Sequential Composition for Relaxed Memory: Pomsets with Predicate Transformers[*]

ALAN JEFFREY, Roblox, USA

JAMES RIELY, DePaul University, USA

This paper presents the first compositional definition of sequential composition that applies to a relaxed memory model weak enough to allow efficient implementation on Arm. We extend the denotational model of pomsets with preconditions with predicate transformers. Previous work has shown that pomsets with preconditions are a model of concurrent composition, and that predicate transformers are a model of sequential composition. This paper show how they can be combined.

CCS Concepts: • **Theory of computation → Parallel computing models**; *Preconditions*.

Additional Key Words and Phrases: Concurrency, Relaxed Memory Models, Multi-Copy Atomicity, ARMv8, Pomsets, Preconditions, Temporal Safety Properties, Thin-Air Reads, Compiler Optimizations

1 INTRODUCTION

This paper is about the interaction of two of the fundamental building blocks of computing: memory and sequential composition. One would like to think that these are well-worn topics, where every issue has been settled, but this is sadly not the case.

1.1 Memory

For single-threaded programs, memory can be thought of as you might expect: programs write to, and read from, memory references. This can be thought of as a total order of reads and writes, where each read has a matching *fulfilling* write, for example:

$$x := 0 \,;\, x := 1 \,;\, y := 2 \,;\, r := y \,;\, s := x$$
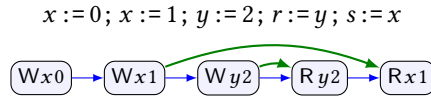
(In examples, r–s range over thread-local registers and x–z range over shared memory references.)

Authors' addresses: Alan Jeffrey, Roblox, Chicago, USA; James Riely, DePaul University, Chicago, USA.

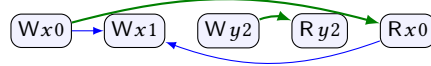This model naturally extends to the case of shared-memory concurrency, leading to a *sequentially consistent* semantics [Lamport 1979], in which *program order* inside a thread implies a total *causal order* between read and write events, for example:

$$x := 0;\ x := 1;\ y := 2 \parallel r := y;\ s := x$$

$$\boxed{\text{W}x0} \rightarrow \boxed{\text{W}x1} \rightarrow \boxed{\text{W}y2} \quad \boxed{\text{R}y2} \rightarrow \boxed{\text{R}x1}$$
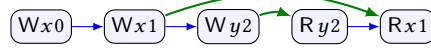
Unfortunately, this model does not compile efficiently to commodity hardware, resulting in a 37–73% increase in CPU time on ARM [Liu et al. 2019] and, hence, in power consumption. Developers of software and compilers have therefore been faced with a difficult trade-off, between an elegant model of memory, and its impact on resource usage (such as size of data centers, electricity bills and carbon footprint). Unsurprisingly, many have chosen to prioritize efficiency over elegance.

This has led to *relaxed memory models*, in which the requirement of sequential consistency is weakened to only apply *per-location* and not globally over the whole program. This allows executions which are inconsistent with program order, such as:

$$x := 0;\ x := 1;\ y := 2 \parallel r := y;\ s := x$$

$$\boxed{\text{W}x0} \rightarrow \boxed{\text{W}x1} \quad \boxed{\text{W}y2} \quad \boxed{\text{R}y2} \quad \boxed{\text{R}x0}$$

In such models, the causal order between events is important, and includes control and data dependencies, to avoid paradoxical "out of thin air" examples such as:

$$r := x;\ \texttt{if}(r)\{y := 1\} \parallel s := y;\ x := s$$

$$\boxed{\text{R}x1} \rightarrow \boxed{\text{W}y1} \quad \boxed{\text{R}y1} \rightarrow \boxed{\text{W}x1}$$
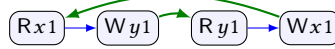
This candidate execution forms a cycle in causal order, so is disallowed, but this depends crucially on the control dependency from ($\text{R}x1$) to ($\text{W}y1$), and the data dependency from ($\text{R}y1$) to ($\text{W}x1$). If either is missing, then this execution is acyclic and hence allowed. For example dropping the control dependency results in:

$$r := x;\ y := 1 \parallel s := y;\ x := s$$

$$\boxed{\text{R}x1} \quad \boxed{\text{W}y1} \quad \boxed{\text{R}y1} \rightarrow \boxed{\text{W}x1}$$
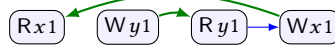
Unfortunately, while a simple syntactic approach to dependency calculation suffices for hardware models, it is not preserved by common compiler optimizations. For example, if we calculate control dependencies syntactically, then there is a dependency from ($\text{R}x1$) to ($\text{W}y1$), and therefore a cycle in, the candidate execution:

$$r := x;\ \texttt{if}(r)\{y := 1\}\,\texttt{else}\,\{y := 1\} \parallel s := y;\ x := s$$

$$\boxed{\text{R}x1} \rightarrow \boxed{\text{W}y1} \quad \boxed{\text{R}y1} \rightarrow \boxed{\text{W}x1}$$

An optimizing compiler might lift the assignment $y := 1$ out of the conditional, thus removing the control dependency.

Prominent solutions to the problem of dependency calculation include:

- *syntactic* methods used in hardware models such as ARM or x86-TSO [Alglave et al. 2014],
- *speculative execution* methods (which give a semantics based on multiple executions of the same program) such as the Java Memory Model [Manson et al. 2005] and related models [Chakraborty and Vafeiadis 2019; Jagadeesan et al. 2010a; Kang et al. 2017],

- *rewriting* methods, which give an operational model up to syntactic rewrites, such as [Pichon-Pharabod and Sewell 2016], and
- *logical* methods, such as the pomsets with preconditions model of [Jagadeesan et al. 2020].

In this paper, we will focus on logical models, as those are compositional, and align well with existing models of sequential composition. The heart of the model of [Jagadeesan et al. 2020] is to add logical preconditions to events, which are introduced by store actions (modeling data dependencies) and conditionals (modeling control dependencies):

$$\texttt{if(s<1)\{z := r*s\}}$$

$$\boxed{(s<1) \wedge (r*s)\texttt{=}0 \mid \mathsf{W}z0}$$

Preconditions are discharged by being ordered after a read:

$$r := x \,;\, s := y \,;\, \texttt{if(s<1)\{z := r*s\}}$$

$$\boxed{\mathsf{R}x0} \qquad \boxed{\mathsf{R}\,y0} \longrightarrow \boxed{(s\texttt{=}0) \Rightarrow (s<1) \wedge (r*s)\texttt{=}0 \mid \mathsf{W}z0}$$

Note that there is dependency order from $(\mathsf{R}\,y0)$ to $(\mathsf{W}z0)$ so the precondition for $(\mathsf{W}z0)$ only has to be satisfied assuming the hypothesis $(s\texttt{=}0)$. There is no matching order from $(\mathsf{R}x0)$ to $(\mathsf{W}z0)$ which is why we do not assume the hypothesis $(r\texttt{=}0)$. Nonetheless, the precondition on $(\mathsf{W}z0)$ is a tautology, and so can be elided in the diagram:

$$\boxed{\mathsf{R}x0} \qquad \boxed{\mathsf{R}\,y0} \longrightarrow \boxed{\mathsf{W}z0}$$

While existing models of relaxed memory have detailed treatments of parallel composition, they often give sequential composition little attention, either ignoring it altogether, or treating it operationally with its usual small-step semantics. This paper investigates how existing models of sequential composition interact with relaxed memory.
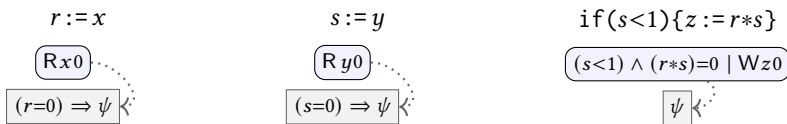
1.2 Sequential composition

Our approach follows that of weakest precondition semantics of Dijkstra [1975], which provides an alternative characterization of Hoare logic [Hoare 1969] by mapping postconditions to preconditions. We recall the definition of $wp_S(\psi)$ for loop-free code below.

- $wp_{\texttt{skip}}(\psi) = \psi$
- $wp_{\texttt{abort}}(\psi) = \mathsf{ff}$
- $wp_{r := M}(\psi) = \psi[M/r]$
- $wp_{S_1;S_2}(\psi) = wp_{S_1}(wp_{S_2}(\psi))$
- $wp_{\texttt{if}(M)\{S_1\}\texttt{else}\{S_2\}}(\psi) =$
  $((M{\neq}0) \Rightarrow wp_{S_1}(\psi)) \wedge ((M{=}0) \Rightarrow wp_{S_2}(\psi))$

The rule we are most interested in is the one for sequential composition, which maps sequential composition of programs to function composition of predicate transformers.

Predicate transformers are a good fit to logical models of dependency calculation, since both are concerned with preconditions, and how they are transformed by sequential composition. Our first attempt is to associate a predicate transformer with each pomset. We visualize this in diagrams by showing how ψ is transformed, for example:

$$r := x \qquad\qquad s := y \qquad\qquad \texttt{if(s<1)\{z := r*s\}}$$

$$\boxed{\mathsf{R}x0} \qquad\qquad \boxed{\mathsf{R}\,y0} \qquad\qquad \boxed{(s<1) \wedge (r*s)\texttt{=}0 \mid \mathsf{W}z0}$$

$$\boxed{(r\texttt{=}0) \Rightarrow \psi} \qquad\quad \boxed{(s\texttt{=}0) \Rightarrow \psi} \qquad\qquad\quad \boxed{\psi}$$

In the rightmost program above, the write to z affects the shared store, not the local state of the thread, therefore we assign it the identity transformer.

For the sequentially consistent semantics, sequential composition is straightforward: we apply each predicate transformer to the preconditions of subsequent events, and compose the predicate transformers:

$$r := x\,;\ s := y\,;\ \texttt{if}(s{<}1)\{z := r{*}s\}$$

$$\boxed{\mathsf{R}\,x\,0} \rightarrow \boxed{\mathsf{R}\,y\,0} \rightarrow \boxed{(r{=}0) \Rightarrow (s{=}0) \Rightarrow (s{<}1) \land (r{*}s){=}0 \mid \mathsf{W}\,z\,0}$$

$$\boxed{(r{=}0) \Rightarrow (s{=}0) \Rightarrow \psi}$$
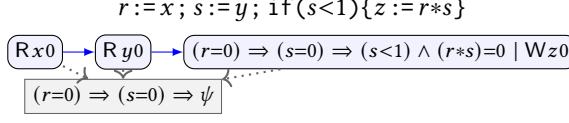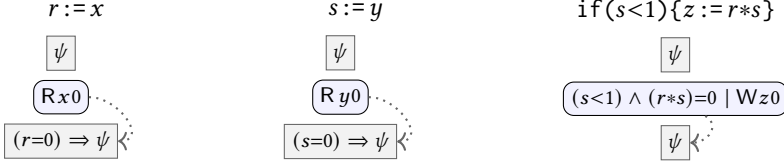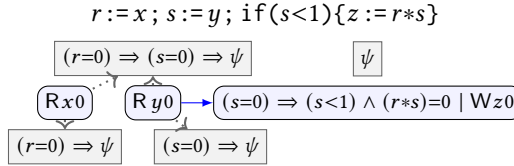
This model works for the sequentially consistent case, but needs to be weakened for the relaxed case. The key observation of this paper is that rather than working with one predicate transformer, we should work with a *family* of predicate transformers, indexed by sets of events.

For example, for single-event pomsets, there are two predicate transformers, since there are two subsets of any one-element set. The *independent* transformer is indexed by the empty set, whereas the *dependent* transformer is indexed by the singleton. We visualize this by including more than one transformed predicate, with an edge leading to the dependent one. For example:

$$r := x \qquad\qquad s := y \qquad\qquad \texttt{if}(s{<}1)\{z := r{*}s\}$$

$$\boxed{\psi} \qquad\qquad \boxed{\psi} \qquad\qquad \boxed{\psi}$$

$$\boxed{\mathsf{R}\,x\,0} \qquad\qquad \boxed{\mathsf{R}\,y\,0} \qquad\qquad \boxed{(s{<}1) \land (r{*}s){=}0 \mid \mathsf{W}\,z\,0}$$

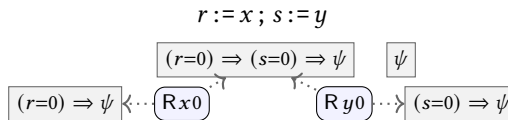$$\boxed{(r{=}0) \Rightarrow \psi} \qquad \boxed{(s{=}0) \Rightarrow \psi} \qquad \boxed{\psi}$$

The model of sequential composition then picks which predicate transformer to apply to an event's precondition by picking the one indexed by all the events before it in causal order.
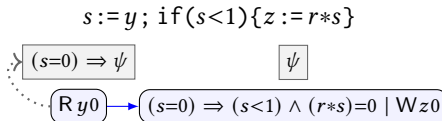
For example, we can recover the expected semantics for the above example by choosing the predicate transformer which is independent of $(\mathsf{R}\,x\,0)$ but dependent on $(\mathsf{R}\,y\,0)$, which is the transformer which maps ψ to $(s{=}0) \Rightarrow \psi$.

$$r := x\,;\ s := y\,;\ \texttt{if}(s{<}1)\{z := r{*}s\}$$

$$\boxed{(r{=}0) \Rightarrow (s{=}0) \Rightarrow \psi} \qquad\qquad \boxed{\psi}$$

$$\boxed{\mathsf{R}\,x\,0} \quad \boxed{\mathsf{R}\,y\,0} \rightarrow \boxed{(s{=}0) \Rightarrow (s{<}1) \land (r{*}s){=}0 \mid \mathsf{W}\,z\,0}$$

$$\boxed{(r{=}0) \Rightarrow \psi} \qquad \boxed{(s{=}0) \Rightarrow \psi}$$

As a sanity check, we can see that sequential composition is associative in this case, since it does not matter whether we associate to the left, with intermediate step:

$$r := x\,;\ s := y$$

$$\boxed{(r{=}0) \Rightarrow (s{=}0) \Rightarrow \psi} \quad \boxed{\psi}$$

$$\boxed{(r{=}0) \Rightarrow \psi} \quad \boxed{\mathsf{R}\,x\,0} \qquad\qquad \boxed{\mathsf{R}\,y\,0} \cdots \boxed{(s{=}0) \Rightarrow \psi}$$

or to the right, with intermediate step:

$$s := y\,;\ \texttt{if}(s{<}1)\{z := r{*}s\}$$

$$\boxed{(s{=}0) \Rightarrow \psi} \qquad\qquad \boxed{\psi}$$

$$\boxed{\mathsf{R}\,y\,0} \rightarrow \boxed{(s{=}0) \Rightarrow (s{<}1) \land (r{*}s){=}0 \mid \mathsf{W}\,z\,0}$$

This is an instance of a general result that sequential composition forms a monoid, as one would hope.

1.3 Contributions

In this paper, we show how pomsets with preconditions [Jagadeesan et al. 2020] can be combined with predicate transformers [Dijkstra 1975] to create a compositional semantics for sequential composition.

- §3 presents the basic model, with few features required of the logic of preconditions, but a resulting lack of fidelity to exiting models,
- §4 adds a model of *quiescence* to the logic, required to model coherence (accessing x has a precondition that x is quiescent) and synchronization (a releasing write requires all locations to be quiescent),
- §5 adds the features required for efficient compilation to modern architectures: downgrading some synchronized accesses to relaxed, and removing read-read dependencies, and
- §6 show how to address common litmus tests.

The definitions in this paper have been formalized in Agda.

Because it is closely related, we expect that the memory-model results of [Jagadeesan et al. 2020] apply to our model, including compositional reasoning for temporal safety properties and local DRF-SC as in [Dolan et al. 2018; Dongol et al. 2019]. In §5, we provide an alternative proof strategy for efficient compilation to Arm8, which improves upon that of [Jagadeesan et al. 2020] by using a recent alternative characterization of Arm8.

2 RELATED WORK

- Models with strong dependencies that disallow compiler optimizations and efficient implementation on Arm8: [Boehm and Demsky 2014; Dolan et al. 2018; Jeffrey and Riely 2016; Kavanagh and Brookes 2018; Lahav et al. 2017,?; Lamport 1979].
- Models with weak dependencies that allow some forms of thin-air reasoning: [Chakraborty and Vafeiadis 2019; Jagadeesan et al. 2010b; Kang et al. 2017; Manson et al. 2005]
- Models that do not calculate dependencies: [Batty et al. 2011; Cox 2016; Watt et al. 2020, 2019].

Like us, Kavanagh and Brookes [2018] use pomsets to create a compositional account of sequential composition. However, their model requires a fence after every read on Arm8, and uses syntactic dependencies, thus invalidating many compiler optimizations.

3 MODEL

In this section, we present the mathematical preliminaries for the model (which can be skipped on first reading). We then present the model incrementally, starting with a model built using *partially ordered multisets* (*pomsets*) [Gischer 1988; Plotkin and Pratt 1996], and then adding preconditions and finally predicate transformers.

In later sections, we will discuss extensions to the logic, and to the semantics of load, store and thread initialization, in order to model relaxed memory more faithfully. We stress that these features do *not* change any of the structures of the language: conditionals, parallel composition, and sequential composition are as defined in this section.

3.1 Preliminaries

The syntax is built from

- a set of *values* $\mathcal{V}$, ranged over by v, w, ℓ, k,
- a set of *registers* $\mathcal{R}$, ranged over by r, s,
- a set of *expressions* $\mathcal{M}$, ranged over by M, N, L.

Memory references are tagged values, written $[\ell]$. Let X be the set of memory references, ranged over by x, y, z.

We require that

- values and registers are disjoint,
- values include at least the constants 0 and 1,
- expressions include at least registers and values,
- expressions do *not* include references: $M[N/x] = M$.

We model the following language.

$$\mu \ ::= \ \mathsf{rlx} \ \mid \ \mathsf{ra} \ \mid \ \mathsf{sc}$$

$$S \ ::= \ \mathsf{skip} \ \mid \ r := M \ \mid \ r := [L]^\mu \ \mid \ [L]^\mu := M \ \mid \ \mathsf{F}^\nu \ \mid \ \mathsf{if}(M)\{S_1\}\,\mathsf{else}\,\{S_2\} \ \mid \ S_1\,;\,S_2 \ \mid \ S_1 \,\rrbracket\, S_2$$
$$\mid \ r := \mathsf{CAS}^{\mu_1,\mu_2}([L], M, N) \ \mid \ r := \mathsf{FADD}^{\mu_1,\mu_2}([L], M) \ \mid \ r := \mathsf{EXCHG}^{\mu_1,\mu_2}([L], M)$$

Memory modes, μ, are relaxed (rlx), release-acquire (ra), and sequentially consistent (sc). Relaxed mode is the default; we regularly elide it from examples. ra/sc accesses are collectively known as *synchronized accesses*.

Commands, aka *statements*, S, include memory accesses at a given mode, as well as the usual structural constructs. Following [Ferreira et al. 1996], $\rrbracket$ denotes parallel composition, preserving thread state on the right after a join. In examples and sublanguages without join, we use the symmetric $\|$ operator.

The semantics is built from the following.

- a set of *events* $\mathcal{E}$, ranged over by e, d, c, b,
- a set of *logical formulae* Φ, ranged over by ϕ, ψ, θ,
- a set of *actions* $\mathcal{A}$, ranged over by a,

Subsets of $\mathcal{E}$ are ranged over by E, D, C, B.

We require that:

- formulae include equalities $(M=N)$ and $(x=M)$,
- formulae are closed under negation, conjunction, disjunction, and substitutions $[M/r]$, $[M/x]$,
- there is a relation $\vDash$ between formulae, capturing entailment,
- $\vDash$ has the expected semantics for $=$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and substitution,
- there are binary relations *matches*, *blocks*, and *delays*, each in $\mathcal{A} \times \mathcal{A}$,
- there are subsets of actions, distinguishing *read* and *release* actions.

Logical formulae include equations over registers, such as $(r=s+1)$. For use in §6.1, we also include equations over memory references, such as $(x=1)$. Formulae are subject to substitutions; actions are not. We use expressions as formulae, coercing M to $M{\neq}0$. Equations have precedence over logical operators; thus $r=v \Rightarrow s>w$ is read $(r=v) \Rightarrow (s>w)$. As usual, implication associates to the right; thus $\phi \Rightarrow \psi \Rightarrow \theta$ is read $\phi \Rightarrow (\psi \Rightarrow \theta)$.

We say ϕ is a *tautology* if $\mathsf{tt} \vDash \phi$. We say ϕ is *unsatisfiable* if $\phi \vDash \mathsf{ff}$.

Throughout §3–5 we additionally require that

- each register is assigned at most once in a program.

In §6, we drop this restriction, requiring instead that

- there are registers $\mathcal{S}_\mathcal{E} = \{s_e \mid e \in \mathcal{E}\}$,
- registers $\mathcal{S}_\mathcal{E}$ do not appear in programs: $S[N/s_e] = S$.

In this paper, we let actions be reads, writes and fences:

$$a, b \ ::= \ \mathsf{W}^\mu x v \ \mid \ \mathsf{R}^\mu x v \ \mid \ \mathsf{F}^\nu$$

In definitions, we drop elements of actions that are existentially quantified. In examples, we drop elements of actions, using defaults. We write (A^μ) to stand for (W^μ) or (R^μ).

Action (Wxv) *matches* (Rxw) when $v = w$.

Action (Wxv) *blocks* (Rxw), for any v, w.

Let $\bowtie_{sc} = \{(A^{sc}, A^{sc})\}$. Let $\bowtie_{co} = \{(Wx, Wx), (Rx, Wx), (Wx, Rx)\}$. Let $\ltimes_{sync} = \{(a, W^{\sqsupseteq ra}), (a, F^{\sqsupseteq rel}), (R, F^{\sqsupseteq acq}), (Rx, R^{\sqsupseteq ra}x), (R^{\sqsupseteq ra}, a), (F^{\sqsupseteq acq}, a), (F^{\sqsupseteq rel}, W), (W^{\sqsupseteq ra}x, Wx)\}$.

Action a *delays* b if $a \bowtie_{sc} b$ or $a \bowtie_{co} b$ or $a \ltimes_{sync} b$.

Actions $(W^{\mu \neq rlx})$ and $(F^{\nu \neq acq})$ are *release* actions.

3.2 Pomsets

We first consider a fragment of our language that can be modeled using simple pomsets. This captures read and write actions which may be reordered, but as we shall see does *not* capture control or data dependencies.

Definition 3.1. A *pomset* over $\mathcal{A}$ is a tuple $(E, \leq, \lambda)$ where

- $E \subset \mathcal{E}$ is a set of *events*,
- $\leq \subseteq (E \times E)$ is the *causality* partial order,
- $\lambda : E \to \mathcal{A}$ is a *labeling*.

Let P range over pomsets, and $\mathcal{P}$ over sets of pomsets. Let Pom be the set of all pomsets.

We lift terminology from actions to events. For example, we say that e writes x if $\lambda(e)$ writes x. We also drop quantifiers when clear from context, such as $(\forall e \in E)(\forall x \in \mathcal{X})$.

Definition 3.2. Action (Wxv) *matches* (Rxw) when $v = w$. Action (Wxv) *blocks* (Rxw), for any v, w.

A read event e is *fulfilled* if there is a $d \leq e$ which matches it and, for any c which can block e, either $c \leq d$ or $e \leq c$.

