Sequential Composition for Relaxed Memory: Pomsets with Predicate Transformers

ANONYMOUS AUTHOR(S)

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

ACM Reference Format:
Anonymous Author(s). 2021. Sequential Composition for Relaxed Memory: Pomsets with Predicate Transformers. *Proc. ACM Program. Lang.* 0, OOPSLA, Article 0 (October 2021), 14 pages.

1 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.

1.1 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 $\mathcal{X}$ 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: $M[N/x] = M$.

2475-1421/2021/10-ART0
https://doi.org/

We model the following language.

$$\mu \;::=\; \mathsf{rlx} \mid \mathsf{ra} \mid \mathsf{sc} \qquad\qquad \nu \;::=\; \mathsf{acq} \mid \mathsf{rel} \mid \mathsf{ar}$$

$$S \;::=\; r\,{:=}\,M \mid r\,{:=}\,[L]^{\mu} \mid [L]^{\mu}\,{:=}\,M \mid \mathsf{F}^{\nu} \mid \mathtt{skip} \mid S_1\,;\,S_2 \mid \mathtt{if}(M)\{S_1\}\mathtt{else}\{S_2\} \mid S_1 \mathbin{]\!|} S_2$$

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*.

Fence modes, ν, are acquire (acq), release (rel), and acquire-release (ar).

Commands, aka *statements*, S, include memory accesses at a given mode, as well as the usual structural constructs. Following [Ferreira et al. 1996], $]\!|$ denotes parallel composition, preserving thread state on the left 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 tt, ff and the equalities $(M{=}N)$ and $(x{=}M)$,
- formulae are closed under $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and substitutions $[M/r]$, $[M/x]$,
- there is a relation $\models$ between formulae, capturing entailment,
- $\models$ has the expected semantics for $=$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and substitutions $[M/r]$, $[M/x]$,
- there are three binary relations over $\mathcal{A} \times \mathcal{A}$: *matches*, *blocks*, and *delays*,
- there are two subsets of $\mathcal{A}$, distinguishing *read* and *release* actions.

Logical formulae include equations over registers, such as $(r{=}s{+}1)$. For use in §??, 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 tt $\models \phi$. We say ϕ is *unsatisfiable* if $\phi \models$ ff.

Throughout §1–?? we additionally require that

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

In §??, 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: $S[N/s_e] = S$.

1.2 Actions in This Paper

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

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

We use shorthand when referring to actions. 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}). Let $\sqsubseteq$ be the least order over access and fence modes such that $\mathsf{rlx} \sqsubseteq \mathsf{ra} \sqsubseteq \mathsf{sc}$ and $\mathsf{rel} \sqsubseteq \mathsf{ar}$ and $\mathsf{acq} \sqsubseteq \mathsf{ar}$. We write $(\mathsf{W}^{\sqsupseteq\mathsf{ra}})$ to stand for either $(\mathsf{W}^{\mathsf{ra}})$ or $(\mathsf{W}^{\mathsf{sc}})$, and similarly for the other actions and modes.

Definition 1.1. Actions (R) are *read* actions. Actions $(\mathsf{W}^{\sqsupseteq\mathsf{ra}})$ and $(\mathsf{F}^{\sqsupseteq\mathsf{rel}})$ are *release* actions.
We say *a matches b* if $a = (\mathsf{W}xv)$ and $b = (\mathsf{R}xv)$.
We say *a blocks b* if $a = (\mathsf{W}x)$ and $b = (\mathsf{R}x)$, regardless of value.
We say *a delays b* if $a \bowtie_{\mathsf{sc}} b$ or $a \bowtie_{\mathsf{co}} b$ or $a \ltimes_{\mathsf{sync}} b$. Let $\bowtie_{\mathsf{sc}} = \{(\mathsf{A}^{\mathsf{sc}}, \mathsf{A}^{\mathsf{sc}})\}$. Let $\bowtie_{\mathsf{co}} = \{(\mathsf{W}x, \mathsf{W}x),\ (\mathsf{R}x, \mathsf{W}x),\ (\mathsf{W}x, \mathsf{R}x)\}$. Let $\ltimes_{\mathsf{sync}} = \{(a, \mathsf{W}^{\sqsupseteq\mathsf{ra}}),\ (a, \mathsf{F}^{\sqsupseteq\mathsf{rel}}),\ (\mathsf{R}, \mathsf{F}^{\sqsupseteq\mathsf{acq}}),\ (\mathsf{R}x, \mathsf{R}^{\sqsupseteq\mathsf{ra}}x),\ (\mathsf{R}^{\sqsupseteq\mathsf{ra}}, a),\ (\mathsf{F}^{\sqsupseteq\mathsf{acq}}, a),\ (\mathsf{F}^{\sqsupseteq\mathsf{rel}}, \mathsf{W}),\ (\mathsf{W}^{\sqsupseteq\mathsf{ra}}x, \mathsf{W}x)\}$.

