discuss the programmability results, such as DRF theorems or the temporal logic from the "Pomset with Preconditions" paper, while the abstract mentions these results. Is it because of lack of space or are they straightforward?

We shall remove this from the abstract.

The results of section 6 and 8 of PwP need to be rephrased to match the details of this paper. But we expect the proof to be nearly identical. We have not worked through the details yet.

At a high-level, emphasizing these detracts from the main point of the paper.

REFEREE C:

> The detailed account is also well-explained, but massive enough that few people will carefully read all of it.

We hope that the introduction has a kernel of an idea which is applicable beyond the details of the model.

> It would improve confidence to also consider POWER, RISC-V, and/or others as a cross-check on too little vs too much generalization.

Work-in-progress demonstrates that the idea of this paper is indeed transferable to these architectures. These models require more than one order, but no changes to preconditions or predicate transformers. The dependency calculation is the same!

> I gather that the authors have checked these with ARM architects; if so they should be clearer about this.

We are using a (just published!) characterization of ARM [Alglave et al 2021]. The authors of that paper have seen a preprint of section 3.

> My only hesitance is that this paper, plus possibly follow-ons might be better suited to straight-to-journal submission.

But isn't PACM-PL a journal? :-)