We introduce reorderability [Mazurkiewicz 1995] in order to provide examples with coherence in this subsection. In §4 we show that coherence can be encoded in the logic, making reorderability unnecessary.

Definition 3.3. Actions a and b are *reorderable* ($a \bowtie b$) if either both are reads or they are accesses to different locations. Formally $\bowtie = \{(Rxv, Ryw)\} \cup \{(Rxv, Wyw), (Wxv, Ryw), (Wxv, Wyw) \mid x \neq y\}$.

Actions that are not reorderable are in *conflict*.

We can now define a model of processes given as sets of pomsets sufficient to give the semantics for a fragment of our language without control or data dependencies.

Definition 3.4. If $P \in NIL$ then $E = \emptyset$.

If $P \in \mathcal{P}_1 \parallel \mathcal{P}_2$ then $(\exists P_1 \in \mathcal{P}_1) \ (\exists P_2 \in \mathcal{P}_2)$

(1) $E = (E_1 \cup E_2)$,
(2) if $d \leq_1 e$ then $d \leq e$,
(3) if $d \leq_2 e$ then $d \leq e$,
(4) if $e \in E_1$ then $\lambda(e) = \lambda_1(e)$,
(5) if $e \in E_2$ then $\lambda(e) = \lambda_2(e)$,
(6) E_1 and E_2 are disjoint.

If $P \in (a \to \mathcal{P}_2)$ then $(\exists E_1) \ (\exists P_2 \in \mathcal{P}_2)$

(1) $E = (E_1 \cup E_2)$,
(2) if $d, e \in E_1$ then $d = e$,

(3) if $d \leq_2 e$ then $d \leq e$,

(4) if $e \in E_1$ then $\lambda(e) = a$,

(5) if $e \in E_2$ then $\lambda(e) = \lambda_2(e)$,

(6) if $d \in E_1$, $e \in E_2$ then either $d \leq e$ or $a \bowtie \lambda_2(e)$.

If $P \in TOP(\mathcal{P})$ then $(\exists P_1 \in \mathcal{P})$

(1) $E = E_1$,

(2) $\lambda(e) = \lambda_1(e)$,

(3) if $d \leq_1 e$ then $d \leq e$,

(4) if $\lambda_1(e)$ is a read then e is fulfilled (Def 3.2).

Definition 3.5. For a language fragment, the semantics is:

$$[\![x^\mu := v\,;\, S]\!] = (\mathsf{W}xv) \to [\![S]\!] \qquad\qquad [\![\mathtt{skip}]\!] = [\![0]\!] = NIL$$

$$[\![r := x^\mu\,;\, S]\!] = \bigcup_v (\mathsf{R}xv) \to [\![S]\!] \qquad\qquad [\![S_1 \parallel S_2]\!] = [\![S_1]\!] \parallel [\![S_2]\!]$$

In this semantics, both skip and 0 map to the empty pomset. Parallel composition is disjoint union, inheriting labeling and order from the two sides. Prefixing may add a new action (on the left) to an existing pomset (on the right), inheriting labeling and order from the right.

It is worth noting that if $\bowtie$ is taken to be the empty relation, then top-level pomsets of Def 3.1 correspond to sequentially consistent executions up to mumbling [Brookes 1996].

Example 3.6. Mumbling is allowed, since there is no requirement that left and right be disjoint in the definition of prefixing. Both of the pomsets below are allowed.

$$x := 1\,;\, x := 1 \qquad\qquad\qquad\qquad x := 1\,;\, x := 1$$

In the left pomset, the order between the events is enforced by clause 6, since the actions are in conflict.

Example 3.7. Although this model enforces coherence, it is very weak. For example, it makes no distinction between synchronizing and relaxed access, thus allowing:

$$x := 0\,;\, x := 1\,;\, y^{\mathsf{ra}} := 1 \parallel r := y^{\mathsf{ra}}\,;\, s := x$$
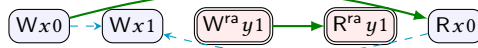
We show how to enforce the intended semantics, where $(\mathsf{W}^{\mathsf{ra}}y1)$ *publishes* $(\mathsf{W}x1)$ in Ex 4.3.

In diagrams, we use different shapes and colors for arrows and events. These are included only to help the reader understand why order is included. We adopt the following conventions (dependency and synchronization order will appear later in the paper):

- relaxed accesses are blue, with a single border,
- synchronized accesses are red, with a double border,
- $e \longrightarrow d$ arises from fulfillment, where e *matches* d,
- $e \dashrightarrow d$ arises either from fulfillment, where e *blocks* d, or from prefixing, where e was prefixed before d and their actions *conflict*,
- $e \longrightarrow d$ arises from control/data/address *dependency*,
- $e \Longrightarrow d$ arises from *synchronized access*.

Definition 3.8. $\mathcal{P}_1$ *refines* $\mathcal{P}_2$ if $\mathcal{P}_1 \subseteq \mathcal{P}_2$.

Example 3.9. Ex 3.6 shows that $[\![x := 1]\!]$ refines $[\![x := 1\,;\, x := 1]\!]$.

3.3 Pomsets with Preconditions

The previous section modeled a language fragment without conditionals (and hence no control dependencies) or expressions (and hence no data dependencies). We now address this, by adopting a *pomsets with preconditions* model similar to [Jagadeesan et al. 2020].

Definition 3.10. A *pomset with preconditions* is a pomset (Def 3.1) together with $\kappa : E \to \Phi$.

Definition 3.11. Let $[\phi/\mathsf{Q}]$ substitute all quiescence symbols by ϕ.

We can now define a model of processes given as sets of pomsets with preconditions sufficient to give the semantics for a fragment of our language where every use of sequential composition is either $(x^\mu := M; S)$ or $(r := x^\mu; S)$.

Definition 3.12. If $P \in NIL$ then $E = \emptyset$.
If $P \in \mathcal{P}_1 \parallel \mathcal{P}_2$ then $(\exists P_1 \in \mathcal{P}_1)$ $(\exists P_2 \in \mathcal{P}_2)$
 1–6) as for $\parallel$ in Def 3.4,
   (7) if $e \in E_1$ then $\kappa(e) \vDash \kappa_1(e)$,
   (8) if $e \in E_2$ then $\kappa(e) \vDash \kappa_2(e)$.
If $P \in IF(\phi, \mathcal{P}_1, \mathcal{P}_2)$ then $(\exists P_1 \in \mathcal{P}_1)$ $(\exists P_2 \in \mathcal{P}_2)$
 1–5) as for $\parallel$ in Def 3.4 (ignoring disjointness),
   (6) if $e \in E_1 \setminus E_2$ then $\kappa(e) \vDash \phi \wedge \kappa_1(e)$,
   (7) if $e \in E_2 \setminus E_1$ then $\kappa(e) \vDash \neg\phi \wedge \kappa_2(e)$,
   (8) if $e \in E_1 \cap E_2$ then
       $\kappa(e) \vDash (\phi \Rightarrow \kappa_1(e)) \wedge (\neg\phi \Rightarrow \kappa_2(e))$.
If $P \in WR(x, M, \mathcal{P}_2)$ then $(\exists P_2 \in \mathcal{P}_2)$ $(\exists v \in \mathcal{V})$
 1–6) as for $(\mathsf{W}xv) \to \mathcal{P}_2$ in Def 3.4,
   (7) if $e \in E_1 \setminus E_2$ then $\kappa(e) \vDash M{=}v$,
   (8) if $e \in E_2 \setminus E_1$ then $\kappa(e) \vDash \kappa_2(e)$,
   (9) if $e \in E_1 \cap E_2$ then $\kappa(e) \vDash M{=}v \vee \kappa_2(e)$.
If $P \in RD(r, x, \mathcal{P}_2)$ then $(\exists P_2 \in \mathcal{P}_2)$ $(\exists v \in \mathcal{V})$
 1–6) as for $(\mathsf{R}xv) \to \mathcal{P}_2$ in Def 3.4,
   (7) if $e \in E_2 \setminus E_1$ then either
       $\kappa(e) \vDash r{=}v \Rightarrow \kappa_2(e)$ and $(\exists d \in E_1)$ $d < e$, or
       $\kappa(e) \vDash \kappa_2(e)$.
If $P \in TOP(\mathcal{P})$ then $(\exists P_1 \in \mathcal{P})$
 1–4) as for *TOP* in Def 3.4,
   (5) if $\lambda_1(e)$ is a write, $\kappa_1(e)[\mathtt{tt}/\mathsf{Q}][\mathtt{tt}/\mathsf{W}]$ is a tautology,
   (6) if $\lambda_1(e)$ is a read, $\kappa_1(e)[\mathtt{tt}/\mathsf{Q}][\mathtt{ff}/\mathsf{W}]$ is a tautology.

Let PomPre be the set of all pomsets with preconditions. The function $TOP : 2^{\mathsf{PomPre}} \to 2^{\mathsf{Pom}}$ embeds sets of pomsets with preconditions into sets of pomsets. It also substitutes formulae for quiescence and write symbols, for use in §4-5. In these "top-level" pomsets, every read is fulfilled and every precondition is a tautology.

Definition 3.13. For a language fragment, the semantics is:

$$[\![\mathtt{if}(M)\{S_1\}\,\mathtt{else}\,\{S_2\}]\!] = IF(M{\neq}0, [\![S_1]\!], [\![S_2]\!])$$
$$[\![x^\mu := M; S]\!] = WR(x, M, [\![S]\!]) \qquad [\![\mathtt{skip}]\!] = [\![0]\!] = NIL$$
$$[\![r := x^\mu; S]\!] = RD(r, x, [\![S]\!]) \qquad [\![S_1 \parallel S_2]\!] = [\![S_1]\!] \parallel [\![S_2]\!]$$

Example 3.14. A simple example of a data dependency is a pomset $P \in [\![r := x \, ; \, y := r]\!]$, for which there must be an $v \in \mathcal{V}$ and $P' \in [\![y := r]\!]$ such as the following, where $v = 1$:

$$y := r$$

$$\boxed{r{=}1 \mid \mathsf{W}\,y\,1}$$

The value chosen for the read may be different from that chosen for the write:

$$r := x \, ; \, y := r$$

$$\boxed{\mathsf{R}\,x\,0} \longrightarrow \boxed{r{=}0 \Rightarrow r{=}1 \mid \mathsf{W}\,y\,1}$$

In this case, the pomset's preconditions depend on a bound register, so cannot contribute to a top-level pomset.

If the values chosen for read and write are compatible, then we have two cases: the independent case, which again cannot be part of a top-level pomset,

$$r := x \, ; \, y := r$$

$$\boxed{\mathsf{R}\,x\,1} \qquad \boxed{r{=}1 \mid \mathsf{W}\,y\,1}$$

and the dependent case:

$$\boxed{\mathsf{R}\,x\,1} \longrightarrow \boxed{r{=}1 \Rightarrow r{=}1 \mid \mathsf{W}\,y\,1}$$

Since $r{=}1 \Rightarrow r{=}1$ is a tautology, this can be part of a top-level pomset.

Example 3.15. Control dependencies are similar, for example for any $P \in [\![r := x \, ; \, \mathtt{if}(r)\{y := 1\}]\!]$, there must be an $v \in \mathcal{V}$ and $P' \in [\![\mathtt{if}(r)\{y := 1\}]\!]$ such as:

$$\mathtt{if}(r)\{y := 1\}$$

$$\boxed{r{\neq}0 \mid \mathsf{W}\,y\,1}$$

The rest of the reasoning is the same as Ex 3.14.

Example 3.16. A simple example of an independency is a pomset $P \in [\![r := x \, ; \, y := 1]\!]$, for which there must be:

$$y := 1$$

$$\boxed{1{=}1 \mid \mathsf{W}\,y\,1}$$

In this case it doesn't matter what value the read chooses:

$$r := x \, ; \, y := 1$$

$$\boxed{\mathsf{R}\,x\,0} \qquad \boxed{1{=}1 \mid \mathsf{W}\,y\,1}$$

Example 3.17. Consider $P \in [\![\mathtt{if}(r{=}1)\{y := r\} \, \mathtt{else} \, \{y := 1\}]\!]$, so there must be $P_1 \in [\![y := r]\!]$, and $P_2 \in [\![y := 1]\!]$, such as:

$$y := r \qquad\qquad\qquad\qquad\qquad y := 1$$

$$\boxed{r{=}1 \mid \mathsf{W}\,y\,1} \qquad\qquad\qquad\qquad \boxed{1{=}1 \mid \mathsf{W}\,y\,1}$$

Since there is no requirement for disjointness in the semantics of conditionals, we can consider the case where the event *coalesces* from the two pomsets, in which case:

$$\mathtt{if}(r{=}1)\{y := r\} \, \mathtt{else} \, \{y := 1\}$$

$$\boxed{(r{=}1 \Rightarrow r{=}1) \wedge (r{\neq}1 \Rightarrow 1{=}1) \mid \mathsf{W}\,y\,1}$$

Here, the precondition is a tautology, independent of r.

3.4 Pomsets with Predicate Transformers

Having reviewed the work we are building on, we now turn to the contribution of this paper, which is a model of *pomsets with predicate transformers*. *Predicate transformers* are functions on formulae which preserve logical structure, providing a natural model of sequential composition.

Definition 3.18. A *predicate transformer* is a function $\tau : \Phi \rightarrow \Phi$ such that

- $\tau(\mathsf{ff})$ is ff,
- $\tau(\psi_1 \wedge \psi_2)$ is $\tau(\psi_1) \wedge \tau(\psi_2)$,
- $\tau(\psi_1 \vee \psi_2)$ is $\tau(\psi_1) \vee \tau(\psi_2)$,
- if $\phi \vDash \psi$, then $\tau(\phi) \vDash \tau(\psi)$.

Note that substitutions $(\tau(\psi) = \psi[M/r])$ and implications on the right $(\tau(\psi) = \phi \Rightarrow \psi)$ are predicate transformers.

As discussed in §1, predicate transformers suffice for sequentially consistent models, but not relaxed models, where dependency calculation is crucial. For dependency calculation, we use a *family* of predicate transformers, indexed by sets of events. We use τ^D as the predicate transformer applied to any event e where if $d \in D$ then $d < e$.

Definition 3.19. A *family of predicate transformers* for E consists of a predicate transformer τ^D for each $D \subseteq \mathcal{E}$, such that if $C \cap E \subseteq D$ then $\tau^C(\psi) \vDash \tau^D(\psi)$.

We write τ as an abbreviation of τ^E.

Definition 3.20. A *pomset with predicate transformers* is a pomset with preconditions (Def 3.12), together with a family of predicate transformers for E.

Definition 3.21. If $P \in ABORT$ then $E = \emptyset$ and

- $\tau^D(\psi) \vDash \mathsf{ff}$.

If $P \in SKIP$ then $E = \emptyset$ and

- $\tau^D(\psi) \vDash \psi$.

If $P \in LET(r, M)$ then $E = \emptyset$ and

- $\tau^D(\psi) \vDash \psi[M/r]$.

If $P \in IF(\phi, \mathcal{P}_1, \mathcal{P}_2)$ then $(\exists P_1 \in \mathcal{P}_1)\ (\exists P_2 \in \mathcal{P}_2)$

 1–8) as for *IF* in Def 3.12,

  (9) $\tau^D(\psi) \vDash (\phi \Rightarrow \tau_1^D(\psi)) \wedge (\neg\phi \Rightarrow \tau_2^D(\psi))$.

If $P \in \mathcal{P}_1 \parallel\!\!\!\mid \mathcal{P}_2$ then $(\exists P_1 \in \mathcal{P}_1)\ (\exists P_2 \in \mathcal{P}_2)$

1–??) as for $\parallel$ in Def 3.12,

  (9) $\tau^D(\psi) \vDash \tau_2^D(\psi)$,

 (10) $\tau^D(\mathsf{s}) \vDash \tau_1^D(\mathsf{s})$, for every quiescence symbol s.

If $P \in \mathcal{P}_1\, ;\, \mathcal{P}_2$ then $(\exists P_1 \in \mathcal{P}_1)\ (\exists P_2 \in \mathcal{P}_2)$

 1–5) as for $\parallel$ in Def 3.4 (ignoring disjointness),

  (6) if $e \in E_1 \setminus E_2$ then $\kappa(e) \vDash \kappa_1(e)$,

  (7) if $e \in E_2 \setminus E_1$ then $\kappa(e) \vDash \kappa_2'(e)$,

  (8) if $e \in E_1 \cap E_2$ then $\kappa(e) \vDash \kappa_1(e) \vee \kappa_2'(e)$,

     where $\kappa_2'(e) = \tau_1^C(\kappa_2(e))$, where $C = \{c \mid c < e\}$,

  (9) $\tau^D(\psi) \vDash \tau_1^D(\tau_2^D(\psi))$.

If $P \in WRITE(x, M, \mu)$ then $(\exists v \in \mathcal{V})$

 S1) if $d, e \in E$ then $d = e$,

 S2) $\lambda(e) = \mathsf{W}xv$,

 S3) $\kappa(e) \vDash M{=}v$,

S4) $\tau^D(\psi) \vDash \psi \wedge M{=}v$,
S5) $\tau^C(\psi) \vDash \psi$,
  where $D \cap E \neq \emptyset$ and $C \cap E = \emptyset$.
If $P \in READ(r, x, \mu)$ then $(\exists v \in \mathcal{V})$
L1) if $d, e \in E$ then $d = e$,
L2) $\lambda(e) = Rxv$,
L3) $\kappa(e) \vDash tt$,
L4) $\tau^D(\psi) \vDash v{=}r \Rightarrow \psi$,
L5) $\tau^C(\psi) \vDash \psi$,
  where $D \cap E \neq \emptyset$ and $C \cap E = \emptyset$,
If $P \in TOP(\mathcal{P})$ then $(\exists P_1 \in \mathcal{P})$
1–6) as in Def 3.12,
  (7) $\tau^{E_1}(s) \vDash s$, for every quiescence symbol s.

Definition 3.22. The semantics of commands is:

$$[\![\text{if}(M)\{S_1\}\text{else}\{S_2\}]\!] = IF(M{\neq}0, [\![S_1]\!], [\![S_2]\!])$$

$$[\![x^\mu := M]\!] = WRITE(x, M, \mu) \qquad [\![\text{skip}]\!] = SKIP$$

$$[\![r := x^\mu]\!] = READ(r, x, \mu) \qquad [\![S_1 \,\|\, S_2]\!] = [\![S_1]\!] \,\|\, [\![S_2]\!]$$

$$[\![r := M]\!] = LET(r, M) \qquad [\![S_1 ; S_2]\!] = [\![S_1]\!] ; [\![S_2]\!]$$

$$[\![F^\mu]\!] = FENCE(\mu)$$

Most of these definitions are straightforward adaptations of §3.3, but the treatment of sequential composition is new. This uses the usual rule for composition of predicate transformers (but preserving the indexing set). For the pomset, we take the union of their events, preserving actions, but crucially in cases 7 and 8 we apply a predicate transformer τ_1^C from the left-hand side to a precondition $\kappa_2(e)$ from the right-hand side to build the precondition $\kappa_2'(e)$. The indexing set C for the predicate transformer is $\{c \mid c < e\}$, so can depend on the causal order.

Example 3.23. For read to write dependency, consider:

Putting these together without order, we calculate the precondition $\kappa(e)$ as $\tau_1^C(\kappa_2(e))$, where C is $\{c \mid c < e\}$, which is $\emptyset$. Since $\tau_1^\emptyset(\psi)$ is ψ, this gives that $\kappa(e)$ is $\kappa_2(e)$, which is $r{=}1$. This gives the pomsaet with predicate transformers:

This pomset's preconditions depend on a bound register, so cannot contribute to a top-level pomset.

Putting them together with order, we calculate the precondition $\kappa(e)$ as $\tau_1^C(\kappa_2(e))$, where C is $\{c \mid c < e\}$, which is $\{d\}$. Since $\tau_1^{\{d\}}(\psi)$ is $(1{=}r \Rightarrow \psi)$, this gives that $\kappa(e)$ is $(1{=}r \Rightarrow \kappa_2(e))$, which is $(1{=}r \Rightarrow r{=}1)$. This gives the pomset with predicate transformers:

This pomset's preconditions do not depend on a bound register, so can contribute to a top-level pomset.

Example 3.24. If the read and write choose different values:

$$r := x \qquad\qquad\qquad\qquad y := r$$

$\boxed{\mathsf{R}\,x1}\cdots\!\!\gg\boxed{1{=}r \Rightarrow \psi}\ \boxed{\psi} \qquad\qquad \boxed{r{=}2 \mid \mathsf{W}\,y2}\cdots\!\!\gg\boxed{\psi}\ \boxed{r{=}2 \wedge \psi}$

Putting these together with order, we have the following, which cannot be part of a top-level pomset:

$$r := x \,;\, y := r$$

$\overset{d}{}\boxed{\mathsf{R}\,x1}\xrightarrow{\;e\;}\boxed{1{=}r \Rightarrow r{=}2 \mid \mathsf{W}\,y2}$

$\boxed{1{=}r \Rightarrow \psi}\quad\boxed{1{=}r \Rightarrow (r{=}2 \wedge \psi)}\quad\boxed{r{=}2 \wedge \psi}\qquad\boxed{\psi}$
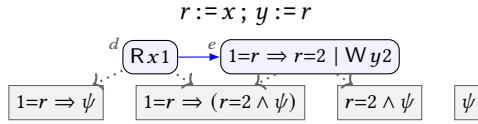
Example 3.25. S4 includes $M{=}v$ to ensure that spurious merges do not go undetected. Consider the following.

$$x := 1 \qquad\qquad\qquad\qquad x := 2$$

$\boxed{1{=}1 \mid \mathsf{W}\,x1}\cdots\!\!\gg\boxed{1{=}1 \wedge \psi}\ \boxed{\psi} \qquad\qquad \boxed{2{=}1 \mid \mathsf{W}\,x1}\cdots\!\!\gg\boxed{2{=}1 \wedge \psi}\ \boxed{\psi}$

Merging the actions, since 2=1 is unsatisfiable, we have:

$$x := 1 \,;\, x := 2$$

$\boxed{\mathsf{W}\,x1}\cdots\!\!\gg\boxed{\mathsf{ff}}\ \boxed{\psi}$
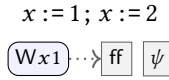
This pomset cannot be part of a top-level pomset, since $\tau^E(\mathsf{s}) = \mathsf{ff}$ for every quiescence symbol s. This is what we would hope: that the program $x := 1 \,;\, x := 2$ should only be top-level if there is a $(\mathsf{W}\,x2)$ event.

Example 3.26. The predicate transformer we have chosen for L4 is different from the one used traditionally, which is written using substitution. Substitution is also used in [Jagadeesan et al. 2020]. Attempting to write the predicate transformers in this style we have:

L4) $\tau^D(\psi) \vDash \psi[v/r]$,
L5) $\tau^C(\psi) \vDash (\forall r)\psi$.

This phrasing of L5 says that ψ must be independent of r in order to appear in a top-level pomset. This choice for L5 is forced by Def 3.19, which states that the predicate transformer for a small subset of E must imply the transformer for a larger subset.

Sadly, this definition fails associativity.

Consider the following, eliding transformers:

$$r := y \qquad\quad x := !r \qquad\qquad x := !!r \qquad\qquad x := 0$$

$\boxed{\mathsf{R}\,y1}\qquad\boxed{r{=}0 \mid \mathsf{W}\,x1}\qquad\boxed{r{\neq}0 \mid \mathsf{W}\,x1}\qquad\boxed{\mathsf{W}\,x0}$

Associating to the right and merging:

$$r := y \qquad\qquad\qquad x := !r \,;\, x := !!r \,;\, x := 0$$

$\boxed{\mathsf{R}\,y1}\qquad\boxed{r{=}0 \vee r{\neq}0 \mid \mathsf{W}\,x1}\!-\!\rightarrow\boxed{\mathsf{W}\,x0}$

The precondition of $(\mathsf{W}\,x1)$ is a tautology, thus we have:

$$r := y \,;\, x := !r \,;\, x := !!r \,;\, x := 0$$
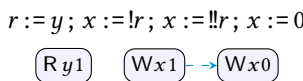
$\boxed{\mathsf{R}\,y1}\qquad\boxed{\mathsf{W}\,x1}\!-\!\rightarrow\boxed{\mathsf{W}\,x0}$

If, instead, we associate to the left:

$$r := y\,;\ x := !r \qquad\qquad\qquad x := !!r\,;\ x := 0$$

$$\boxed{\mathsf{R}\,y1} \quad \boxed{1{=}0 \mid \mathsf{W}x1} \qquad\qquad \boxed{r{\neq}0 \mid \mathsf{W}x1}\,\dashrightarrow\,\boxed{\mathsf{W}x0}$$

Sequencing and merging:

$$r := y\,;\ x := !r\,;\ x := !!r\,;\ x := 0$$

$$\boxed{\mathsf{R}\,y1} \qquad \boxed{1{=}0 \lor r{\neq}0 \mid \mathsf{W}x1}\,\dashrightarrow\,\boxed{\mathsf{W}x0}$$

In this case, the precondition of $(\mathsf{W}x1)$ is not a tautology, forcing a dependency $(\mathsf{R}\,y1) \longrightarrow (\mathsf{W}x1)$.

Our solution is to Skolemize. We have proven associativity of Def 3.21 in Agda. The proof requires that predicate transformers distribute through disjunction (Def 3.18). The attempt to define predicate transformers using substitution fails for L5 because the predicate transformer $\tau(\psi) = (\forall r)\psi$ does not distribute through disjunction: $\tau(\psi_1 \lor \psi_2) = (\forall r)(\psi_1 \lor \psi_2) \neq ((\forall r)(\psi_1)) \lor ((\forall r)(\psi_2)) = \tau(\psi_1) \lor \tau(\psi_2)$.

3.5 The Road Ahead

The final semantic functions for load, store, and thread initialization are given in Fig 1, at the end of the paper. In §4–6, we explain this definition by looking at its constituent parts, building on Def 3.21. In §4, we add *quiescence*, which encodes coherence, release-acquire access, and SC access. In §5, we add peculiarities that are necessary for efficient implementation on Arm8. In §6, we discuss other features such as invariant reasoning, case analysis and register recycling.

The final definitions of load and store are quite complex, due to the inherent complexities of relaxed memory. The core of Def 3.21, modeling sequential composition, parallel composition, and conditionals, is stable, remaining unchanged in later sections. The messiness of relaxed memory is quarantined to the rules for load and store, rather than permeating the entire semantics.

4 QUIESCENCE

We introduce *quiescence*, which captures *coherence* and *synchronized access*. Recall from §3.1 that formulae include symbols $\mathsf{Q_{sc}}$, $\mathsf{Q_{ro}^x}$, and $\mathsf{Q_{wo}^x}$. We refer to these collectively as *quiescence symbols*. In this section, we will show how these logical symbols can be used to capture coherence and synchronization. This illustrates a feature of our model, which is that many features of weak memory can be cautured in the logic, not in the pomset model itself.

4.1 Coherence (co)

In the logic, the quiescence symbols are just uninterpreted formula, but the semantics uses them as preconditions, to ensure appropriate causal order. For example, *write-write coherence* enforces order between writes to the same location in the same thread. We model this by adding the precondition $(\mathsf{Q_{ro}^x} \land \mathsf{Q_{wo}^x})$ to events that write to x, for example:

$$x := 1\,;\ x := 2$$

$$\boxed{1{=}1 \land \mathsf{Q_{ro}^x} \land \mathsf{Q_{wo}^x} \mid \mathsf{W}x1}\,\dashrightarrow\,\boxed{2{=}2 \land \mathsf{Q_{ro}^x} \land \mathsf{Q_{wo}^x} \mid \mathsf{W}x2}$$

These symbols are left alone in the dependent case, but in the independent case we substitute ff for $\mathsf{Q_{wo}^x}$:

$$\boxed{1{=}1 \land \mathsf{Q_{ro}^x} \land \mathsf{Q_{wo}^x} \mid \mathsf{W}x1} \qquad \boxed{2{=}2 \land \mathsf{Q_{ro}^x} \land \mathsf{ff} \mid \mathsf{W}x2}$$

This substitution is part of the predicate transformer:

$$x := 1$$

$$\boxed{1{=}1 \wedge Q_{ro}^x \wedge Q_{wo}^x \mid Wx1} \dashrightarrow \boxed{\psi} \qquad \boxed{\psi[\mathrm{ff}/Q_{wo}^x]}$$

We treat read-write and write-read coherence similarly:

$$r := x$$

$$\boxed{Q_{wo}^x \mid Rx1} \dashrightarrow \boxed{r{=}1 \Rightarrow \psi} \qquad \boxed{\psi[\mathrm{ff}/Q_{ro}^x]}$$

In this model, there is no read-read coherence, but to restore it we would identify Q_{ro}^x with Q_{wo}^x.

When threads are initialized, we substitute every quiesence symbol with tt, so at top level there are no remaining quiescence symbols, for example:

$$x := 1;\ x := 2 \parallel r := x$$

$$\boxed{1{=}1 \wedge \mathrm{tt} \wedge \mathrm{tt} \mid Wx1} \dashrightarrow \boxed{2{=}2 \wedge \mathrm{tt} \wedge \mathrm{tt} \mid Wx2}$$
$$\boxed{\mathrm{tt} \mid Rx1} \dashrightarrow$$
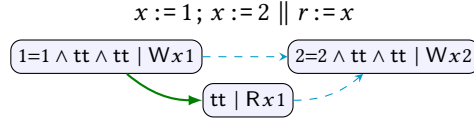
Definition 4.1 (co). Update Def 3.21:

S3) $\kappa(e)$ implies $Q_{ro}^x \wedge Q_{wo}^x \wedge M{=}v$,
L3) $\kappa(e)$ implies Q_{wo}^x,

S5) $\tau^C(\psi)$ implies $\psi[\mathrm{ff}/Q_{wo}^x]$,
L5) $\tau^C(\psi)$ implies $\psi[\mathrm{ff}/Q_{ro}^x]$.

Example 4.2. Def 4.1 enforces coherence. Consider:

$$x := 1 \qquad\qquad\qquad\qquad x := 2$$

$$\boxed{1{=}1 \wedge Q_{ro}^x \wedge Q_{wo}^x \mid Wx1} \qquad\qquad \boxed{2{=}2 \wedge Q_{ro}^x \wedge Q_{wo}^x \mid Wx2}$$
$$\boxed{\psi \wedge 1{=}1} \qquad\qquad\qquad\qquad \boxed{\psi \wedge 2{=}2}$$
$$\boxed{\psi[\mathrm{ff}/Q_{wo}^x]} \qquad\qquad\qquad\qquad \boxed{\psi[\mathrm{ff}/Q_{wo}^x]}$$

Simplifying, we have:

$$\boxed{Q_{ro}^x \wedge Q_{wo}^x \mid Wx1} \qquad\qquad\qquad \boxed{Q_{ro}^x \wedge Q_{wo}^x \mid Wx2}$$
$$\boxed{\psi[\mathrm{ff}/Q_{wo}^x]} \dashrightarrow \boxed{\psi} \qquad\qquad \boxed{\psi} \quad \boxed{\psi[\mathrm{ff}/Q_{wo}^x]}$$
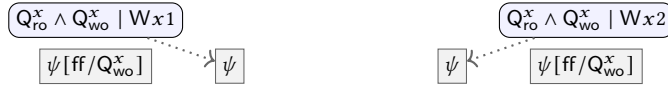
If we attempt to put these together unordered, the precondition of $(Wx2)$ becomes unsatisfiable:

$$x := 1;\ x := 2$$

$$\boxed{Q_{ro}^x \wedge Q_{wo}^x \mid Wx1} \qquad\qquad \boxed{Q_{ro}^x \wedge \mathrm{ff} \mid Wx2}$$
$$\dashrightarrow \boxed{\psi[\mathrm{ff}/Q_{wo}^x]} \dashrightarrow \boxed{\psi} \quad \boxed{\psi[\mathrm{ff}/Q_{wo}^x]} \quad \boxed{\psi[\mathrm{ff}/Q_{wo}^x]}$$
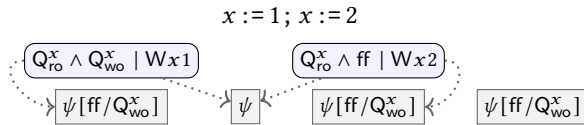
In order to get a satisfiable precondition for $(Wx2)$, we must introduce order:

$$\boxed{Q_{ro}^x \wedge Q_{wo}^x \mid Wx1} \dashrightarrow \boxed{Q_{ro}^x \wedge Q_{wo}^x \mid Wx2}$$
$$\dashrightarrow \boxed{\psi[\mathrm{ff}/Q_{wo}^x]} \dashrightarrow \boxed{\psi} \quad \boxed{\psi[\mathrm{ff}/Q_{wo}^x]} \quad \boxed{\psi[\mathrm{ff}/Q_{wo}^x]}$$
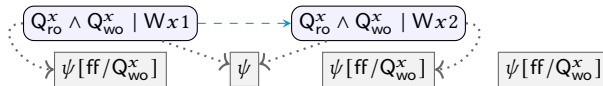
4.2   Synchronized Access (SYNC)

Example 4.3. The publication idiom requires that we disallow the execution below, which is allowed by Def 4.1.

$$x := 0;\ x := 1;\ y^{\mathsf{ra}} := 1 \parallel r := y^{\mathsf{ra}};\ s := x$$
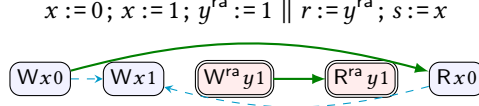
We disallow this by introducing order $(\mathsf{W}x1) \rightarrow (\mathsf{W}^{\mathsf{ra}}y1)$ and $(\mathsf{R}^{\mathsf{ra}}y1) \rightarrow (\mathsf{R}x0)$.

Definition 4.4. Let $\mathsf{Q}_{\mathsf{ro}}^* = \bigwedge_y \mathsf{Q}_{\mathsf{ro}}^y$, and similarly for $\mathsf{Q}_{\mathsf{wo}}^*$.
Let formulae $\mathsf{Q}_\mu^{\mathsf{S}x}$ and $\mathsf{Q}_\mu^{\mathsf{L}x}$ be defined:

$$\mathsf{Q}_{\mathsf{rlx}}^{\mathsf{S}x} = \mathsf{Q}_{\mathsf{ro}}^x \wedge \mathsf{Q}_{\mathsf{wo}}^x \qquad\qquad \mathsf{Q}_{\mathsf{rlx}}^{\mathsf{L}x} = \mathsf{Q}_{\mathsf{wo}}^x$$
$$\mathsf{Q}_{\mathsf{ra}}^{\mathsf{S}x} = \mathsf{Q}_{\mathsf{ro}}^* \wedge \mathsf{Q}_{\mathsf{wo}}^* \qquad\qquad \mathsf{Q}_{\mathsf{ra}}^{\mathsf{L}x} = \mathsf{Q}_{\mathsf{wo}}^x$$
$$\mathsf{Q}_{\mathsf{sc}}^{\mathsf{S}x} = \mathsf{Q}_{\mathsf{ro}}^* \wedge \mathsf{Q}_{\mathsf{wo}}^* \wedge \mathsf{Q}_{\mathsf{sc}} \qquad\qquad \mathsf{Q}_{\mathsf{sc}}^{\mathsf{L}x} = \mathsf{Q}_{\mathsf{wo}}^x \wedge \mathsf{Q}_{\mathsf{sc}}$$

Let $[\phi/\mathsf{Q}_{\mathsf{ro}}^*]$ substitute ϕ for every $\mathsf{Q}_{\mathsf{ro}}^y$, and similarly for $\mathsf{Q}_{\mathsf{wo}}^*$. Let substitutions $[\phi/\mathsf{Q}_\mu^{\mathsf{S}x}]$ and $[\phi/\mathsf{Q}_\mu^{\mathsf{L}x}]$ be defined:

$$[\phi/\mathsf{Q}_{\mathsf{rlx}}^{\mathsf{S}x}] = [\phi/\mathsf{Q}_{\mathsf{wo}}^x] \qquad\qquad [\phi/\mathsf{Q}_{\mathsf{rlx}}^{\mathsf{L}x}] = [\phi/\mathsf{Q}_{\mathsf{ro}}^x]$$
$$[\phi/\mathsf{Q}_{\mathsf{ra}}^{\mathsf{S}x}] = [\phi/\mathsf{Q}_{\mathsf{wo}}^x] \qquad\qquad [\phi/\mathsf{Q}_{\mathsf{ra}}^{\mathsf{L}x}] = [\phi/\mathsf{Q}_{\mathsf{ro}}^*, \phi/\mathsf{Q}_{\mathsf{wo}}^*]$$
$$[\phi/\mathsf{Q}_{\mathsf{sc}}^{\mathsf{S}x}] = [\phi/\mathsf{Q}_{\mathsf{wo}}^x, \phi/\mathsf{Q}_{\mathsf{sc}}] \qquad\qquad [\phi/\mathsf{Q}_{\mathsf{sc}}^{\mathsf{L}x}] = [\phi/\mathsf{Q}_{\mathsf{ro}}^*, \phi/\mathsf{Q}_{\mathsf{wo}}^*, \phi/\mathsf{Q}_{\mathsf{sc}}]$$
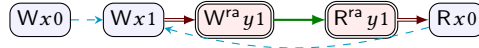
Definition 4.5 (CO/SYNC). Update Def 3.21 to:

S3) $\kappa(e)$ implies $\mathsf{Q}_\mu^{\mathsf{S}x} \wedge M{=}v$,
L3) $\kappa(e)$ implies $\mathsf{Q}_\mu^{\mathsf{L}x}$,

S5) $\tau^C(\psi)$ implies $\psi[\mathsf{ff}/\mathsf{Q}_\mu^{\mathsf{S}x}]$,
L5) $\tau^C(\psi)$ implies $\psi[\mathsf{ff}/\mathsf{Q}_\mu^{\mathsf{L}x}]$.

The quiescence formulae indicate what must precede an event. For example, all preceding accesses must be ordered before a releasing write, whereas only writes on x must be ordered before a releasing read on x.

The quiescence substitutions update quiescence symbols in subsequent code. For subsequent independent code, *S5* and *L5* substitute false. In top-level pomsets, *TOP* substitutes true (Def 3.12). For example, we substitute ff for $\mathsf{Q}_{\mathsf{ra}}^{\mathsf{S}x}$ in the independent case for a releasing write; this ensures that subsequent writes to x follow the releasing write in top-level pomsets. Similarly, we substitute ff for $\mathsf{Q}_{\mathsf{ra}}^{\mathsf{L}x}$ in the independent case for an acquiring write; this ensures that all subsequent accesses follow the acquiring read in top-level pomsets.

Example 4.6. The following pomsets show the effect of quiescence for each access mode.

$$x := M$$

$$\psi[\text{ff}/Q^x_{\text{wo}}]$$

$$Q^x_{\text{ro}} \wedge Q^x_{\text{wo}} \wedge M{=}v \mid \text{W}xv$$

$$\psi \wedge M{=}v$$

$$r := x$$

$$\psi[\text{ff}/Q^x_{\text{ro}}]$$

$$Q^x_{\text{wo}} \mid \text{R}xv$$

$$v{=}r \Rightarrow \psi$$

$$x^{\text{ra}} := M$$

$$\psi[\text{ff}/Q^x_{\text{wo}}]$$

$$Q^*_{\text{ro}} \wedge Q^*_{\text{wo}} \wedge M{=}v \mid \text{W}^{\text{ra}}xv$$

$$\psi \wedge M{=}v$$

$$r := x^{\text{ra}}$$
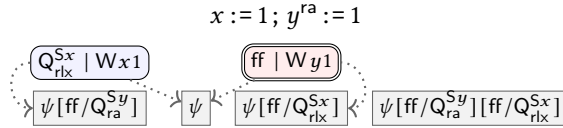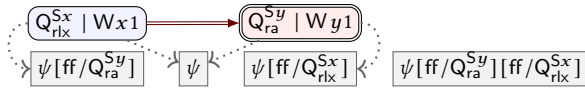
$$\psi[\text{ff}/Q^*_{\text{ro}}][\text{ff}/Q^*_{\text{wo}}]$$

$$Q^x_{\text{wo}} \mid \text{R}^{\text{ra}}xv$$

$$v{=}r \Rightarrow \psi$$

$$x^{\text{sc}} := M$$

$$\psi[\text{ff}/Q^x_{\text{wo}}][\text{ff}/Q_{\text{sc}}]$$

$$Q^*_{\text{ro}} \wedge Q^*_{\text{wo}} \wedge Q_{\text{sc}} \wedge M{=}v \mid \text{W}^{\text{sc}}xv$$

$$\psi \wedge M{=}v$$

$$r := x^{\text{sc}}$$

$$\psi[\text{ff}/Q^*_{\text{ro}}][\text{ff}/Q^*_{\text{wo}}][\text{ff}/Q_{\text{sc}}]$$

$$Q^x_{\text{wo}} \wedge Q_{\text{sc}} \mid \text{R}^{\text{sc}}xv$$

$$v{=}r \Rightarrow \psi$$
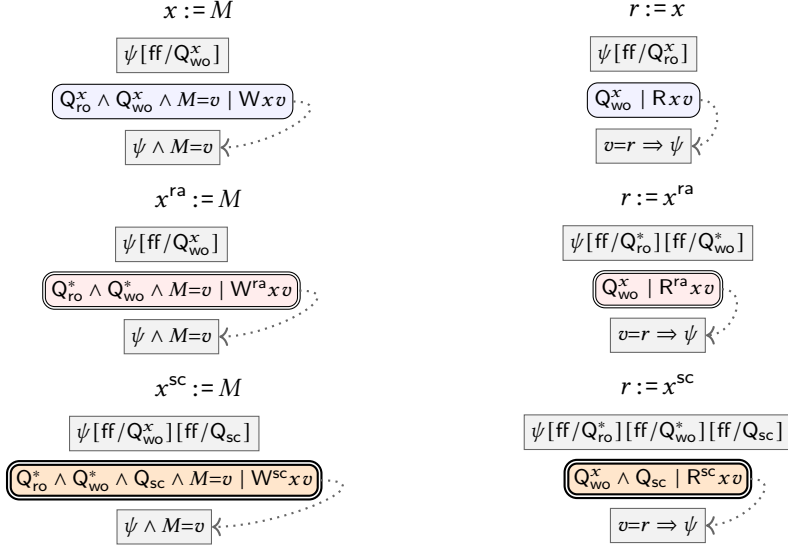
Example 4.7. Def 4.1 enforces publication. Consider:

$$x := 1$$

$$1{=}1 \wedge Q^{Sx}_{\text{rlx}} \mid \text{W}x1$$

$$1{=}1 \wedge \psi$$

$$\psi[\text{ff}/Q^{Sx}_{\text{rlx}}]$$

$$y^{\text{ra}} := 1$$

$$1{=}1 \wedge Q^{Sy}_{\text{ra}} \mid \text{W}y2$$

$$1{=}1 \wedge \psi$$

$$\psi[\text{ff}/Q^{Sy}_{\text{ra}}]$$

Since $Q^{Sy}_{\text{ra}}[\text{ff}/Q^{Sx}_{\text{rlx}}]$ is ff, composing these without order simplifies to:

$$x := 1 \, ; \, y^{\text{ra}} := 1$$

$$Q^{Sx}_{\text{rlx}} \mid \text{W}x1 \qquad \text{ff} \mid \text{W}y1$$

$$\psi[\text{ff}/Q^{Sy}_{\text{ra}}] \quad \psi \quad \psi[\text{ff}/Q^{Sx}_{\text{rlx}}] \quad \psi[\text{ff}/Q^{Sy}_{\text{ra}}][\text{ff}/Q^{Sx}_{\text{rlx}}]$$

In order to get a satisfiable precondition for $(\text{W}y1)$, we must introduce order:

$$Q^{Sx}_{\text{rlx}} \mid \text{W}x1 \longrightarrow Q^{Sy}_{\text{ra}} \mid \text{W}y1$$

$$\psi[\text{ff}/Q^{Sy}_{\text{ra}}] \quad \psi \quad \psi[\text{ff}/Q^{Sx}_{\text{rlx}}] \quad \psi[\text{ff}/Q^{Sy}_{\text{ra}}][\text{ff}/Q^{Sx}_{\text{rlx}}]$$

Example 4.8. Def 4.1 enforces subscription. Consider:
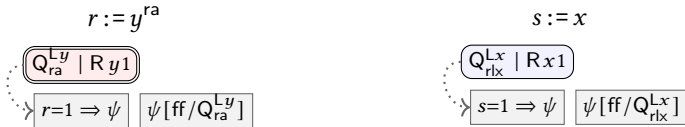
$$r := y^{\text{ra}}$$

$$Q^{Ly}_{\text{ra}} \mid \text{R}y1$$

$$r{=}1 \Rightarrow \psi \quad \psi[\text{ff}/Q^{Ly}_{\text{ra}}]$$

$$s := x$$

$$Q^{Lx}_{\text{rlx}} \mid \text{R}x1$$

$$s{=}1 \Rightarrow \psi \quad \psi[\text{ff}/Q^{Lx}_{\text{rlx}}]$$

Since $Q^{Lx}_{\text{rlx}}[\text{ff}/Q^{Ly}_{\text{ra}}]$ is ff, composing these without order simplifies to:

$$r := y^{\text{ra}} \, ; \, s := x$$

$$Q^{Ly}_{\text{ra}} \mid \text{R}y1 \qquad \psi[\text{ff}/Q^{Ly}_{\text{ra}}][\text{ff}/Q^{Lx}_{\text{rlx}}] \qquad \text{ff} \mid \text{R}x1$$

$$r{=}1 \Rightarrow \psi[\text{ff}/Q^{Lx}_{\text{rlx}}] \quad r{=}1 \Rightarrow s{=}1 \Rightarrow \psi \quad s{=}1 \Rightarrow \psi[\text{ff}/Q^{Ly}_{\text{ra}}]$$
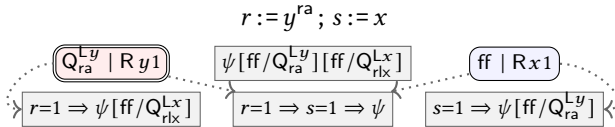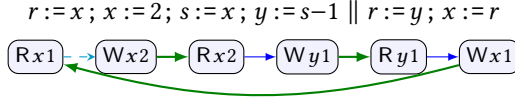
In order to get a satisfiable precondition for $(\mathsf{R}\,x1)$, we must introduce order:
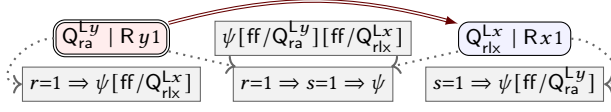

5 EFFICIENT IMPLEMENTATION ON ARMV8

We discuss Arm8 using *external completion* (EC) [Alglave 2020] [Arm Limited 2020, §B2.3.6] which is very close to our model.

Example 5.1. Bad example:

$$r := x\,;\, x := 2\,;\, s := x\,;\, y := s{-}1 \parallel r := y\,;\, x := r$$

Example 5.2. Should not wipe out local state only when there is a release-acquire pair, as in the definition of [Jagadeesan et al. 2020], which allows the following execution. Even though there is a blocker $(\mathsf{W}x3)$, here we are using $(\mathsf{W}x0)$ to justify the lack of dependence $(\mathsf{R}\,x4) \longrightarrow (\mathsf{W}w4)$.

$$x := 0\,;\, \text{if}(x)\{y := 2\}\,;\, \text{if}(z^{\mathsf{ra}})\{s := x\,;\, \text{if}(s\ \text{even})\{w := 4\}\} \parallel x := 2\,;\, \text{if}(y^{\mathsf{ra}})\{x := 3\,;\, z^{\mathsf{ra}} := 1\}\,;\, x := w$$
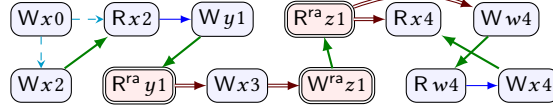

(The read-clobbering semantics from §B.4 does not change this.)

It does seem to me like a non-local blocker for program-order must include a read of x or a release (to get order from the write) and an acquire (to get the order to the read).

What if an acquire preceded by a release or non-local read on x wiped the local state of x? The problem with this kind of solution (and P6 in ooplsa specifically) is that it destroys roach-motel.

5.1 Downgraded Reads (DGR)

Example 5.3. The following example is from Podkopaev [Podkopaev 2020]. It is allowed by Arm8, but disallowed by Def 4.5. The coherence order between the writes can be witnessed by a separate thread, which we have elided.

$$x := 2\,;\, r := x^{\mathsf{ra}}\,;\, y := 1 \parallel y := 2\,;\, x^{\mathsf{ra}} := 1$$
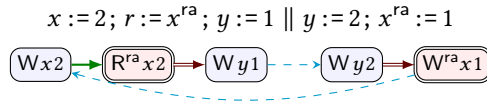
Under EGC, this is explained by dropping the order $(\mathsf{R}^{\mathsf{ra}}x2) \longrightarrow (\mathsf{W}y1)$, because $(\mathsf{R}^{\mathsf{ra}}x2)$ is fulfilled by a relaxed write in the same thread.
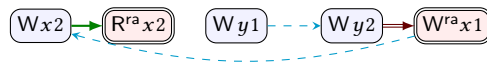
More generally, this can be understood as a compiler optimization that downgrades a read from ra to rlx when it can be fulfilled by a relaxed write in the same thread.

To model such *downgraded reads*, we use the uninterpreted symbols $\downarrow^x$.

Definition 5.4. Let $[\phi/{\downarrow}^*]$ substitute ϕ for every ${\downarrow}^y$.
Let formula ${\downarrow}^x_\mu$ and substitution $[\mu/{\downarrow}^x]$ be defined:

$$
{\downarrow}^x_\mu = \begin{cases} {\downarrow}^x & \text{if } \mu \neq \mathsf{rlx} \\ \mathsf{tt} & \text{otherwise} \end{cases}
\qquad
[\mu/{\downarrow}^x] = \begin{cases} [\mathsf{tt}/{\downarrow}^x] & \text{if } \mu = \mathsf{rlx} \\ [\mathsf{tt}/{\downarrow}^x][\mathsf{ff}/{\downarrow}^*] & \text{otherwise} \end{cases}
$$

Definition 5.5 (CO/SYNC/DGR). Update Def 4.5 to:

S4) $\tau^D(\psi)$ implies $\psi[\mu/{\downarrow}^x] \wedge M{=}v,$
S5) $\tau^C(\psi)$ implies $\psi[\mu/{\downarrow}^x][\mathsf{ff}/\mathsf{Q}^{Sx}_\mu],$
L5) $\tau^C(\psi)$ implies ${\downarrow}^x_\mu \wedge \psi[\mathsf{ff}/\mathsf{Q}^{Lx}_\mu].$

Load actions that require downgrading introduce ${\downarrow}^x$. Relaxed stores on x substitute true for ${\downarrow}^x$, whereas synchronizing stores substitute false for ${\downarrow}^x$.

Example 5.6. Revisiting Ex 5.3 and eliding irrelevant transformers:
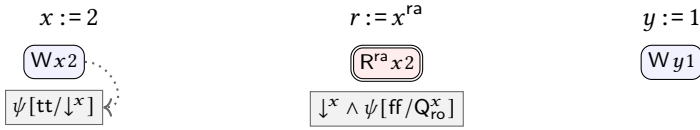
Associating right:

Composing, we have, as desired:
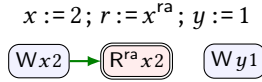
Example 5.7. One might worry that our model is too permissive for sc access, but Arm8 itself allows some very counterintuitive results for sc access. In the following execution we elide the initializing write ($\mathsf{W}y0$).
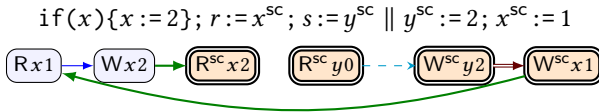
Under EGC, this is explained by dropping the order $(\mathsf{R}^{sc}x2) \rightarrow (\mathsf{R}y0)$, because $(\mathsf{R}^{sc}x2)$ is fulfilled by a relaxed write in the same thread.

5.2 Removing Read-Read dependencies (RRD)

Example 5.8. The following execution is allowed by Arm8, but disallowed by Def 4.5.
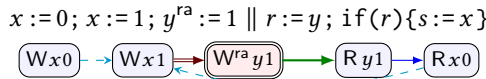
Under EGC, this is explained by dropping the order $(\mathsf{R}y1) \rightarrow (\mathsf{R}x0)$, because Arm8 does not include control dependencies between reads in the locally-ordered-before relation.
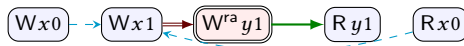
Since we do not distinguish control dependencies from other dependencies, we are forced to drop all dependencies between reads. In order to do so, we use the uninterpreted symbol W.

*Definition 5.9 (*RRD*).* Update Def 3.21 to:

L4) $\tau^D(\psi)$ implies $v{=}r \Rightarrow \psi$,

L5) $\tau^C(\psi)$ implies $(v{=}r \vee \mathsf{W}) \Rightarrow \psi$, when $E \neq \emptyset$,

L6) $\tau^B(\psi)$ implies ψ, when $E = \emptyset$.

L4 is the *dependent* case, L5 the *independent* case, and L6 the *empty* case. Recall the substitutions performed by *TOP* (Def 3.12). For writes, L5 is the same as L6 (since W is tt). For reads, L5 is the same as L4 (since W is ff).

One reading of L5 is that when satisfying a precondition ϕ it is safe to use the value of the read, as long as the action is not a write. If the pomset is empty, however, there is no value v to use. Therefore the best we can do is to emulate skip, as in L6.

Example 5.10. Revisiting Ex 5.8 and eliding irrelevant transformers:

$$r := y \qquad\qquad\qquad\qquad \mathtt{if}(r)\{s := x\}$$

$$\boxed{\mathsf{R}\,y1}\;\; \boxed{(1{=}r \vee \mathsf{W}) \Rightarrow \psi} \qquad\qquad \boxed{r \mid \mathsf{R}\,x0}$$

Composing sequentially:

$$r := y\,;\, \mathtt{if}(r)\{s := x\}$$

$$\boxed{\mathsf{R}\,y1}\qquad \boxed{(1{=}r \vee \mathsf{W}) \Rightarrow r \mid \mathsf{R}\,x0}$$

At top-level, *TOP* yields:

$$\boxed{\mathsf{R}\,y1}\qquad \boxed{(1{=}r \vee \mathsf{ff}) \Rightarrow r \mid \mathsf{R}\,x0}$$

The precondition of $(\mathsf{R}\,x0)$ is a tautology, as required.

5.3 Full Semantics For Arm

Def 5.11 combines all of the features of §4–5.

*Definition 5.11 (*CO/SYNC/DGR/RRD*).* Let $[\phi/\mathsf{Q}]$ be the substitution $[\phi/\mathsf{Q}^*_{\mathsf{ro}}][\phi/\mathsf{Q}^*_{\mathsf{wo}}][\phi/\mathsf{Q}_{\mathsf{sc}}]$. Update Def 3.21 to:

S3) $\kappa(e)$ implies $\mathsf{Q}^{\mathsf{S}x}_\mu \wedge M{=}v$,

L3) $\kappa(e)$ implies $\mathsf{Q}^{\mathsf{L}x}_\mu$.

S4) $\tau^D(\psi)$ implies $\psi[\mu/{\downarrow}^x] \wedge M{=}v$

S5) $\tau^C(\psi)$ implies $\psi[\mu/{\downarrow}^x][\mathsf{ff}/\mathsf{Q}^{\mathsf{S}x}_\mu]$

L4) $\tau^D(\psi)$ implies $v{=}r \Rightarrow \psi$,

L5) $\tau^C(\psi)$ implies ${\downarrow}^x_\mu \wedge ((v{=}r \vee \mathsf{W}) \Rightarrow \psi[\mathsf{ff}/\mathsf{Q}^{\mathsf{L}x}_\mu])$,

L6) $\tau^B(\psi)$ implies ${\downarrow}^x_\mu \wedge \psi$, when $E = \emptyset$.

Example 5.12. RRD does not adversely affect subscription (Ex 4.8).

$$r := y^{\mathsf{ra}} \qquad\qquad\qquad\qquad s := x$$

$$\boxed{\boxed{\mathsf{Q}^{\mathsf{L}y}_{\mathsf{ra}} \mid \mathsf{R}^{\mathsf{ra}}\,y1}}\;\boxed{{\downarrow}^y \wedge (\mathsf{W} \Rightarrow \psi[\mathsf{ff}/\mathsf{Q}^{\mathsf{L}x}_{\mathsf{ra}}])} \qquad\qquad \boxed{\mathsf{Q}^{\mathsf{L}y}_{\mathsf{ra}} \mid \mathsf{W}\,x0}$$

Since $\mathsf{Q}^{\mathsf{L}y}_{\mathsf{ra}}[\mathsf{ff}/\mathsf{Q}^{\mathsf{L}x}_{\mathsf{ra}}]$ is false, we have:

$$r := y^{\mathsf{ra}}\,;\, s := x$$

$$\boxed{\boxed{\mathsf{Q}^{\mathsf{L}y}_{\mathsf{ra}} \mid \mathsf{R}^{\mathsf{ra}}\,y1}}\qquad \boxed{{\downarrow}^y \wedge (\mathsf{W} \Rightarrow \mathsf{ff}) \mid \mathsf{W}\,x0}$$

Applying *TOP* yields:

$$\boxed{\boxed{\mathsf{tt} \mid \mathsf{R}^{\mathsf{ra}}\,y1}} \qquad \boxed{{\downarrow}^{y} \wedge (\mathsf{ff} \Rightarrow \mathsf{ff}) \mid \mathsf{W}x0}$$

Although $(\mathsf{ff} \Rightarrow \mathsf{ff})$ is a tautology, ${\downarrow}^{y}$ is not. Thus, the pomset can only contribute to a top-level pomset if the thread is preceded by a relaxed write to y, allowing $(\mathsf{R}^{\mathsf{ra}}\,y1)$ to be downgraded to a relaxed read.

Every Arm8 execution is allowed by Def 5.11. The proof of this fact is simplified by the recent characterization of Arm8 in terms of *external global completion* (EGC) [Arm Limited 2020, §B2.3.6]. Under EGC, an Arm8 execution is a linearization of per-location program order and a subset of local-order. Every such linearization is also a valid pomset under Def 5.11.

6 OTHER FEATURES

6.1 Local Invariant Reasoning (LIR)

Example 6.1. JMM causality Test Case 1 [Pugh 2004] states the following execution should be allowed "since interthread compiler analysis could determine that x and y are always non-negative, allowing simplification of $r{\geq}0$ to true, and allowing write $y := 1$ to be moved early."

$$x := 0; \; \mathsf{fork}\,\{(r := x\,;\, \mathsf{if}(r{\geq}0)\{y := 1\} \parallel x := y)\}$$
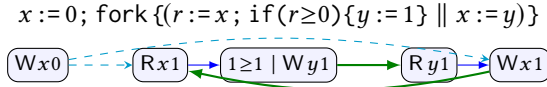
Under the definitions given thus far, the precondition on $(\mathsf{W}y1)$ can only be satisfied by the read of x, disallowing this execution.

In order to allow such executions, we include memory references in formula, resulting in:
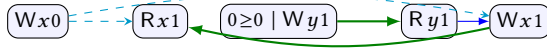
Definition 6.2 (LIR). Update Def 3.21 to:

S4) $\tau^{D}(\psi)$ implies $\psi[M/x] \wedge M{=}v$,
S5) $\tau^{C}(\psi)$ implies $\psi[M/x]$,
L4) $\tau^{D}(\psi)$ implies $v{=}r \Rightarrow \psi$,
L5) $\tau^{C}(\psi)$ implies $(v{=}r \vee x{=}r) \Rightarrow \psi$, when $E \neq \emptyset$,
L6) $\tau^{B}(\psi)$ implies ψ, when $E = \emptyset$.

L5 introduces memory references. It states that to be independent of the read, we must establish both $\psi[v/r]$ and $\psi[x/r]$. If a precondition holds in both circumstances, S5 allows a local write to satisfy the precondition without introducing dependence. As in Def 5.9, we include L6 to provide a predicate transformer for the empty pomset.

Example 6.3. Revisiting Ex 6.1 and eliding irrelevant transformers:
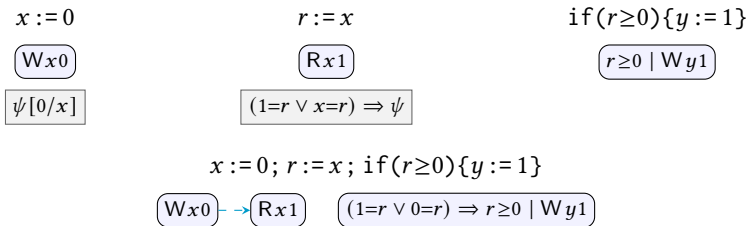
The precondition of $(\mathsf{W}y1)$ is a tautology, as required.

If $P \in WRITE(x, M, \mu)$ then $(\exists v : E \to \mathcal{V})\ (\exists \theta : E \to \Phi)$

    S1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

    S2) $\lambda(e) = \mathsf{W}xv_e$,

    S3) $\kappa(e)$ implies $\theta_e \wedge \mathsf{Q}_\mu^{\mathsf{S}x} \wedge M{=}v_e$,

    S4) $(\forall e \in E \cap D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow (\psi[M/x][\mu/{\downarrow}^x] \wedge M{=}v_e)$,

    S5) $(\forall e \in E \setminus D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow \psi[M/x][\mu/{\downarrow}^x][\mathsf{ff}/\mathsf{Q}_\mu^{\mathsf{S}x}]$.

If $P \in READ(r, x, \mu)$ then $(\exists v : E \to \mathcal{V})\ (\exists \theta : E \to \Phi)$

    L1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

    L2) $\lambda(e) = \mathsf{R}xv_e$,

    L3) $\kappa(e)$ implies $\theta_e \wedge \mathsf{Q}_\mu^{\mathsf{L}x}$,

    L4) $(\forall e \in E \cap D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow (v_e{=}s_e) \Rightarrow \psi[s_e/r]$,

    L5) $(\forall e \in E \setminus D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow {\downarrow}_\mu^x \wedge (\langle v_e{=}s_e \vee (\mathsf{W} \wedge x{=}s_e)\rangle \Rightarrow \psi[s_e/r][\mathsf{ff}/\mathsf{Q}_\mu^{\mathsf{L}x}])$,

    L6) $(\forall s)\ \tau^D(\psi)$ implies $(\nexists e \in E \mid \theta_e) \Rightarrow ({\downarrow}_\mu^x \wedge \psi[s/r][\mathsf{ff}/\mathsf{Q}_\mu^{\mathsf{L}x}])$.

Fig. 1. Full Semantics of Loads and Stores (See Def 4.4 for $\mathsf{Q}_\mu^{\mathsf{S}x}, \mathsf{Q}_\mu^{\mathsf{L}x}$ and Def 5.4 for ${\downarrow}_\mu^x, [\mu/{\downarrow}^x]$)

If L5 required only that $x{=}r \Rightarrow \psi$, then the following execution would be allowed:

$$x := 0;\ r := x;\ \mathsf{if}(r{\geq}0)\{y := 1\}$$

$$\boxed{\mathsf{W}x0} \dashrightarrow \boxed{\mathsf{R}x\text{-}1} \qquad \boxed{\mathsf{W}y1}$$

But this would violate the expected local invariant: that all values seen for x are nonnegative.

It is worth emphasizing that this reasoning is local, and therefore unaffected by the introduction of additional threads, as in Test Case 9 [Pugh 2004].

Some care is necessary when combining LIR Def 6.2 and RRD Def 5.9. The proper form for L5 is:

L5) $\tau^C(\psi)$ implies $(v{=}r \vee (\mathsf{W} \wedge x{=}r)) \Rightarrow \psi$.

When W is true, this is unchanged from L5 in Def 6.2. When W is false, it is the same as L4 in Def 6.2.

One must also be careful when combing LIR with RMW operations, such as CAS. For these operations, LIR is unsound. Thus reads in RMWs should always use L6 for the independent case.

6.2 Register Recycling (ALPHA)

The semantics considered thus far assume that each register is assigned at most once in a program. We relax this by renaming.

Example 6.4. JMM causality Test Case 2 [Pugh 2004] states the following execution should be allowed "since redundant read elimination could result in simplification of $r{=}s$ to true, allowing $y := 1$ to be moved early."

$$r := x;\ s := x;\ \mathsf{if}(r{=}s)\{y := 1\} \parallel x := y$$

$$\boxed{\mathsf{R}x1} \longleftrightarrow \boxed{\mathsf{W}y1} \longrightarrow \boxed{\mathsf{R}y1} \longrightarrow \boxed{\mathsf{W}x1}$$

This execution is not allowed under Def 6.2, since the precondition of $(\mathsf{W}y1)$ in the independent case is

$$(r{=}1 \vee r{=}x) \Rightarrow (s{=}1 \vee s{=}r) \Rightarrow (r{=}s),$$

which is not a tautology. Our solution is to rename registers using the set $\mathcal{S}_{\mathcal{E}} = \{s_e \mid e \in \mathcal{E}\}$, which are banned from source programs, as per §3.1. This allows us to resolve nondeterminism in loads when merging, resulting in:
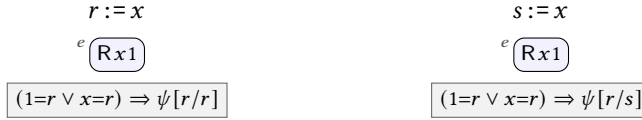
$$\boxed{\mathsf{R}\,x1} \longleftarrow \boxed{\mathsf{W}\,y1} \longrightarrow \boxed{\mathsf{R}\,y1} \longrightarrow \boxed{\mathsf{W}\,x1}$$
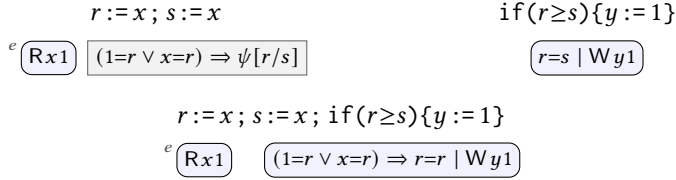
Definition 6.5 (ALPHA). Update Def 3.21 to:

L4) $\tau^D(\psi)$ implies $v{=}s_e \Rightarrow \psi[s_e/r]$,
L5) $(\forall s)\ \tau^C(\psi)$ implies $\psi[s/r]$.

Example 6.6. Revisiting Ex 6.4 and choosing $s_e = r$:

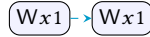$$r := x \qquad\qquad\qquad\qquad\qquad s := x$$
$$^e\boxed{\mathsf{R}\,x1} \qquad\qquad\qquad\qquad\qquad ^e\boxed{\mathsf{R}\,x1}$$
$$\boxed{(1{=}r \vee x{=}r) \Rightarrow \psi[r/r]} \qquad\qquad\qquad \boxed{(1{=}r \vee x{=}r) \Rightarrow \psi[r/s]}$$

Coalescing and composing:

$$r := x\,;\, s := x \qquad\qquad\qquad\qquad \mathtt{if}(r{\geq}s)\{y := 1\}$$
$$^e\boxed{\mathsf{R}\,x1}\ \boxed{(1{=}r \vee x{=}r) \Rightarrow \psi[r/s]} \qquad\qquad \boxed{r{=}s \mid \mathsf{W}\,y1}$$

$$r := x\,;\, s := x\,;\, \mathtt{if}(r{\geq}s)\{y := 1\}$$
$$^e\boxed{\mathsf{R}\,x1}\ \ \boxed{(1{=}r \vee x{=}r) \Rightarrow r{=}r \mid \mathsf{W}\,y1}$$

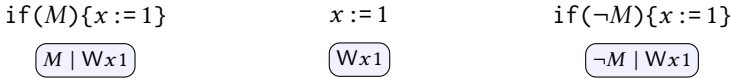The precondition of $(\mathsf{W}\,y1)$ is a tautology, as required.

6.3 If-Closure (IF)

Example 6.7. If $S = (x := 1)$, then Def 3.21 does *not* allow:

$$\mathtt{if}(M)\{x := 1\}\,;\, S\,;\, \mathtt{if}(\neg M)\{x := 1\}$$
$$\boxed{\mathsf{W}\,x1} \dashrightarrow \boxed{\mathsf{W}\,x1}$$

However, if $S = (\mathtt{if}(\neg M)\{x := 1\}\,;\, \mathtt{if}(M)\{x := 1\})$, then it *does* allow the execution. Looking at the initial program:

$$\mathtt{if}(M)\{x := 1\} \qquad\qquad x := 1 \qquad\qquad \mathtt{if}(\neg M)\{x := 1\}$$
$$\boxed{M \mid \mathsf{W}\,x1} \qquad\qquad \boxed{\mathsf{W}\,x1} \qquad\qquad \boxed{\neg M \mid \mathsf{W}\,x1}$$

The difficulty is that the middle action can coalesce either with the right action, or the left, but not both. Thus, we are stuck with some non-tautological precondition. Our solution is to allow a pomset to contain many events for a single action, as long as the events have disjoint preconditions.

This is not simply a theoretical question; it is observable. For example, Def 3.21 does not allow the following.

$$r := y\,;\, \mathtt{if}(r)\{x := 1\}\,;\, x := 1\,;\, \mathtt{if}(\neg r)\{x := 1\}\,;\, z := r$$
$$\parallel \mathtt{if}(x)\{x := 0\,;\, \mathtt{if}(x)\{y := 1\}\}$$

$$\boxed{\mathsf{R}\,y1}\ \boxed{\mathsf{W}\,x1} \dashrightarrow \boxed{\mathsf{W}\,x1}\ \boxed{\mathsf{W}\,z1}$$
$$\boxed{\mathsf{R}\,x1} \dashrightarrow \boxed{\mathsf{W}\,x0} \dashrightarrow \boxed{\mathsf{R}\,x1} \longrightarrow \boxed{\mathsf{W}\,y1}$$
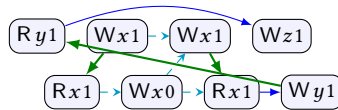
Definition 6.8 (ALPHA/IF). Update Def 3.21 to:
If $P \in WRITE(x, M, \mu)$ then $(\exists v : E \to \mathcal{V})\ (\exists \theta : E \to \Phi)$

S1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

S2) $\lambda(e) = \mathsf{W}xv_e$,

S3) $\kappa(e)$ implies $\theta_e \wedge M{=}v$,

S4) $(\forall e \in E \cap D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow (\psi \wedge M{=}v)$,

S5) $\tau^C(\psi)$ implies $(\nexists e \in E \cap C \mid \theta_e) \Rightarrow \psi$,

If $P \in READ(r, x, \mu)$ then $(\exists v : E \to \mathcal{V})\ (\exists \theta : E \to \Phi)$

L1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

L2) $\lambda(e) = \mathsf{R}xv_e$,

L3) $\kappa(e)$ implies θ_e.

L4) $(\forall e \in E \cap D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow v_e{=}s_e \Rightarrow \psi[s_e/r]$,

L5) $(\forall s)\ \tau^C(\psi)$ implies $(\nexists e \in E \mid \theta_e) \Rightarrow \psi[s/r]$.

Example 6.9. Revisiting Ex 6.7, we can split the middle command:

$$\mathrm{if}(M)\{x := 1\} \qquad\qquad x := 1 \qquad\qquad \mathrm{if}(\neg M)\{x := 1\}$$

$$^d\boxed{M \mid \mathsf{W}x1} \qquad ^d\boxed{\neg M \mid \mathsf{W}x1}\,^e\boxed{M \mid \mathsf{W}x1} \qquad ^e\boxed{\neg M \mid \mathsf{W}x1}$$

Coalescing events gives the desired result.

These examples show that we must allow inconsistent predicates in a single pomset, unlike [Jagadeesan et al. 2020].

7 CONCLUSIONS

We have presented the first model of relaxed memory that treats sequential composition as a first-class citizen. The model builds directly on [Jagadeesan et al. 2020].

For sequential composition, parallel composition and the conditional, we believe that the definition is *natural*, even *canonical*. For stores and loads, instead, the definition in Fig 1 is a Frankenstein's monster of features. This complexity is *essential*, however, not just an accident of our poor choices. Relaxed memory models must please many audiences: compiler writers want one thing, hardware designers another, and programmers yet another still. The result is inevitably full of compromise.

Given that *complexity* cannot be eliminated from relaxed memory models, the best one can do is attempt to understand its causes. We have broken the problem into seven manageable pieces, discussed throughout §4–6. Def 5.11 summarizes all the features necessary for efficient implementation on Arm8. We discuss address calculation, read-modify-write operations and fences in the appendix.

Logic is the thread that sews these features together.

REFERENCES

Jade Alglave. 2020. This commit adds three alternative formulations of the Arm model, both for non-mixed and mixed size accesses. https://github.com/herd/herdtools7/commit/685ee4.

Jade Alglave, Luc Maranget, and Michael Tautschnig. 2014. Herding Cats: Modelling, Simulation, Testing, and Data Mining for Weak Memory. *ACM Trans. Program. Lang. Syst.* 36, 2, Article 7 (July 2014), 74 pages. https://doi.org/10.1145/2627752

Arm Limited. 2020. Arm Architecture Reference Manual: Armv8, for Armv8-A Architecture Profile (Issue F.c). https://developer.arm.com/documentation/ddi0487/latest.

Mark Batty, Scott Owens, Susmit Sarkar, Peter Sewell, and Tjark Weber. 2011. Mathematizing C++ Concurrency. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Austin, Texas, USA) *(POPL '11)*. ACM, New York, NY, USA, 55–66. https://doi.org/10.1145/1926385.1926394

Hans-J. Boehm and Brian Demsky. 2014. Outlawing Ghosts: Avoiding Out-of-thin-air Results. In *Proceedings of the Workshop on Memory Systems Performance and Correctness* (Edinburgh, United Kingdom) *(MSPC '14)*. ACM, New York, NY, USA, Article 7, 6 pages. https://doi.org/10.1145/2618128.2618134

Stephen D. Brookes. 1996. Full Abstraction for a Shared-Variable Parallel Language. *Inf. Comput.* 127, 2 (1996), 145–163. https://doi.org/10.1006/inco.1996.0056

Soham Chakraborty and Viktor Vafeiadis. 2019. Grounding thin-air reads with event structures. *PACMPL* 3, POPL (2019), 70:1–70:28. https://doi.org/10.1145/3290383

Russ Cox. 2016. Go's Memory Model. http://nil.csail.mit.edu/6.824/2016/notes/gomem.pdf.

Edsger W. Dijkstra. 1975. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Commun. ACM* 18, 8 (1975), 453–457. https://doi.org/10.1145/360933.360975

Stephen Dolan, KC Sivaramakrishnan, and Anil Madhavapeddy. 2018. Bounding Data Races in Space and Time. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Philadelphia, PA, USA) *(PLDI 2018)*. ACM, New York, NY, USA, 242–255. https://doi.org/10.1145/3192366.3192421

Brijesh Dongol, Radha Jagadeesan, and James Riely. 2019. Modular transactions: bounding mixed races in space and time. In *Proceedings of the 24th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2019, Washington, DC, USA, February 16-20, 2019*, Jeffrey K. Hollingsworth and Idit Keidar (Eds.). ACM, 82–93. https://doi.org/10.1145/3293883.3295708

William Ferreira, Matthew Hennessy, and Alan Jeffrey. 1996. A Theory of Weak Bisimulation for Core CML. In *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming, ICFP 1996, Philadelphia, Pennsylvania, USA, May 24-26, 1996*, Robert Harper and Richard L. Wexelblat (Eds.). ACM, 201–212. https://doi.org/10.1145/232627.232649

Jay L. Gischer. 1988. The equational theory of pomsets. *Theoretical Computer Science* 61, 2 (1988), 199–224. https://doi.org/10.1016/0304-3975(88)90124-7

C.A.R. Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (Oct. 1969), 576–580. https://doi.org/10.1145/363235.363259

Radha Jagadeesan, Alan Jeffrey, and James Riely. 2020. Pomsets with preconditions: a simple model of relaxed memory. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 194:1–194:30. https://doi.org/10.1145/3428262

Radha Jagadeesan, Corin Pitcher, and James Riely. 2010a. Generative Operational Semantics for Relaxed Memory Models. In *Proceedings of the 19th European Conference on Programming Languages and Systems* (Paphos, Cyprus) *(ESOP'10)*. Springer-Verlag, Berlin, Heidelberg, 307–326. https://doi.org/10.1007/978-3-642-11957-6_17

Radha Jagadeesan, Corin Pitcher, and James Riely. 2010b. Generative Operational Semantics for Relaxed Memory Models. In *Programming Languages and Systems, 19th European Symposium on Programming, ESOP 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6012)*, Andrew D. Gordon (Ed.). Springer, 307–326. https://doi.org/10.1007/978-3-642-11957-6_17

Alan Jeffrey and James Riely. 2016. On Thin Air Reads Towards an Event Structures Model of Relaxed Memory. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, M. Grohe, E. Koskinen, and N. Shankar (Eds.). ACM, 759–767. https://doi.org/10.1145/2933575.2934536

Jeehoon Kang, Chung-Kil Hur, Ori Lahav, Viktor Vafeiadis, and Derek Dreyer. 2017. A promising semantics for relaxed-memory concurrency. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 175–189. http://dl.acm.org/citation.cfm?id=3009850

Ryan Kavanagh and Stephen Brookes. 2018. A denotational account of C11-style memory. *CoRR* abs/1804.04214 (2018). arXiv:1804.04214 http://arxiv.org/abs/1804.04214

Ori Lahav, Viktor Vafeiadis, Jeehoon Kang, Chung-Kil Hur, and Derek Dreyer. 2017. Repairing sequential consistency in C/C++11. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, Albert Cohen and Martin T. Vechev (Eds.). ACM, 618–632. https://doi.org/10.1145/3062341.3062352

Leslie Lamport. 1979. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Trans. Comput.* 28, 9 (Sept. 1979), 690–691. https://doi.org/10.1109/TC.1979.1675439

Sung-Hwan Lee, Minki Cho, Anton Podkopaev, Soham Chakraborty, Chung-Kil Hur, Ori Lahav, and Viktor Vafeiadis. 2020. Promising 2.0: global optimizations in relaxed memory concurrency. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 362–376. https://doi.org/10.1145/3385412.3386010

Lun Liu, Todd Millstein, and Madanlal Musuvathi. 2019. Accelerating Sequential Consistency for Java with Speculative Compilation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. ACM, New York, NY, USA, 16–30. https://doi.org/10.1145/3314221.3314611

Jeremy Manson, William Pugh, and Sarita V. Adve. 2005. The Java Memory Model. *SIGPLAN Not.* 40, 1 (Jan. 2005), 378–391. https://doi.org/10.1145/1047659.1040336

Antoni W. Mazurkiewicz. 1995. Introduction to Trace Theory. In *The Book of Traces*, Volker Diekert and Grzegorz Rozenberg (Eds.). World Scientific, 3–41. https://doi.org/10.1142/9789814261456_0001

If $P \in \text{WRITE}(L, M, \mu)$ then $(\exists \ell : E \to \mathcal{V})\,(\exists v : E \to \mathcal{V})\,(\exists \theta : E \to \Phi)$

    S1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

    S2) $\lambda(e) = \mathsf{W}[\ell_e]v_e$,

    S3) $\kappa(e)$ implies $\theta_e \wedge \mathsf{Q}_\mu^{\mathsf{S}[\ell_e]} \wedge L{=}\ell_e \wedge M{=}v_e$,

    S4) $(\forall e \in E \cap D)\; \tau^D(\psi)$ implies $\theta_e \Rightarrow (\psi[M/[\ell_e]][\mu/\downarrow^{[\ell_e]}] \wedge L{=}\ell_e \wedge M{=}v_e)$,

    S5) $(\forall e \in E \setminus D)\; \tau^D(\psi)$ implies $\theta_e \Rightarrow \psi[M/[\ell_e]][\mu/\downarrow^{[\ell_e]}][\mathsf{ff}/\mathsf{Q}_\mu^{\mathsf{S}[\ell_e]}]$,

    S6) $(\forall k)\; \tau^D(\psi)$ implies $(\nexists e \in E \mid \theta_e) \Rightarrow (L{=}k) \Rightarrow \psi[M/[k]][\mu/\downarrow^{[k]}][\mathsf{ff}/\mathsf{Q}_\mu^{\mathsf{S}[k]}]$.

If $P \in \text{READ}(r, L, \mu)$ then $(\exists \ell : E \to \mathcal{V})\,(\exists v : E \to \mathcal{V})\,(\exists \theta : E \to \Phi)$

    L1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

    L2) $\lambda(e) = \mathsf{R}[\ell_e]v_e$,

    L3) $\kappa(e)$ implies $\theta_e \wedge \mathsf{Q}_\mu^{\mathsf{L}[\ell_e]} \wedge L{=}\ell_e$,

    L4) $(\forall e \in E \cap D)\; \tau^D(\psi)$ implies $\theta_e \Rightarrow (L{=}\ell_e \Rightarrow v_e{=}s_e) \Rightarrow \psi[s_e/r]$,

    L5) $(\forall e \in E \setminus D)\; \tau^D(\psi)$ implies $\theta_e \Rightarrow \downarrow_\mu^{[\ell_e]} \wedge (\langle (L{=}\ell_e \Rightarrow v_e{=}s_e) \vee (\mathsf{W} \wedge (L{=}\ell_e \Rightarrow [\ell_e]{=}s_e)) \rangle \Rightarrow \psi[s_e/r][\mathsf{ff}/\mathsf{Q}_\mu^{\mathsf{L}[\ell_e]}])$,

    L6) $(\forall k)(\forall s)\; \tau^D(\psi)$ implies $(\nexists e \in E \mid \theta_e) \Rightarrow (L{=}k) \Rightarrow (\downarrow_\mu^{[k]} \wedge \psi[s/r][\mathsf{ff}/\mathsf{Q}_\mu^{\mathsf{L}[k]}])$.

Fig. 2. Full Semantics with Address Calculation (See Def 4.4 for $\mathsf{Q}_\mu^{\mathsf{S}x}$, $\mathsf{Q}_\mu^{\mathsf{L}x}$ and Def 5.4 for $\downarrow_\mu^x$, $[\mu/\downarrow^x]$)

Jean Pichon-Pharabod and Peter Sewell. 2016. A Concurrency Semantics for Relaxed Atomics That Permits Optimisation and Avoids Thin-air Executions. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (St. Petersburg, FL, USA) *(POPL '16)*. ACM, New York, NY, USA, 622–633. https://doi.org/10.1145/2837614.2837616

Gordon D. Plotkin and Vaughan R. Pratt. 1996. Teams can see pomsets. In *Partial Order Methods in Verification, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, July 24-26, 1996 (DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 29)*, Doron A. Peled, Vaughan R. Pratt, and Gerard J. Holzmann (Eds.). DIMACS/AMS, 117–128. https://doi.org/10.1090/dimacs/029/07

Anton Podkopaev. 2020. Private correspondence.

Anton Podkopaev, Ori Lahav, and Viktor Vafeiadis. 2019. Bridging the gap between programming languages and hardware weak memory models. *Proc. ACM Program. Lang.* 3, POPL (2019), 69:1–69:31. https://doi.org/10.1145/3290382

William Pugh. 2004. Causality Test Cases. https://perma.cc/PJT9-XS8Z

Conrad Watt, Christopher Pulte, Anton Podkopaev, Guillaume Barbier, Stephen Dolan, Shaked Flur, Jean Pichon-Pharabod, and Shu-yu Guo. 2020. Repairing and mechanising the JavaScript relaxed memory model. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 346–361. https://doi.org/10.1145/3385412.3385973

Conrad Watt, Andreas Rossberg, and Jean Pichon-Pharabod. 2019. Weakening WebAssembly. *Proc. ACM Program. Lang.* 3, OOPSLA (2019), 133:1–133:28. https://doi.org/10.1145/3360559

A ADDITIONAL FEATURES

A.1 Address Calculation (ADDR)

Fig 2 describes the full semantics with address calculation.
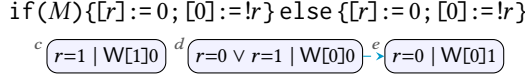
Definition A.1 (ADDR). Update Def 3.21 to existentially quantify over ℓ in *WRITE* and *READ*:

  S2) $\lambda(e) = \mathsf{W}[\ell]v$,

  L2) $\lambda(e) = \mathsf{R}[\ell]v$.

  S3) $\kappa(e) \vDash L{=}\ell \wedge M{=}v$,

  L3) $\kappa(e) \vDash L{=}\ell$.

  L4) $\tau^D(\psi) \vDash (L{=}\ell \Rightarrow v{=}r) \Rightarrow \psi$,

  L5) $\tau^C(\psi) \vDash \psi$.

Example A.2. Def A.1 is naive with respect to merging events. Consider the following example from [Jagadeesan et al. 2020]:
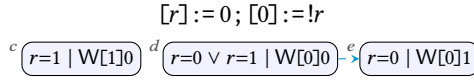
$$[r] := 0 \,;\, [0] := !r \qquad\qquad [r] := 0 \,;\, [0] := !r$$

$${}^{c} \boxed{r{=}1 \mid \mathsf{W[1]}0} \,{}^{d}\!\boxed{r{=}1 \mid \mathsf{W[0]}0} \qquad\qquad {}^{d}\boxed{r{=}0 \mid \mathsf{W[0]}0} \,{}^{e}\!\boxed{r{=}0 \mid \mathsf{W[0]}1}$$

Merging, we have:

$$\mathtt{if}(M)\{[r]{:=}0\,;\,[0]{:=}!r\}\,\mathtt{else}\,\{[r]{:=}0\,;\,[0]{:=}!r\}$$

$${}^{c}\boxed{r{=}1 \mid \mathsf{W[1]}0} \,{}^{d}\!\boxed{r{=}0 \vee r{=}1 \mid \mathsf{W[0]}0} \,{}^{e}\!\boxed{r{=}0 \mid \mathsf{W[0]}1}$$

The precondition of W[0]0 is a tautology; however, this is not possible for $([r]{:=}0\,;\,[0]{:=}!r)$ alone, using Def A.1. The full semantics, given in Fig 2, enables this execution using if-closure. The individual commands have the pomsets:

$$[r] := 0 \qquad\qquad\qquad [0] := !r$$

$${}^{c}\boxed{r{=}1 \mid \mathsf{W[1]}0} \qquad\qquad {}^{d}\boxed{r{=}1 \mid \mathsf{W[0]}0}$$

$${}^{d}\boxed{r{=}0 \mid \mathsf{W[0]}0} \qquad\qquad {}^{e}\boxed{r{=}0 \mid \mathsf{W[0]}1}$$

Sequencing and merging, we have:

$$[r] := 0 \,;\, [0] := !r$$

$${}^{c}\boxed{r{=}1 \mid \mathsf{W[1]}0} \,{}^{d}\!\boxed{r{=}0 \vee r{=}1 \mid \mathsf{W[0]}0} \,{}^{e}\!\boxed{r{=}0 \mid \mathsf{W[0]}1}$$
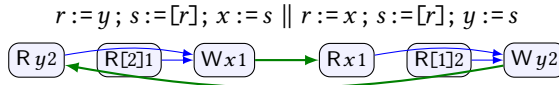
The precondition of (W[0]0) is a tautology, as required.

Example A.3. The combination of read-read independency and address calculation (ADDR/RRD) is somewhat delicate. Combing Def 5.9 and Def A.1 we have:

L4) $\tau^D(\psi) \vDash (L{=}\ell \Rightarrow v{=}r) \Rightarrow \psi$,

L5) $\tau^C(\psi) \vDash ((L{=}\ell \Rightarrow v{=}r) \vee \mathsf{W}) \Rightarrow \psi$.

If we replace the use of $(L{=}\ell \Rightarrow v{=}r)$ by $(v{=}r)$, thin air reads are possible. The subsection of §C on Causal Strengthening discusses this example using the semantics of [Jagadeesan et al. 2020].

Consider the following program, from [Jagadeesan et al. 2020, §5], where initially $x = 0$, $y = 0$, $[0] = 0, [1] = 2$, and $[2] = 1$. It should only be possible to read 0, disallowing the attempted execution below:

$$r := y \,;\, s := [r] \,;\, x := s \parallel r := x \,;\, s := [r] \,;\, y := s$$

$$\boxed{\mathsf{R}\,y2} \leftarrow \boxed{\mathsf{R[2]}1} \rightarrow \boxed{\mathsf{W}\,x1} \longrightarrow \boxed{\mathsf{R}\,x1} \quad \boxed{\mathsf{R[1]}2} \rightarrow \boxed{\mathsf{W}\,y2}$$

Looking at the left thread:

$$r := y \qquad\qquad\qquad s := [r] \qquad\qquad\qquad x := s$$

$$\boxed{(2{=}r \vee \mathsf{W}) \Rightarrow \psi} \qquad \boxed{((r{=}2 \Rightarrow 1{=}s) \vee \mathsf{W}) \Rightarrow \psi} \qquad \boxed{\psi}$$

$$\boxed{\mathsf{R}\,y2}\cdots \qquad\qquad \boxed{r{=}2 \mid \mathsf{R[2]}1}\cdots \qquad\qquad \boxed{s{=}1 \mid \mathsf{W}\,x1}$$

$$\boxed{2{=}r \Rightarrow \psi} \,\cdot \qquad \boxed{(r{=}2 \Rightarrow 1{=}s) \Rightarrow \psi}\,\cdot \qquad \boxed{\psi}\,\cdot$$

Composing, we have:

$$r := y \,;\, s := [r] \,;\, x := s$$

$$\boxed{\mathsf{R}\,y2} \quad \boxed{(2{=}r \vee \mathsf{W}) \Rightarrow r{=}2 \mid \mathsf{R[2]}1}$$

$$\boxed{(2{=}r \vee \mathsf{W}) \Rightarrow (r{=}2 \Rightarrow 1{=}s) \Rightarrow s{=}1 \mid \mathsf{W}\,x1}$$

Substituting for W:

$$\boxed{\text{R } y2} \quad \boxed{(2{=}r \lor \text{ff}) \Rightarrow r{=}2 \mid \text{R}[2]1}$$
$$\boxed{(2{=}r \lor \text{tt}) \Rightarrow (r{=}2 \Rightarrow 1{=}s) \Rightarrow s{=}1 \mid \text{W}x1}$$

Which is:

$$\boxed{\text{R } y2} \quad \boxed{2{=}r \Rightarrow r{=}2 \mid \text{R}[2]1}$$
$$\boxed{(r{=}2 \Rightarrow 1{=}s) \Rightarrow s{=}1 \mid \text{W}x1}$$
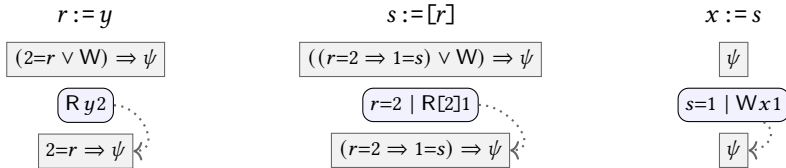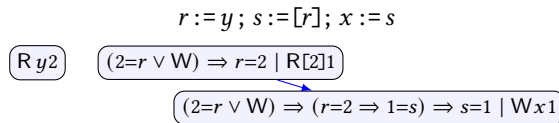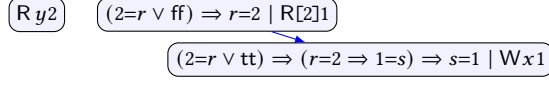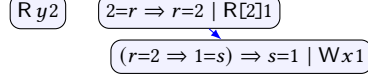
The precondition of (R[2]1) is a tautology, but the precondition of (Wx1) is not. This forces a dependency:

$$r := y \,;\, s := [r] \,;\, x := s$$

$$\boxed{\text{R } y2} \quad \boxed{2{=}r \Rightarrow r{=}2 \mid \text{R}[2]1}$$
$$\boxed{2{=}r \Rightarrow (r{=}2 \Rightarrow 1{=}s) \Rightarrow s{=}1 \mid \text{W}x1}$$
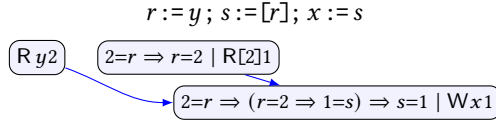
All the preconditions are now tautologies.

A.2 Merging Different labels

We combine access and fence modes into a single order:

$$\text{rlx} \longrightarrow \text{ra} \longrightarrow \text{sc} \qquad \qquad \begin{matrix} \text{acq} \\ \text{rel} \end{matrix} \mathrel{\substack{\searrow \\ \nearrow}} \text{ar}$$

Definition A.4. Define merge : $\mathcal{A} \times \mathcal{A} \to 2^{\mathcal{A}}$ as follows. If $a_0 \in \text{merge}(a_1, a_2)$, then a_1 and a_2 can coalesce, resulting in a_0. This is useful for replacing $(x := 1 \,;\, x := 2)$ by $(x := 2)$.

$$\text{merge}(\text{R}^{\mu}xv, \ \text{R}^{\nu}xv) = \{\text{R}^{\mu \sqcup \nu}xv\}$$
$$\text{merge}(\text{W}^{\mu}xv, \ \text{W}^{\nu}xw) = \{\text{W}^{\mu \sqcup \nu}xw\}$$
$$\text{merge}(\text{R}^{\text{rlx}}xv, \ \text{W}^{\mu}xv) = \{\text{W}^{\mu}xv\}$$
$$\text{merge}(\text{R}^{\mu \neq \text{rlx}}xv, \ \text{W}^{\nu}xv) = \{\text{W}^{\text{sc}}xv\}$$
$$\text{merge}(\text{F}^{\mu}, \ \text{F}^{\nu}) = \{\text{F}^{\mu \sqcup \nu}\}$$
$$\text{merge}(a, \ b) = \emptyset, \text{ otherwise}$$

(1) if $e \in E_1 \setminus E_2$ then $\lambda(e) = \lambda_1(e)$,
(2) if $e \in E_2 \setminus E_1$ then $\lambda(e) = \lambda_2(e)$,
(3) if $e \in E_1 \cap E_2$ then $\lambda(e) \in \text{merge}(\lambda_1(e), \ \lambda_2(e))$, the first has no rf,

A.3 Read-Modify-Write Operations (RMW)

RMW operations are formalized by adding a relation $\xrightarrow{\text{rmw}} \subseteq E \times E$ that relates the read of a successful RMW to the succeeding write. We say a *overlaps* b if they access the same location. Extend the definition of a pomset as follows.

(M0) rmw : $E \to E$ is a partial function capturing read-modify-write *atomicity*, such that
  (M0a) if $d \xrightarrow{\text{rmw}} e$ then $\lambda(e)$ blocks $\lambda(d)$,
  (M0b) if $d \xrightarrow{\text{rmw}} e$ then $d \leq e$,
  (M0c) if $\lambda(c)$ overlaps $\lambda(d)$ then
    (i) if $d \xrightarrow{\text{rmw}} e$ then $c \unlhd e$ implies $c \leq d$,
    (ii) if $d \xrightarrow{\text{rmw}} e$ then $d \unlhd c$ implies $e \leq c$.

Extend the definition of par, if, seq to include:

(0) rmw = (rmw$_1$ ∪ rmw$_2$),

Example A.5. This definition ensures atomicity, disallowing executions such as [Podkopaev et al. 2019, Ex. 3.2]:

$$x := 0 \,;\, s := \mathsf{FADD}^{\mathsf{rlx},\mathsf{rlx}}(x, 1) \,\|\, x := 2 \,;\, s := x$$
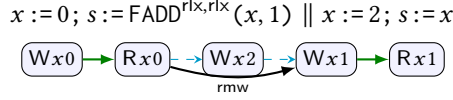
By м0c(i), since $(\mathsf{W}x2) \dashrightarrow (\mathsf{W}x1)$, it must be that $(\mathsf{W}x2) \dashrightarrow (\mathsf{R}x0)$, creating a cycle.
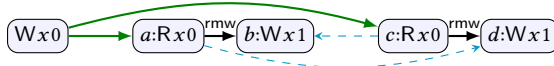
Example A.6. Two successful rmws cannot see the same write:

$$x := 0 \,;\, (\mathsf{FADD}^{\mathsf{rlx},\mathsf{rlx}}(x, 1) \,\|\, \mathsf{FADD}^{\mathsf{rlx},\mathsf{rlx}}(x, 1))$$

The order from read-to-write is required by fulfillment. Apply **??** to $a \dashrightarrow d$, we have that $a \dashrightarrow c$. Subsequently applying **??**, we have $b \dashrightarrow c$, creating a cycle.

Example A.7. By using two actions rather than one, the definition allows examples such as the following, which is allowed by Arm8 [Podkopaev et al. 2019, Ex. 3.10]:

$$r := y \,;\, z := r \,\|\, r := z \,;\, x := 0 \,;\, s := \mathsf{FADD}^{\mathsf{rlx},\mathsf{ra}}(x, 1) \,;\, y := s{+}1$$
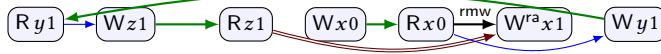
Example A.8. For rmw operations, the independent case for a read should be the same as the empty case. To see why, consider the semantics of local invariant reasoning (lir) from Def 6.2:

S4) $\tau^D(\psi) \vDash \psi[M/x] \wedge M{=}v$,

S5) $\tau^C(\psi) \vDash \psi[M/x]$,

L4) $\tau^D(\psi) \vDash v{=}r \Rightarrow \psi$,

L5) $\tau^C(\psi) \vDash (v{=}r \vee x{=}r) \Rightarrow \psi$, when $E \neq \emptyset$,

L6) $\tau^B(\psi) \vDash \psi$, when $E = \emptyset$.

Consider the relaxed variant of the cdrf example from [Lee et al. 2020], using a semantics for FADD that simply composes the rules for load and store above.

$$x := 0 \,;\, (r := \mathsf{FADD}^{\mathsf{rlx},\mathsf{rlx}}(x, 1) \,;\, \mathtt{if(!}r)\{\mathtt{if}(y)\{x := 0\}\} \quad \|$$
$$r := \mathsf{FADD}^{\mathsf{rlx},\mathsf{rlx}}(x, 1) \,;\, \mathtt{if(!}r)\{y := 1\})$$
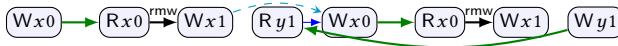
Looking at the independent transformers of the second thread and initializer, we have:
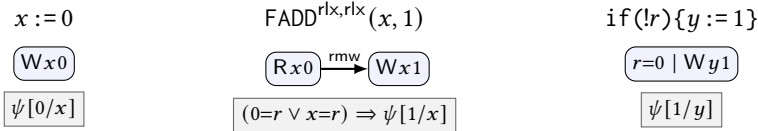
After sequencing, the precondition of $(\mathsf{W}y1)$ is a tautology: $(0{=}r \vee 0{=}r) \Rightarrow r{=}0$.

Here, local invariant reasoning is using the initializing write to x to justify the independence of the write to y. But this write is made unavailable by the first thread's successful rmw.

As a result, we disallow the use of L5 when treating the read event in an RMW.
[Todo: write out the rules.]

Example A.9. Consider the CDRF example from [Lee et al. 2020]:

$$r := \mathsf{FADD}^{\mathsf{ra},\mathsf{ra}}(x,1)\,;\, \mathrm{if}(r{=}0)\{y := 1\}$$
$$\|\;\; r := \mathsf{FADD}^{\mathsf{ra},\mathsf{ra}}(x,1)\,;\, \mathrm{if}(r{=}0)\{\mathrm{if}(y)\{x := 0\}\}$$

Example A.10. Consider this example from [Lee et al. 2020, §C]:

$$r := \mathsf{CAS}^{\mathsf{rlx},\mathsf{rlx}}(x,0,1)\,;\, \mathrm{if}(r{\leq}1)\{y := 1\}$$
$$\|\;\; r := \mathsf{CAS}^{\mathsf{rlx},\mathsf{rlx}}(x,0,2)\,;\, \mathrm{if}(r{=}0)\{\mathrm{if}(y)\{x := 0\}\}$$

A.4 Fence Operations (FENCE)

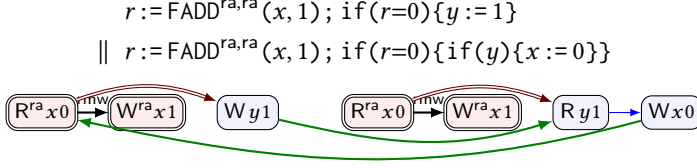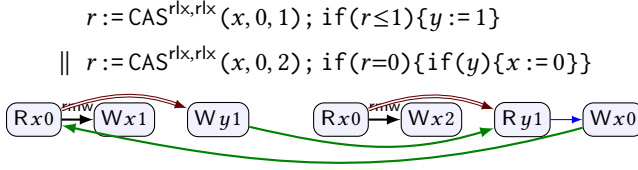Syntactic fences F^ν have corresponding actions: (F^ν). The *syntactic fence mode* ($\nu ::= \mathsf{rel} \mid \mathsf{acq} \mid \mathsf{ar}$) is either *release*, *acquire*, or *sequentially-consistent*.

Formalizing this, $Q_{\mathsf{rel}}^{\mathsf{S}x}$ substitutes for Q_{wo}^* in addition to Q_{wo}^x, as in $Q_{\mathsf{ra}}^{\mathsf{S}x}$. $Q_{\mathsf{acq}}^{\mathsf{L}x}$ requires Q_{ro}^* in addition to Q_{wo}^x, as in $Q_{\mathsf{ra}}^{\mathsf{L}x}$.

Definition A.11. Extend Def 4.4 and Def 5.4.

$$Q_{\mathsf{rel}}^{\mathsf{F}x} = Q_{\mathsf{ro}}^* \wedge Q_{\mathsf{wo}}^* \qquad\qquad Q_{\mathsf{acq}}^{\mathsf{F}x} = Q_{\mathsf{wo}}^x \wedge Q_{\mathsf{ro}}^*$$
$$[\phi/Q_{\mathsf{rel}}^{\mathsf{F}x}] = [\phi/Q_{\mathsf{wo}}^*] \qquad\qquad [\phi/Q_{\mathsf{acq}}^{\mathsf{F}x}] = [\phi/Q_{\mathsf{ro}}^*, \phi/Q_{\mathsf{wo}}^*]$$

$$Q_{\mathsf{ar}}^{\mathsf{F}x} = Q_{\mathsf{ro}}^* \wedge Q_{\mathsf{wo}}^* \wedge Q_{\mathsf{sc}}$$
$$[\phi/Q_{\mathsf{ar}}^{\mathsf{F}x}] = [\phi/Q_{\mathsf{ro}}^*, \phi/Q_{\mathsf{wo}}^*, \phi/Q_{\mathsf{sc}}]$$

If $P \in \mathit{FENCE}(\mu)$ then $(\exists X \subseteq \mathcal{X})$

(1) if $d, e \in E$ then $d = e$,
(2) $\lambda(e) = \mathsf{F}^\mu X$,
(3) $\kappa(e) \vDash \bigwedge_{x \in (\mathcal{X} \backslash X)} Q_\mu^{\mathsf{F}x}$,
(4) $\tau^D(\psi) \vDash \psi$, where $D \cap E \neq \emptyset$,
(5) $\tau^C(\psi) \vDash \bigwedge_{x \in X} \downarrow^x \wedge \psi[\mathsf{ff}/Q^{\mathsf{F}x}]$, where $C \cap E = \emptyset$.

Example A.12. Extend Ex 4.6.

$$x^{\mathsf{rel}} := M \qquad\qquad r := x^{\mathsf{acq}}$$

$$\boxed{\psi[\mathsf{ff}/Q_{\mathsf{wo}}^*]} \qquad\qquad \boxed{\psi[\mathsf{ff}/Q_{\mathsf{ro}}^*][\mathsf{ff}/Q_{\mathsf{wo}}^*]}$$

$$\boxed{Q_{\mathsf{ro}}^* \wedge Q_{\mathsf{wo}}^* \wedge M{=}v \mid \mathsf{W}^{\mathsf{ra}}xv} \qquad\qquad \boxed{Q_{\mathsf{wo}}^x \wedge Q_{\mathsf{ro}}^* \mid \mathsf{R}^{\mathsf{acq}}xv}$$

$$\boxed{\psi \wedge M{=}v} \qquad\qquad \boxed{v{=}r \Rightarrow \psi}$$

$$\mathsf{F}^{\mathsf{rel}} \qquad\qquad \mathsf{F}^{\mathsf{acq}}$$

$$\boxed{\psi[\mathsf{ff}/Q_{\mathsf{wo}}^*]} \qquad\qquad \boxed{\psi[\mathsf{ff}/Q_{\mathsf{ro}}^*][\mathsf{ff}/Q_{\mathsf{wo}}^*]}$$

$$\boxed{Q_{\mathsf{ro}}^* \wedge Q_{\mathsf{wo}}^* \mid \mathsf{F}^{\mathsf{rel}}} \qquad\qquad \boxed{Q_{\mathsf{ro}}^* \mid \mathsf{F}^{\mathsf{acq}}}$$
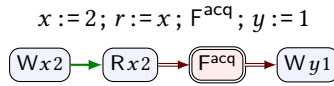
$$\boxed{\psi} \qquad\qquad \boxed{\psi}$$

$$\mathsf{F}^{\mathsf{ar}}$$

$$\boxed{\psi[\mathsf{ff}/Q_{\mathsf{ro}}^*][\mathsf{ff}/Q_{\mathsf{wo}}^*][\mathsf{ff}/Q_{\mathsf{sc}}]}$$

$$\boxed{Q_{\mathsf{ro}}^* \wedge Q_{\mathsf{wo}}^* \wedge Q_{\mathsf{sc}} \mid \mathsf{F}^{\mathsf{sc}}}$$

$$\boxed{\psi}$$

$$x^{\mathsf{ar}} := M \qquad\qquad M := x^{\mathsf{ar}}$$

$$\boxed{\psi[\mathsf{ff}/Q_{\mathsf{ro}}^*][\mathsf{ff}/Q_{\mathsf{wo}}^*][\mathsf{ff}/Q_{\mathsf{sc}}]} \qquad \boxed{\psi[\mathsf{ff}/Q_{\mathsf{ro}}^*][\mathsf{ff}/Q_{\mathsf{wo}}^*][\mathsf{ff}/Q_{\mathsf{sc}}]}$$

$$\boxed{Q_{\mathsf{ro}}^* \wedge Q_{\mathsf{wo}}^* \wedge Q_{\mathsf{sc}} \wedge M{=}v \mid \mathsf{W}^{\mathsf{ar}}xv} \qquad \boxed{Q_{\mathsf{ro}}^* \wedge Q_{\mathsf{wo}}^* \wedge Q_{\mathsf{sc}} \mid \mathsf{R}^{\mathsf{ar}}xv}$$

$$\boxed{\psi \wedge M{=}v} \qquad\qquad \boxed{v{=}r \Rightarrow \psi}$$
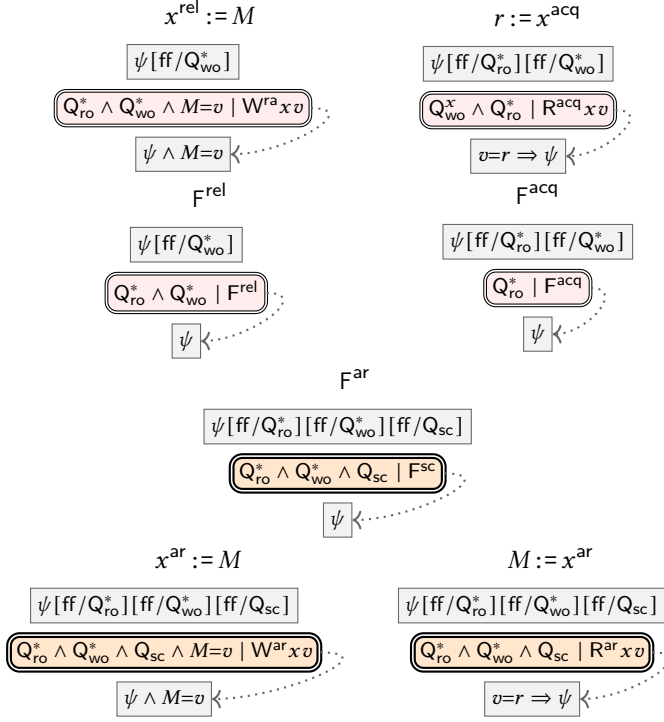
A.5 Fence Actions with Downgrading Reads (FENCE/DGR)

Example A.13. Revisiting Ex 5.6 using fences:

$$x := 2\,;\; r := x\,;\; \mathsf{F}^{\mathsf{acq}}\,;\; y := 1$$

$$\boxed{\mathsf{W}x2} \to \boxed{\mathsf{R}x2} \to \boxed{\mathsf{F}^{\mathsf{acq}}} \to \boxed{\mathsf{W}y1}$$

What we want is this:

$$\boxed{\mathsf{W}x2} \to \boxed{\mathsf{R}x2} \qquad \boxed{\mathsf{F}^{\mathsf{acq}}} \to \boxed{\mathsf{W}y1}$$

Let acquiring fence actions include a set: ($\mathsf{F}^v X$). The set is nondeterministically chosen by the semantics. Idea is to downgrade the reads in X and fence everything else.

A.6 Access Elimination

For reads, get rid of ff/Q in L6.

For writes, change the label rules of sequential composition to:

(1) if $e \in E_1 \setminus E_2$ then $\lambda(e) = \lambda_1(e)$,
(2) if $e \in E_2 \setminus E_1$ then $\lambda(e) = \lambda_2(e)$,
(3) if $e \in E_1 \cap E_2$ then $\lambda(e) \in \mathsf{merge}(\lambda_1(e),\ \lambda_2(e))$.

Definition A.14.

$$\text{merge}(\mathsf{R}^\mu xv, \ \mathsf{R}^\nu xv) = \{\mathsf{R}^{\mu \sqcup \nu} xv\}$$
$$\text{merge}(\mathsf{W}^\mu xv, \ \mathsf{W}^\nu xw) = \{\mathsf{W}^{\mu \sqcup \nu} xw\}$$
$$\text{merge}(\mathsf{F}^\mu, \ \mathsf{F}^\nu) = \{\mathsf{F}^{\mu \sqcup \nu}\}$$
$$\text{merge}(a, \ b) = \emptyset, \ \text{otherwise}$$

A.7   Extended Access Modes

We can enrich read and write actions to use fence modes. The resulting order is:

$$\mathsf{rlx} \longrightarrow \mathsf{ra} \overset{\mathsf{acq}}{\underset{\mathsf{rel}}{\rightrightarrows}} \mathsf{sc} \overset{}{\rightrightarrows} \mathsf{ar}$$

We write $\mu \sqsubseteq \nu$ for this order. Let $\mu \sqcup \nu$ denote the least upper bound of μ and ν.

Reads allow all annotations but rel. Writes allow all annotations but acq. Fences allow only the three annotations rel, acq and ar.

Definition A.15.

$$\text{merge}(\mathsf{R}^\mu xv, \ \mathsf{R}^\nu xv) = \{\mathsf{R}^{\mu \sqcup \nu} xv\}$$
$$\text{merge}(\mathsf{W}^\mu xv, \ \mathsf{W}^\nu xw) = \{\mathsf{W}^{\mu \sqcup \nu} xw\}$$
$$\text{merge}(\mathsf{F}^\mu, \ \mathsf{F}^\nu) = \{\mathsf{F}^{\mu \sqcup \nu}\}$$
$$\text{merge}(\mathsf{F}^\mu, \ \mathsf{R}^\nu xv) = \text{merge}(\mathsf{R}^\mu xv, \ \mathsf{F}^\nu) = \{\mathsf{R}^{\mu \sqcup \nu} xv\}$$
$$\text{merge}(\mathsf{F}^\mu, \ \mathsf{W}^\nu xw) = \text{merge}(\mathsf{W}^\mu xw, \ \mathsf{F}^\nu) = \{\mathsf{W}^{\mu \sqcup \nu} xw\}$$
$$\text{merge}(a, \ b) = \emptyset, \ \text{otherwise}$$

B   DISCUSSION

B.1   Closure Properties

We would like the semantics to be closed with respect to *augments* and *downsets*.

Augments include more order and stronger formulae; in examples, we typically consider pomsets that are augment-minimal. One intuitive reading of augment closure is that adding order can only cause preconditions to weaken.

Definition B.1. P_2 is an *augment* of P_1 if

(1) $E_2 = E_1$,
(2) $\lambda_2(e) = \lambda_1(e)$,
(3) $\kappa_2(e) \vDash \kappa_1(e)$,
(4) $\tau_2^D(e) \vDash \tau_1^D(e)$,
(5) if $d \leq_2 e$ then $d \leq_1 e$.

PROPOSITION B.2. *If* $P_1 \in [\![S]\!]$ *and* P_2 *augments* P_1 *then* $P_2 \in [\![S]\!]$.

Downsets include a subset of initial events, similar to *prefixes* for strings.

Definition B.3. P_2 is a *downset* of P_1 if

(1) $E_2 \subseteq E_1$,
(2) $(\forall e \in E_2) \ \lambda_2(e) = \lambda_1(e)$,
(3) $(\forall e \in E_2) \ \kappa_2(e) = \kappa_1(e)$,
(4) $(\forall e \in E_2) \ \tau_2^D(e) = \tau_1^D(e)$,

(5) $(\forall d \in E_2)\ (\forall e \in E_2)\ d \leq_2 e$ if and only if $d \leq_1 e$,

(6) $(\forall d \in E_1)\ (\forall e \in E_2)$ if $d \leq_1 e$ then $d \in E_2$.

Downset closure fails due to RRD and LIR. The key property is that the empty set transformer should behave the same as the independent transformer.

Example B.4. For RRD, Def 5.9 states:

L4) $\tau^D(\psi) \vDash v{=}r \Rightarrow \psi$,

L5) $\tau^C(\psi) \vDash (v{=}r \vee W) \Rightarrow \psi$,

L6) $\tau^B(\psi) \vDash \psi$, when $E = \emptyset$.

This semantics is not downset closed due to the lack of read-read dependencies. In both cases, for subsequent writes, L5 is the same as L6. For subsequent reads, L5 is the same as L4. Consider

$$r := x\,;\ \text{if}(!r)\{s := y\}$$
$$\boxed{\mathsf{R}\,x0} \qquad \boxed{\mathsf{R}\,y0}$$

The semantics of this program includes the singleton pomset $(\mathsf{R}\,x0)$, but not the singleton pomset $(\mathsf{R}\,y0)$. To get $(\mathsf{R}\,x0)$, we combine:

$$r := x \qquad\qquad\qquad \text{if}(!r)\{s := y\}$$
$$\boxed{\mathsf{R}\,x0} \qquad\qquad\qquad \emptyset$$

Attempting to get $(\mathsf{R}\,y0)$, we instead get:

$$r := x \qquad\qquad\qquad \text{if}(!r)\{s := y\}$$
$$\emptyset \qquad\qquad\qquad \boxed{r{=}0 \mid \mathsf{R}\,y0}$$

Since r appears only once in the program, this pomset cannot contribute to a top-level pomset.

Example B.5. For LIR, Def 6.2 states:

L4) $\tau^D(\psi) \vDash v{=}r \Rightarrow \psi$,

L5) $\tau^C(\psi) \vDash (v{=}r \vee x{=}r) \Rightarrow \psi$, when $E \neq \emptyset$,

L6) $\tau^B(\psi) \vDash \psi$, when $E = \emptyset$.

This semantics is not downset closed: The independency reasoning of L5 is only applicable for pomsets where the ignored read is present! Revisiting Ex 6.3

$$x := 0 \qquad\qquad r := x \qquad\qquad \text{if}(r{\geq}0)\{y := 1\}$$
$$\boxed{\mathsf{W}\,x0} \qquad\qquad \boxed{\mathsf{R}\,x1} \qquad\qquad \boxed{r{\geq}0 \mid \mathsf{W}\,y1}$$
$$\boxed{\psi[0/x]} \qquad\qquad \boxed{(1{=}r \vee x{=}r) \Rightarrow \psi}$$

$$x := 0\,;\ r := x\,;\ \text{if}(r{\geq}0)\{y := 1\}$$
$$\boxed{\mathsf{W}\,x0}\dashrightarrow\boxed{\mathsf{R}\,x1} \qquad \boxed{(1{=}r \vee 0{=}r) \Rightarrow r{\geq}0 \mid \mathsf{W}\,y1}$$

The precondition of $(\mathsf{W}\,y1)$ is a tautology.

Taking the empty set for the read, however, we have:

$$x := 0\,;\ r := x\,;\ \text{if}(r{\geq}0)\{y := 1\}$$
$$\boxed{\mathsf{W}\,x0} \qquad\qquad \boxed{r{\geq}0 \mid \mathsf{W}\,y1}$$

The precondition of $(\mathsf{W}\,y1)$ is not a tautology.

B.2 Comparison with Weakest Preconditions

We compare traditional transformers to the dependent-case transformers of Def 6.2; thus we consider only totally ordered executions. Because we only consider the dependent case, we drop the superscript E on τ^E throughout this section. We also assume that each register appears at most once in a program, as we did throughout §3–5.

Because of augment closure, we are not interested in isolating the *weakest* precondition. Thus we think of transformers as Hoare triples. In addition, all programs in our language are strongly normalizing, so we need not distinguish strong and weak correctness. In this setting, the Hoare triple $\{\phi\}\, S\, \{\psi\}$ holds exactly when $\phi \Rightarrow wp_S(\psi)$.

Hoare triples do not distinguish thread-local variables from shared variables. Thus, the assignment rule applies to all types of storage. The rules can be written as follows:

$$wp_{x\,:=\,M}(\psi) = \psi[M/x]$$
$$wp_{r\,:=\,M}(\psi) = \psi[M/r]$$
$$wp_{r\,:=\,x}(\psi) = x{=}r \Rightarrow \psi$$

Here we have chosen an alternative formulation for the read rule, which is equivalent the more traditional $\psi[x/r]$, as long as registers occur at most once in a program. In Def 6.2, the transformers for the dependent case are as follows:

$$\tau_{x\,:=\,M}(\psi) = \psi[M/x]$$
$$\tau_{r\,:=\,M}(\psi) = \psi[M/r]$$
$$\tau_{r\,:=\,x}(\psi) = v{=}r \Rightarrow \psi \qquad\qquad \text{where } \lambda(e) = \mathsf{R}xv$$

Only the read rule differs from the traditional one.

For programs where every register is bound and every read is fulfilled, our dependent transformers are the same as the traditional ones. In our semantics, thus, we only consider totally-ordered executions where every read could be fulfilled by prepending some writes. For example, we ignore pomsets of $x\,:=\,2\,;\, r\,:=\,x$ that read 1 for x.

For example, let S_i be defined:

$$S_1 = s\,:=\,x\,;\, x\,:=\,s{+}r$$
$$S_2 = x\,:=\,t\,;\, S_1$$
$$S_3 = t\,:=\,2\,;\, r\,:=\,5\,;\, S_2$$

The following pomset appears in the semantics of S_2. A pomset for S_3 can be derived by substituting $[2/t, 5/r]$. A pomset for S_1 can be derived by eliminating the initial write.

$$x\,:=\,t\,;\, s\,:=\,x\,;\, x\,:=\,s{+}r$$
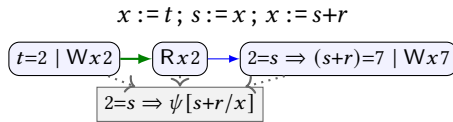
The predicate transformers are:

$$wp_{S_1}(\psi) = x{=}s \Rightarrow \psi[s{+}r/x] \qquad\qquad \tau_{S_1}(\psi) = 2{=}s \Rightarrow \psi[s{+}r/x]$$
$$wp_{S_2}(\psi) = t{=}s \Rightarrow \psi[s{+}r/x] \qquad\qquad \tau_{S_2}(\psi) = 2{=}s \Rightarrow \psi[s{+}r/x]$$
$$wp_{S_3}(\psi) = 2{=}s \Rightarrow \psi[s{+}5/x] \qquad\qquad \tau_{S_3}(\psi) = 2{=}s \Rightarrow \psi[s{+}5/x]$$

If $P \in \mathit{WRITE}(x, M, \mu)$ then $(\exists v : E \to \mathcal{V})$ $(\exists \theta : E \to \Phi)$

   S1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

   S2) $\lambda(e) = \mathsf{W}^\mu x v_e$,

   S3) $\kappa(e)$ implies $\theta_e \wedge M{=}v_e$,

   S4) $(\forall e \in E \cap D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow (\psi[M/x][\mu/{\downarrow^x}] \wedge M{=}v_e)$,

   S5) $(\forall e \in E \setminus D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow \psi[M/x][\mu/{\downarrow^x}][\mathsf{ff}/\mathsf{Q}]$.

   S6) $\tau^D(\psi)$ implies $(\nexists e \in E \mid \theta_e) \Rightarrow \psi[M/x][\mu/{\downarrow^x}][\mathsf{ff}/\mathsf{Q}]$.

If $P \in \mathit{READ}(r, x, \mu)$ then $(\exists v : E \to \mathcal{V})$ $(\exists \theta : E \to \Phi)$ $(\exists \nu \in \{\mu, \mathsf{rlx}\})$

   L1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

   L2) $\lambda(e) = \mathsf{R}^\nu x v_e$

   L3) $\kappa(e)$ implies θ_e,

   L4) $(\forall e \in E \cap D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow v_e{=}s_e \Rightarrow \psi[s_e/r]$,

   L5) $(\forall e \in E \setminus D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow {\downarrow^x_{\mu,\nu}} \wedge (\langle v_e{=}s_e \vee (\mathsf{W} \wedge x{=}s_e)\rangle \Rightarrow \psi[s_e/r][\mathsf{ff}/\mathsf{Q}])$,

   L6) $(\forall s)\ \tau^D(\psi)$ implies $(\nexists e \in E \mid \theta_e) \Rightarrow ({\downarrow^x_{\mu,\nu}} \wedge \psi[s/r][\mathsf{ff}/\mathsf{Q}])$.

Fig. 3. Simplified Quiescence Semantics w/o Address Calculation (See Def B.9 for ${\downarrow^x_{\mu,\nu}}$, $[\mu/{\downarrow^x}]$)

B.3 Coherence/Synchronization via Reordering

In §4.2, we encoded coherence and synchronized access using quiescence symbols. Building on the language with fork-join, it is possible to model these using reorderability (§3.2), rather than encoding them in the logic. With synchronization, the relationship becomes asymmetric.

To capture completion, we use a single quiescence symbol: Q.

Update actions to include access modes: ($\mathsf{W}^\mu x v$) and ($\mathsf{R}^\mu x v$). Reorderability of two sequential actions is determined, in part, by the modes of the two actions, capturing synchronization:

1st	2nd					
	$\mathsf{R}^{\mathsf{rlx}}$	R^{ra}	R^{sc}	$\mathsf{W}^{\mathsf{rlx}}$	W^{ra}	W^{sc}
$\mathsf{R}^{\mathsf{rlx}}$	✓	✓	✓	✓	✗	✗
R^{ra}	✗	✗	✗	✗	✗	✗
R^{sc}	✗	✗	✗	✗	✗	✗
$\mathsf{W}^{\mathsf{rlx}}$	✓	✓	✓	✓	✗	✗
W^{ra}	✓	✓	✓	✓	✗	✗
W^{sc}	✓	✓	✗	✓	✗	✗

It seems that fences generally do not commute, except for road-motel.

1st	2nd								
	$\mathsf{R}^{\mathsf{rlx}}$	R^{ra}	R^{sc}	$\mathsf{W}^{\mathsf{rlx}}$	W^{ra}	W^{sc}	$\mathsf{F}^{\mathsf{rel}}$	$\mathsf{F}^{\mathsf{acq}}$	F^{ar}
$\mathsf{R}^{\mathsf{rlx}}$	✓	✓	✓	✓	✗	✗	✗	✗	✗
R^{ra}	✗	✗	✗	✗	✗	✗	✗	✗	✗
R^{sc}	✗	✗	✗	✗	✗	✗	✗	✗	✗
$\mathsf{W}^{\mathsf{rlx}}$	✓	✓	✓	✓	✗	✗	✗	✓	✗
W^{ra}	✓	✓	✓	✓	✗	✗	✗	✓	✗
W^{sc}	✓	✓	✗	✓	✗	✗	✗	✓	✗
$\mathsf{F}^{\mathsf{rel}}$	✓	✓	✓	✗	✗	✗	✗	✓	✗
$\mathsf{F}^{\mathsf{acq}}$	✗	✗	✗	✗	✗	✗	✗	✗	✗
F^{ar}	✗	✗	✗	✗	✗	✗	✗	✗	✗

Definition B.6. Including coherence, *reorderability* is defined:

$$\bowtie_{\mathsf{co}} = \{(Wx, Ry) \mid x \neq y\} \cup \{(Wx, Wy) \mid x \neq y\}$$
$$\cup \{(Rx, Wy) \mid x \neq y\} \cup \{(Rx, Ry)\}$$

$$\bowtie_{\mathsf{sync}} = \{(W^\mu, R^\nu) \mid \mu \neq \mathsf{sc} \vee \nu \neq \mathsf{sc}\} \cup \{(W^\mu, W^{\mathsf{rlx}}\}$$
$$\cup \{(R^\mu, W^\nu) \mid \mu = \mathsf{rlx} \wedge \nu = \mathsf{rlx}\} \cup \{(R^{\mathsf{rlx}}, R^\nu)\}$$
$$\cup \{(F^{\mathsf{rel}}, F^{\mathsf{acq}})\} \cup \{(F^{\mathsf{rel}}, R^\nu)\} \cup \{(W^\mu, F^{\mathsf{acq}})\}$$

$$\bowtie \; = \; \bowtie_{\mathsf{sync}} \cap \bowtie_{\mathsf{co}}$$

Here is the version with generalized modes for read and write:

$$\bowtie_{\mathsf{sync}} = \{(W^\mu, R^\nu) \mid \mu \not\sqsupseteq \mathsf{sc} \vee \nu \not\sqsupseteq \mathsf{sc}\} \cup \{(W^\mu, W^{\mathsf{rlx}} \mid \mu \not\sqsupseteq \mathsf{rel}\}$$
$$\cup \{(R^\mu, W^\nu) \mid \mu = \mathsf{rlx} \wedge \nu = \mathsf{rlx}\} \cup \{(R^{\mathsf{rlx}}, R^\nu) \mid \nu \not\sqsupseteq \mathsf{acq}\}$$
$$\cup \{(F^{\mathsf{rel}}, F^{\mathsf{acq}})\} \cup \{(F^{\mathsf{rel}}, R^\nu)\} \cup \{(W^\mu, F^{\mathsf{acq}})\}$$

1st	R^{rlx}	R^{ra}	R^{acq}	R^{sc}	R^{ar}	W^{rlx}	W^{ra}	W^{rel}	W^{sc}	W^{ar}
R^{rlx}	✓	✓	✗	✓	✗	✓	✗	✗	✗	✗
R^{ra}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
R^{acq}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
R^{sc}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
R^{ar}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
W^{rlx}	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗
W^{ra}	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗
W^{rel}	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗
W^{sc}	✓	✓	✓	✗	✗	✓	✗	✗	✗	✗
W^{ar}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

(header for 2nd across columns; "2nd" spans the column group)

Definition B.7. If $P \in \mathcal{P}_1 \, ; \, \mathcal{P}_2$ then $(\exists P_1 \in \mathcal{P}_1) \, (\exists P_2 \in \mathcal{P}_2)$

1–9) as for ; in Def 3.21,
  (10) if $d \in E_1$ and $e \in E_2$ either $d \le e$ or $a \bowtie \lambda_2(e)$.

Update the read and write rules of Def 3.21 to:

S2) $\lambda(e) = W^\mu xv$,
S3) $\kappa(e) \vDash M{=}v$,
S4) $\tau^D(\psi) \vDash \psi \wedge M{=}v$,
S5) $\tau^C(\psi) \vDash \psi[\mathsf{ff}/Q]$,
L2) $\lambda(e) = R^\mu xv$,
L3) $\kappa(e) \vDash \mathsf{tt}$.
L4) $\tau^D(\psi) \vDash v{=}r \Rightarrow \psi$,
L5) $\tau^C(\psi) \vDash \psi[\mathsf{ff}/Q]$.

Example B.8. The logic of quiescence is greatly simplified. Compare the following to Ex 4.6.
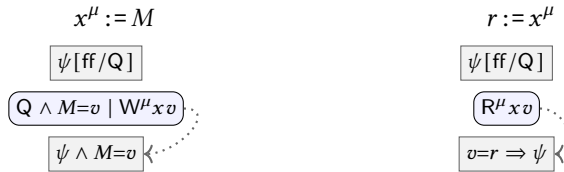

Fig 3 shows the resulting full semantics of read and write, without address calculation. Fig 4 shows the resulting full semantics with address calculation.

If $P \in \textit{WRITE}(L, M, \mu)$ then $(\exists \ell : E \to \mathcal{V})\,(\exists v : E \to \mathcal{V})\,(\exists \theta : E \to \Phi)$

S1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

S2) $\lambda(e) = \mathsf{W}^\mu[\ell_e]v_e$,

S3) $\kappa(e)$ implies $\theta_e \wedge L{=}\ell_e \wedge M{=}v_e$,

S4) $(\forall e \in E \cap D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow \psi[M/[\ell_e]][\mu/\downarrow^{[\ell_e]}] \wedge M{=}v_e \wedge L{=}\ell_e$,

S5) $(\forall e \in E \setminus D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow \psi[M/[\ell_e]][\mu/\downarrow^{[\ell_e]}][\mathsf{ff}/\mathsf{Q}]$,

S6) $(\forall k)\ \tau^D(\psi)$ implies $(\nexists e \in E \mid \theta_e) \Rightarrow (L{=}k) \Rightarrow \psi[M/[k]][\mu/\downarrow^{[k]}][\mathsf{ff}/\mathsf{Q}]$.

If $P \in \textit{READ}(r, L, \mu)$ then $(\exists \ell : E \to \mathcal{V})\,(\exists v : E \to \mathcal{V})\,(\exists \theta : E \to \Phi)\,(\exists \nu \in \{\mu, \mathsf{rlx}\})$

L1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

L2) $\lambda(e) = \mathsf{R}^\nu[\ell_e]v_e$,

L3) $\kappa(e)$ implies $\theta_e \wedge L{=}\ell_e$,

L4) $(\forall e \in E \cap D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow (L{=}\ell_e \Rightarrow v_e{=}s_e) \Rightarrow \psi[s_e/r]$,

L5) $(\forall e \in E \setminus D)\ \tau^D(\psi)$ implies $\theta_e \Rightarrow \downarrow^{[\ell_e]}_{\mu,\nu} \wedge \big(\langle (L{=}\ell_e \Rightarrow v_e{=}s_e) \vee (\mathsf{W} \wedge (L{=}\ell_e \Rightarrow [\ell_e]{=}s_e)) \rangle \Rightarrow \psi[s_e/r][\mathsf{ff}/\mathsf{Q}]\big)$,

L6) $(\forall k)(\forall s)\ \tau^D(\psi)$ implies $(\nexists e \in E \mid \theta_e) \Rightarrow (L{=}k) \Rightarrow (\downarrow^{[k]}_{\mu,\nu} \wedge \psi[s/r][\mathsf{ff}/\mathsf{Q}])$.

Fig. 4. Simplified Quiescence Semantics with Address Calculation (See Def B.9 for $\downarrow^x_{\mu,\nu}$, $[\mu/\downarrow^x]$)

Definition B.9. Let substitution $[\mu/\downarrow^x]$ be defined:

$$[\mu/\downarrow^x] = \begin{cases} [\mathsf{tt}/\downarrow^x] & \text{if } \mu = \mathsf{rlx} \\ [\mathsf{ff}/\downarrow^*] & \text{otherwise} \end{cases}$$

Let formula $\downarrow^x_{\mu,\nu}$ be defined:

$$\downarrow^x_{\mu,\nu} = \begin{cases} \mathsf{tt} & \text{if } \mu = \mathsf{rlx} \text{ or } \mu = \nu \\ \downarrow^x & \text{otherwise} \end{cases}$$

B.4 Substitutions

Recall the load rules from §6.1:

L4) $\tau^D(\psi) \vDash v{=}r \Rightarrow \psi$,

L5) $\tau^C(\psi) \vDash (v{=}r \vee x{=}r) \Rightarrow \psi$, when $E \neq \emptyset$,

L6) $\tau^B(\psi) \vDash \psi$, when $E = \emptyset$.

It is also possible to collapse x and r when doing a load:

L4) $\tau^D(\psi) \vDash v{=}r \Rightarrow \psi[r/x]$,

L5) $\tau^C(\psi) \vDash (v{=}r \vee x{=}r) \Rightarrow \psi[r/x]$, when $E \neq \emptyset$.

L6) $\tau^B(\psi) \vDash \psi[r/x]$, when $E = \emptyset$.

Perhaps surprisingly, these two semantics are incomparable. Consider the following:

$$\mathtt{if}(r \wedge s \text{ even})\{y := 1\};\ \mathtt{if}(r \wedge s)\{z := 1\}$$

$$\boxed{r \wedge s \text{ even} \mid \mathsf{W}\,y1}$$

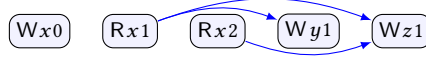$$\boxed{r \wedge s \mid \mathsf{W}\,z1}$$

Prepending $(s := x)$, we get the same result regardless of whether we substitute $[s/x]$, since x does not occur in either precondition. Here we show the independent case:

$$s := x \,;\, \text{if}(r \wedge s \text{ even})\{y := 1\} \,;\, \text{if}(r \wedge s)\{z := 1\}$$

$$\boxed{(2{=}s \vee x{=}s) \Rightarrow (r \wedge s \text{ even}) \mid \mathsf{W}\,y1}$$

$$\boxed{\mathsf{R}\,x2} \qquad \boxed{(2{=}s \vee x{=}s) \Rightarrow (r \wedge s) \mid \mathsf{W}\,z1}$$

Prepending $(r := x)$, we now get different results since the preconditions mention x. Without substitution:

$$r := x \,;\, s := x \,;\, \text{if}(r \wedge s \text{ even})\{y := 1\} \,;\, \text{if}(r \wedge s)\{z := 1\}$$

$$\boxed{\mathsf{R}\,x1} \longrightarrow \boxed{1{=}r \Rightarrow (2{=}s \vee x{=}s) \Rightarrow (r \wedge s \text{ even}) \mid \mathsf{W}\,y1}$$

$$\boxed{\mathsf{R}\,x2} \qquad \boxed{1{=}r \Rightarrow (2{=}s \vee x{=}s) \Rightarrow (r \wedge s) \mid \mathsf{W}\,z1}$$
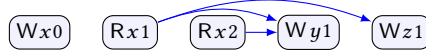
Prepending $(x := 0)$, which substitutes $[0/x]$, the precondition of $(\mathsf{W}\,y1)$ becomes $(1{=}r \Rightarrow (2{=}s \vee 0{=}s) \Rightarrow (r \wedge s \text{ even}))$, which is a tautology, whereas the precondition of $\mathsf{W}\,z1$ becomes $(1{=}r \Rightarrow (2{=}s \vee 0{=}s) \Rightarrow (r \wedge s))$, which is not. In order to be top-level, $\mathsf{W}\,z1$ must depend on $\mathsf{R}\,x2$; in this case the precondition becomes $(1{=}r \Rightarrow 2{=}s \Rightarrow (r \wedge s))$, which is a tautology.

$$\boxed{\mathsf{W}\,x0} \qquad \boxed{\mathsf{R}\,x1} \qquad \boxed{\mathsf{R}\,x2} \qquad \boxed{\mathsf{W}\,y1} \qquad \boxed{\mathsf{W}\,z1}$$

The situation reverses with the substitution $[r/x]$:

$$r := x \,;\, s := x \,;\, \text{if}(r \wedge s \text{ even})\{y := 1\} \,;\, \text{if}(r \wedge s)\{z := 1\}$$

$$\boxed{\mathsf{R}\,x1} \longrightarrow \boxed{1{=}r \Rightarrow (2{=}s \vee r{=}s) \Rightarrow (r \wedge s \text{ even}) \mid \mathsf{W}\,y1}$$

$$\boxed{\mathsf{R}\,x2} \qquad \boxed{1{=}r \Rightarrow (2{=}s \vee r{=}s) \Rightarrow (r \wedge s) \mid \mathsf{W}\,z1}$$

Prepending $(x := 0)$:

$$\boxed{\mathsf{W}\,x0} \qquad \boxed{\mathsf{R}\,x1} \qquad \boxed{\mathsf{R}\,x2} \longrightarrow \boxed{\mathsf{W}\,y1} \qquad \boxed{\mathsf{W}\,z1}$$

The dependency has changed from $(\mathsf{R}\,x2) \longrightarrow (\mathsf{W}\,z1)$ to $(\mathsf{R}\,x2) \longrightarrow (\mathsf{W}\,y1)$. The resulting sets of pomsets are incomparable.

Thinking in terms of hardware, the difference is whether reads update the cache, thus clobbering preceding writes. With $[r/x]$, reads clobber the cache, whereas without the substitution, they do not. Since most caches work this way, the model with $[r/x]$ is likely preferred for modeling hardware. In a software model, however, we see no reason to prefer one of these over the other.

C DIFFERENCES WITH OOPSLA

C.1 Substitution

[Jagadeesan et al. 2020] uses substitution rather than Skolemizing. Indeed our use of Skolemization is motivated by disjunction closure for predicate transformers, which do not appear in [Jagadeesan et al. 2020]. In §6.1 on local invariant reasoning (LIR), we gave the semantics of load for nonempty pomsets as:

L4) $\tau^D(\psi) \vDash v{=}r \Rightarrow \psi$,
L5) $\tau^C(\psi) \vDash (v{=}r \vee x{=}r) \Rightarrow \psi$.

In [Jagadeesan et al. 2020], the definition is roughly as follows:

L4) $\tau^D(\psi) \vDash \psi[v/r][v/x]$,
L5) $\tau^C(\psi) \vDash \psi[v/r][v/x] \wedge \psi[x/r]$.

The use of conjunction in L5 causes disjunction closure to fail because the predicate transformer $\tau(\psi) = \psi' \wedge \psi''$ does not distribute through disjunction, even assuming that the prime operations do:[1] $\tau(\psi_1 \vee \psi_2) = (\psi'_1 \vee \psi'_2) \wedge (\psi''_1 \vee \psi''_2) \neq (\psi'_1 \wedge \psi''_1) \vee (\psi'_2 \wedge \psi''_2) = \tau(\psi_1) \vee \tau(\psi_2)$. See also Ex 3.26.

The substitutions collapse x and r, allowing local invariant reasoning, as in §6.1. Without Skolemizing it is necessary to substitute $[x/r]$, since the reverse substitution $[r/x]$ is useless when r is bound—compare with §B.4. As discussed below (§C.2), including this substitution affects the interaction of LIR and downset closure.

Removing the substitution of $[x/r]$ in the independent case has a technical advantage: we no longer require *extended* expressions (which include memory references), since substitutions no longer introduce memory references.

The substitution $[x/r]$ does not work with Skolemization, even for the dependent case, since we lose the unique marker for each read. In effect, this forces the reads to the same values. To be concrete, the candidate definition would modify L4 to be:

L4) $\tau^D(\psi) \vDash v{=}x \Rightarrow \psi[x/r]$.

Using this definition, consider the following:

$$r := x \,;\, s := x \,;\, \texttt{if}(r{<}s)\{y := 1\}$$
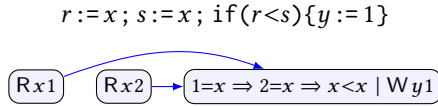
Although the execution seems reasonable, the precondition on the write is not a tautology.

C.2 Downset closure

[Jagadeesan et al. 2020] enforces downset closure in the prefixing rule. Even without this, downset closure would be different for the two semantics, due to the use of substitution in [Jagadeesan et al. 2020]. Consider the final pomset of Ex B.5, under the semantics of this paper, which elides the middle read event:

$$x := 0 \,;\, r := x \,;\, \texttt{if}(r{\geq}0)\{y := 1\}$$

In [Jagadeesan et al. 2020], the substitution $[x/r]$ is performed by the middle read regardless of whether it is included in the pomset, with the subsequent substitution of $[0/x]$ by the preceding write, we have $[x/r][0/x]$, which is $[0/r][0/x]$, resulting in:

C.3 Consistency

[Jagadeesan et al. 2020] imposes *consistency*, which requires that for every pomset P, $\bigwedge_e \kappa(e)$ is satisfiable. Associativity requires that we allow pomsets with inconsistent preconditions. Consider a variant of Ex 6.7 from §6.3.

Associating left and right, we have:

---

[1] $(\psi_1 \vee \psi_2)' = (\psi'_1 \vee \psi'_2)$ and $(\psi_1 \vee \psi_2)'' = (\psi''_1 \vee \psi''_2)$.

Associating into the middle, instead, we require:

$$\texttt{if}(M)\{x:=1\} \qquad \texttt{if}(!M)\{x:=1\};\ \texttt{if}(M)\{y:=1\} \qquad \texttt{if}(!M)\{y:=1\}$$

$$\boxed{M \mid \mathsf{W}x1} \qquad\qquad \boxed{\neg M \mid \mathsf{W}x1}\quad\boxed{M \mid \mathsf{W}y1} \qquad\qquad \boxed{\neg M \mid \mathsf{W}y1}$$

Joining left and right, we have:

$$\texttt{if}(M)\{x:=1\};\ \texttt{if}(!M)\{x:=1\};\ \texttt{if}(M)\{y:=1\};\ \texttt{if}(!M)\{y:=1\}$$

$$\boxed{\mathsf{W}x1}\quad\boxed{\mathsf{W}y1}$$

C.4 Causal Strengthening

Causal Strengthening [Jagadeesan et al. 2020] imposes *causal strengthening*, which requires for every pomset P, if $d \le e$ then $\kappa(e) \vDash \kappa(d)$. Associativity requires that we allow pomsets without causal strengthening. Consider the following.

$$\texttt{if}(M)\{r:=x\} \qquad\qquad y:=r \qquad\qquad \texttt{if}(!M)\{s:=x\}$$

$$\boxed{M \mid \mathsf{R}x1} \qquad\qquad \boxed{r{=}1 \mid \mathsf{W}y1} \qquad\qquad \boxed{\neg M \mid \mathsf{R}x1}$$

Associating left, with causal strengthening:

$$\texttt{if}(M)\{r:=x\};\ y:=r \qquad\qquad \texttt{if}(!M)\{s:=x\}$$

$$\boxed{M \mid \mathsf{R}x1}\!\rightarrow\!\boxed{M \mid \mathsf{W}y1} \qquad\qquad \boxed{\neg M \mid \mathsf{R}x1}$$

Finally, merging:

$$\texttt{if}(M)\{r:=x\};\ y:=r;\ \texttt{if}(!M)\{s:=x\}$$

$$\boxed{\mathsf{R}x1}\!\rightarrow\!\boxed{M \mid \mathsf{W}y1}$$

Instead, associating right:

$$\texttt{if}(M)\{r:=x\} \qquad\qquad y:=r;\ \texttt{if}(!M)\{s:=x\}$$

$$\boxed{M \mid \mathsf{R}x1} \qquad\qquad \boxed{r{=}1 \mid \mathsf{W}y1}\quad\boxed{\neg M \mid \mathsf{R}x1}$$

Merging:

$$\texttt{if}(M)\{r:=x\};\ y:=r;\ \texttt{if}(!M)\{s:=x\}$$

$$\boxed{\mathsf{R}x1}\!\rightarrow\!\boxed{\mathsf{W}y1}$$

With causal strengthening, the precondition of $\mathsf{W}y1$ depends upon how we associate. This is not an issue in [Jagadeesan et al. 2020], which always associates to the right.

One use of causal strengthening is to ensure that address dependencies do not introduce thin air reads. Associating to the right, the intermediate state of Ex A.3 is:

$$s:=[r];\ x:=s$$

$$\boxed{r{=}2 \mid \mathsf{R}[2]1}\!\longrightarrow\!\boxed{(r{=}2 \Rightarrow 1{=}s) \Rightarrow s{=}1 \mid \mathsf{W}x1}$$

In [Jagadeesan et al. 2020], we have, instead:

$$s:=[r];\ x:=s$$

$$\boxed{r{=}2 \mid \mathsf{R}[2]1}\!\longrightarrow\!\boxed{r{=}2 \wedge [2]{=}1 \mid \mathsf{W}x1}$$

Without causal strengthening, the precondition of $(\mathsf{W}x1)$ would be simply $[2]{=}1$. The treatment in this paper, using implication rather than conjunction, is more precise.

C.5 Parallel Composition

In [Jagadeesan et al. 2020, §2.4], parallel composition is defined allowing coalescing of events. Here we have forbidden coalescing. This difference appears to be arbitrary. In [Jagadeesan et al. 2020], however, there is a mistake in the handling of termination actions. The predicates should be joined using $\wedge$, not $\vee$.

C.6 Read-Modify-Write Actions

In [Jagadeesan et al. 2020], the atomicity axioms m0c erroneously applies only to overlapping writes, not overlapping reads. The difficulty can be seen in Ex A.6.

[Jagadeesan et al. 2020] does not specify the calculation of dependency for rmws, as discussed in Ex A.8.

C.7 Downgrading Internal Acquiring Reads

Shortly after publication, Podkopaev [2020] noticed a shortcoming of the implementation on Arm8 in [Jagadeesan et al. 2020, §7]. The proof given there assumes that all internal reads can be dropped. However, this is not the case for acquiring reds. For example, [Jagadeesan et al. 2020] disallows the following execution, which is allowed by Arm8 and tso.

$$x := 2 \,;\, r := x^{\mathsf{ra}} \,;\, s := y \parallel y := 2 \,;\, x^{\mathsf{ra}} := 1$$
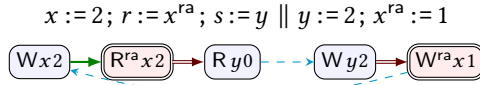
The solution we have adopted is to allow an acquiring read to be downgraded to a relaxed read when it is preceded (sequentially) by a relaxed write that could fulfill it. This solution allows executions that are not allowed under Arm8 since we do not insist that the local relaxed write is actually read from. This may seem counterintuitive, but we don't see a local way to be more precise.

As a result, we use a different proof strategy for Arm8 implementation, which does not rely on read elimination. The proof idea uses a recent alternative characterization of Arm8 [Alglave 2020; Arm Limited 2020].

C.8 Redundant Read Elimination

Contrary to the claim, redundant read elimination fails for [Jagadeesan et al. 2020]. We discussed redundant read elimination in §6.2. Consider JMM Causality Test Case 2, which we discussed there.

$$r := x \,;\, s := x \,;\, \mathtt{if}(r{=}s)\{y := 1\} \parallel x := y$$
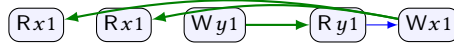
Under the semantics of [Jagadeesan et al. 2020], we have

$$r := x \,;\, s := x \,;\, \mathtt{if}(r{=}s)\{y := 1\}$$

The precondition of ($\mathsf{W}y1$) is *not* a tautology, and therefore redundant read elimination fails. (It is a tautology in $r := x \,;\, s := r \,;\, \mathtt{if}(r{=}s)\{y := 1\}$.) In [Jagadeesan et al. 2020, §3.1], we incorrectly stated that the precondition of ($\mathsf{W}y1$) was $1{=}1 \wedge x{=}x$.

D MORE STUFF

D.1 A Note on Mixed-Mode Data Races

In preparing this paper, we came across the following example, which appears to invalidate Theorem 4.1 of [Dongol et al. 2019].

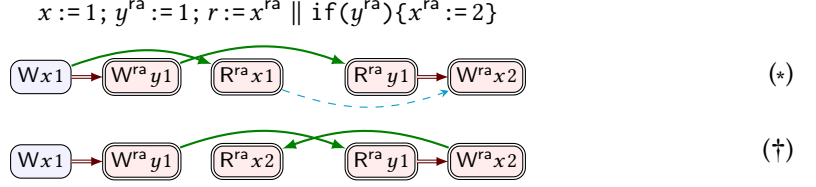$$x := 1; \, y^{\mathsf{ra}} := 1; \, r := x^{\mathsf{ra}} \; \| \; \mathtt{if}(y^{\mathsf{ra}})\{x^{\mathsf{ra}} := 2\}$$

(∗)

(†)

The program is data-race free. The two executions shown are the only top-level executions that include $(\mathsf{W}^{\mathsf{ra}}x2)$.

Theorem 4.1 of [Dongol et al. 2019] is stated by extending execution sequences. In the terminology of [Dongol et al. 2019], a read is *L-weak* if it is sequentially stale. Let $\rho = (\mathsf{W}x1)(\mathsf{W}^{\mathsf{ra}}y1)(\mathsf{R}^{\mathsf{ra}}y1)(\mathsf{W}^{\mathsf{ra}}x2)$ be a sequence and $\alpha = (\mathsf{R}^{\mathsf{ra}}x1)$. ρ is *L*-sequential and α is *L*-weak in $\rho\alpha$. But there is no execution of this program that includes a data race, contradicting the theorem. The error seems to be in Lemma A.4 of [Dongol et al. 2019], which states that if α is *L*-weak after an *L*-sequential ρ, then α must be in a data race. That is clearly false here, since $(\mathsf{R}^{\mathsf{ra}}x1)$ is stale, but the program is data race free.

In proving the SC-LDRF result in [Jagadeesan et al. 2020, §8], we noted that our proof technique is more robust than that of [Dongol et al. 2019], because it limits the prefixes that must be considered. In (∗), the induction hypothesis requires that we add $(\mathsf{R}^{\mathsf{ra}}x1)$ before $(\mathsf{W}^{\mathsf{ra}}x2)$ since $(\mathsf{R}^{\mathsf{ra}}x1) \dashrightarrow (\mathsf{W}^{\mathsf{ra}}x2)$. In particular,

is not a downset of (∗), because $(\mathsf{R}^{\mathsf{ra}}x1) \dashrightarrow (\mathsf{W}^{\mathsf{ra}}x2)$. As we noted in [Jagadeesan et al. 2020, §8], this affects the inductive order in which we move across pomsets, but does not affect the set of pomsets that are considered. In particular,

is a downset of (∗).

D.2 Downgraded Reads

We allow downgrades in executions that are not be allowed by Arm8.

$$x := 2; \, r := x^{\mathsf{ra}}; \, y := 1 \; \| \; y := 2; \, x^{\mathsf{ra}} := 1 \; \| \; x := 3$$
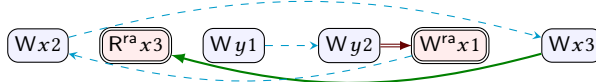
Arm8 disallows this because the acquiring read is fulfilled by an external write.

$$x := z; \, r := x^{\mathsf{ra}}; \, y := 1 \; \| \; z := y$$
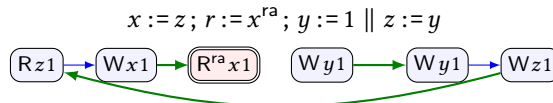
Arm8 disallows this because data and control dependencies change acquiring read is fulfilled by an external write.

D.3 If Closure and Address Dependencies

An optimization (p/q are registers):

$$r := [p]; \, s := [q]$$

vs

$$r := [p]; \, \texttt{if} \, (p{=}q)\{s := r\} \, \texttt{else} \, \{s := [q]\}$$

$$r := \texttt{new}; \, [r] := 42; \, s := [r]; \, x := r \parallel r := x; \, [r] := 7$$

If closure is at odds with Java Final field semantics.
Do sequencing and if commute?

D.4 About Arm

Hypothesis: gcb cannot contradict (poloc minus RxR).

D.5 Using Independency for Coherence

It is also possible to use independency only to capture coherence, but the results are less interesting.

In the logic, we remove the symbols Q_{wo}^x and Q_{ro}^x. Previously, we had given the semantics of ra access using Q_{wo}^* and Q_{ro}^*, which were encoded using Q_{wo}^x and Q_{ro}^x. With these gone, we introduce the quiescence symbol Q^* and Q_{acq}. Thus, the only quiescence symbols required are Q^*, Q_{acq} and Q_{sc}. Fig 5 shows the difference with the semantics of §4.2.

Definition D.1. Let formulae Q_μ^S and Q_μ^L be defined:

$$Q_{rlx}^S = Q_{acq} \qquad\qquad\qquad Q_{rlx}^L = Q_{acq}$$
$$Q_{ra}^S = Q_{acq} \wedge Q^* \qquad\qquad Q_{ra}^L = Q_{acq}$$
$$Q_{sc}^S = Q_{acq} \wedge Q^* \wedge Q_{sc} \qquad Q_{sc}^L = Q_{acq} \wedge Q_{sc}$$

Let substitutions $[\phi/Q_\mu^S]$ and $[\phi/Q_\mu^L]$ be defined:

$$[\phi/Q_{rlx}^S] = [\phi/Q^*] \qquad\qquad [\phi/Q_{rlx}^L] = [\phi/Q^*]$$
$$[\phi/Q_{ra}^S] = [\phi/Q^*] \qquad\qquad [\phi/Q_{ra}^L] = [\phi/Q^*, \phi/Q_{acq}]$$
$$[\phi/Q_{sc}^S] = [\phi/Q^*, \phi/Q_{sc}] \qquad [\phi/Q_{sc}^L] = [\phi/Q^*, \phi/Q_{acq}, \phi/Q_{sc}]$$

Definition D.2. Update Def 3.21 to:

S3) $\kappa(e) \vDash Q_\mu^S \wedge M{=}v$,
L3) $\kappa(e) \vDash Q_\mu^L$,
S4) $\tau^D(\psi) \vDash \psi \wedge M{=}v$,
S5) $\tau^C(\psi) \vDash \psi[\mathsf{ff}/Q_\mu^S]$,
L4) $\tau^D(\psi) \vDash v{=}r \Rightarrow \psi$,
L5) $\tau^C(\psi) \vDash \psi[\mathsf{ff}/Q_\mu^L]$.

The most interesting examples in Fig 5b concern ra access. Every independent transformer substitutes $[\mathsf{ff}/Q^*]$. Q^* is a precondition for any releasing write e, ensuring that all preceding events must are ordered before e. Conversely, Q_{acq} is a precondition of every event. The independent transformer for any acquiring read e substitutes $[\mathsf{ff}/Q_{acq}]$, ensuring that all following events must be ordered after e.

$$x := M$$

$$\psi[\mathsf{ff}/\mathsf{Q}^x_{\mathsf{wo}}]$$

$$Q^x_{\mathsf{ro}} \wedge Q^x_{\mathsf{wo}} \wedge M{=}v \mid \mathsf{W}xv$$

$$\psi \wedge M{=}v$$

$$r := x$$

$$\psi[\mathsf{ff}/\mathsf{Q}^x_{\mathsf{ro}}]$$

$$Q^x_{\mathsf{wo}} \mid \mathsf{R}xv$$

$$v{=}r \Rightarrow \psi$$

$$x^{\mathsf{ra}} := M$$

$$\psi[\mathsf{ff}/\mathsf{Q}^x_{\mathsf{wo}}]$$

$$Q^*_{\mathsf{ro}} \wedge Q^*_{\mathsf{wo}} \wedge M{=}v \mid \mathsf{W}^{\mathsf{ra}}xv$$

$$\psi \wedge M{=}v$$

$$r := x^{\mathsf{ra}}$$

$$\psi[\mathsf{ff}/\mathsf{Q}^*_{\mathsf{ro}}][\mathsf{ff}/\mathsf{Q}^*_{\mathsf{wo}}]$$

$$Q^x_{\mathsf{wo}} \mid \mathsf{R}^{\mathsf{ra}}xv$$

$$v{=}r \Rightarrow \psi$$

$$x^{\mathsf{sc}} := M$$

$$\psi[\mathsf{ff}/\mathsf{Q}^x_{\mathsf{wo}}][\mathsf{ff}/\mathsf{Q}_{\mathsf{sc}}]$$

$$Q^*_{\mathsf{ro}} \wedge Q^*_{\mathsf{wo}} \wedge Q_{\mathsf{sc}} \wedge M{=}v \mid \mathsf{W}^{\mathsf{sc}}xv$$

$$\psi \wedge M{=}v$$

$$r := x^{\mathsf{sc}}$$

$$\psi[\mathsf{ff}/\mathsf{Q}^*_{\mathsf{ro}}][\mathsf{ff}/\mathsf{Q}^*_{\mathsf{wo}}][\mathsf{ff}/\mathsf{Q}_{\mathsf{sc}}]$$

$$Q^x_{\mathsf{wo}} \wedge Q_{\mathsf{sc}} \mid \mathsf{R}^{\mathsf{sc}}xv$$

$$v{=}r \Rightarrow \psi$$

(a) Quiescence Examples (§4.2)

$$x := M$$

$$\psi[\mathsf{ff}/\mathsf{Q}^*]$$

$$Q_{\mathsf{acq}} \wedge M{=}v \mid \mathsf{W}xv$$

$$\psi \wedge M{=}v$$

$$r := x$$

$$\psi[\mathsf{ff}/\mathsf{Q}^*]$$

$$Q_{\mathsf{acq}} \mid \mathsf{R}xv$$

$$v{=}r \Rightarrow \psi$$

$$x^{\mathsf{ra}} := M$$

$$\psi[\mathsf{ff}/\mathsf{Q}^*]$$

$$Q_{\mathsf{acq}} \wedge Q^* \wedge M{=}v \mid \mathsf{W}^{\mathsf{ra}}xv$$

$$\psi \wedge M{=}v$$

$$r := x^{\mathsf{ra}}$$

$$\psi[\mathsf{ff}/\mathsf{Q}^*][\mathsf{ff}/\mathsf{Q}_{\mathsf{acq}}]$$

$$Q_{\mathsf{acq}} \mid \mathsf{R}^{\mathsf{ra}}xv$$

$$v{=}r \Rightarrow \psi$$

$$x^{\mathsf{sc}} := M$$

$$\psi[\mathsf{ff}/\mathsf{Q}^*][\mathsf{ff}/\mathsf{Q}_{\mathsf{sc}}]$$

$$Q_{\mathsf{acq}} \wedge Q^* \wedge Q_{\mathsf{sc}} \wedge M{=}v \mid \mathsf{W}^{\mathsf{sc}}xv$$

$$\psi \wedge M{=}v$$

$$r := x^{\mathsf{sc}}$$

$$\psi[\mathsf{ff}/\mathsf{Q}^*][\mathsf{ff}/\mathsf{Q}_{\mathsf{acq}}][\mathsf{ff}/\mathsf{Q}_{\mathsf{sc}}]$$

$$Q_{\mathsf{acq}} \wedge Q_{\mathsf{sc}} \mid \mathsf{R}^{\mathsf{sc}}xv$$

$$v{=}r \Rightarrow \psi$$
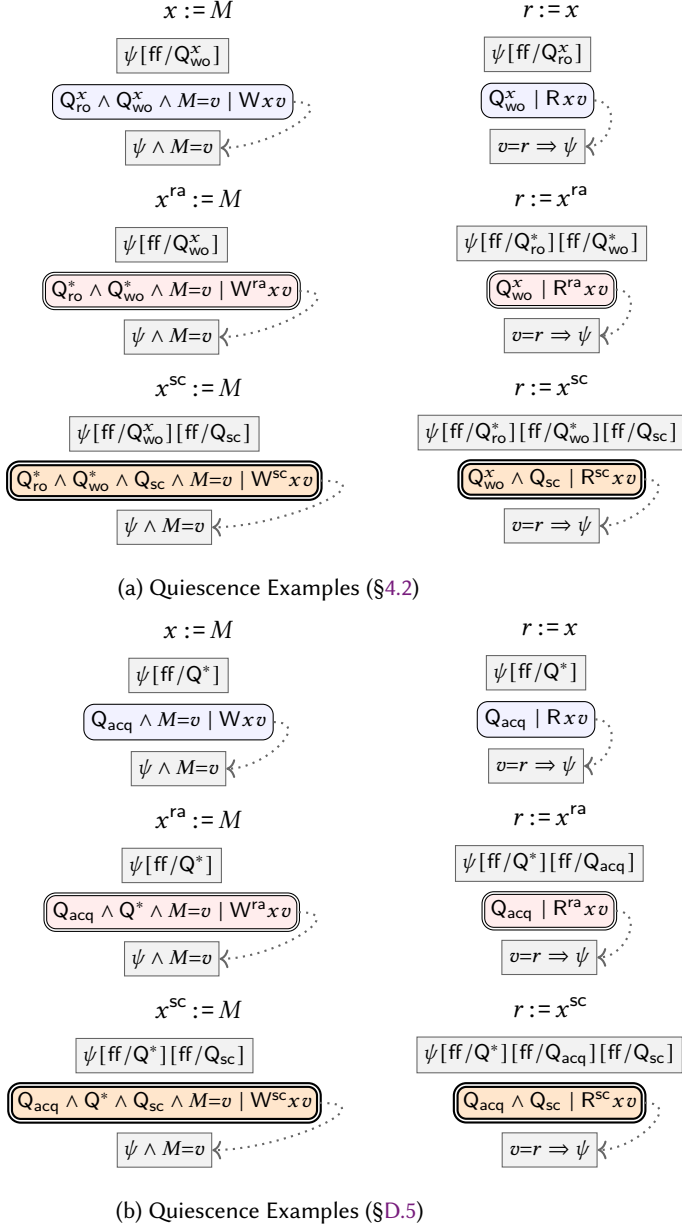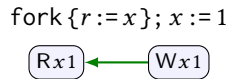
(b) Quiescence Examples (§D.5)

Fig. 5. Quiescence Examples for Coherence

Item 10 of Def B.7 ensures coherence. This definition is incompatible with asynchronous fork parallelism of Def ??, where we expect executions such as:

$$\mathsf{fork}\,\{r := x\};\, x := 1$$

$$\mathsf{R}x1 \longleftarrow \mathsf{W}x1$$

Item 10 would require $(Rx1) \dashrightarrow (Wx1)$, forbidding this.