1.3 Model

Definition 1.2. A *pomset with predicate transformers* over $\mathcal{A}$ is a tuple $(E, \lambda, \kappa, \tau, \checkmark, \mathsf{rf}, \leq)$ where

(M1) $E \subseteq \mathcal{E}$ is a set of *events*,
(M2) $\lambda : E \to \mathcal{A}$ defines a *label* for each event,
(M3) $\kappa : E \to \Phi$ defines a *precondition* for each event,
(M4) $\tau : 2^{\mathcal{E}} \to \Phi \to \Phi$ is a *family of predicate transformers* over E,
(M5) $\checkmark : \Phi$ defines a *termination condition*,
(M6) $\mathsf{rf} : E \to E$ is an injective relation capturing *reads-from* such that
  (M6a) if $d \xrightarrow{\mathsf{rf}} e$ then $\lambda(d)$ matches $\lambda(e)$,
(M7) $\leq : E \times E$, is a partial order capturing *causality*, such that
  (M7a) if $d \xrightarrow{\mathsf{rf}} e$ and $\lambda(c)$ blocks $\lambda(e)$ then either $c \leq d$ or $e \leq c$.

A pomset is *top-level* if for every $e \in E$,

(M8) $\kappa(e)$ is a tautology,
(M9) if $\lambda(e)$ is a read then there is some $d \xrightarrow{\mathsf{rf}} e$.

Lemma 1.3. *For any P in the range of $[\![\cdot]\!]$, $d \xrightarrow{\mathsf{rf}} e$ implies $d \leq e$.*

Proof. Induction on the definition of $[\![\cdot]\!]$. □

Note that E_1 and E_2 are not necessarily disjoint. In *IF*, the definition of *extends* stops coalescing the rf in

$$\mathtt{if}(b)\{r := x \parallel x := 1\}\,\mathtt{else}\,\{r := x\,;\, x := 1\}$$

We have given the semantics of *IF* using disjunctive normal form. Dijkstra [1975] used conjunctive normal form. Note that $(\phi \wedge \theta_1) \vee (\neg\phi \wedge \theta_2)$ is logically equivalent to $(\phi \Rightarrow \theta_1) \wedge (\neg\phi \Rightarrow \theta_2)$.
We include empty sets as prep for adding while loops.

1.4 Arm Compilation

The model compiles correctly to arm using the lowering:

$$\|[r]^{\sqsupseteq\mathsf{ra}} := s\| = \mathtt{stlr}\ s, [r] \qquad \|s := [r]^{\mathsf{rlx}}\| = \mathtt{ldr}\ s, [r]$$
$$\|[r]^{\mathsf{rlx}} := s\| = \mathtt{str}\ s, [r] \qquad \|s := [r]^{\sqsupseteq\mathsf{ra}}\| = \mathtt{dmb\ st;\ ldar}\ s, [r]$$

Two changes in the definition of sequential composition:

- Replace (s7b) with: if $\lambda_1(c)$ blocks $\lambda_2(e)$ then $d \xrightarrow{\mathsf{rf}} e$ implies $c \leq d$.
- Replace $\bowtie_{\mathsf{co}}$ by $\ltimes_{\mathsf{lws}}$ in Def 1.1 of *delays*, where $\ltimes_{\mathsf{lws}} = \{(\mathsf{W}x, \mathsf{W}x),\ (\mathsf{R}x, \mathsf{W}x)\}$,

If one wants a post-hoc verification technique for rf, it is possible to include program order (po) in the pomset. For any $d \xrightarrow{\mathsf{rf}} e$ require either

- external fulfillment: $d \leq e$ and if $\lambda(c)$ blocks $\lambda(e)$ then either $c \leq d$ or $e \leq c$,
- internal fulfillment: $d \xrightarrow{\mathsf{po}} e$ and $(\nexists c)\ \lambda(c)$ blocks $\lambda(e)$ and $d \xrightarrow{\mathsf{po}} c \xrightarrow{\mathsf{po}} e$.

Suppose $R_1 : E_1 \times E_1$ and $R_2 : E_2 \times E_2$.
We say *R extends R_1 and R_2* if $R \supseteq (R_1 \cup R_2)$ and $R \cap (E_1 \times E_1) = R_1$ and $R \cap (E_2 \times E_2) = R_2$.

If $P \in \mathit{LET}(r, M)$ then $E = \emptyset$ and $\tau^D(\psi) \vDash \psi[M/r]$.

If $P \in \mathit{READ}(r, x, \mu)$ then $(\exists v \in \mathcal{V})$
- (R1) if $d, e \in E$ then $d = e$,
- (R2) $\lambda(e) = \mathsf{R}^\mu x v$,
- (R4a) if $(E \cap D) \neq \emptyset$ then $\tau^D(\psi) \vDash v{=}r \Rightarrow \psi$,
- (R4b) if $(E \cap D) = \emptyset$ then $\tau^D(\psi) \vDash \psi$.

If $P \in \mathit{WRITE}(x, M, \mu)$ then $(\exists v \in \mathcal{V})$
- (W1) if $d, e \in E$ then $d = e$,
- (W2) $\lambda(e) = \mathsf{W}^\mu x v$,
- (W3) $\kappa(e) \vDash M{=}v$,
- (W4) $\tau^D(\psi) \vDash \psi$,
- (W5a) if $E \neq \emptyset$ then $\checkmark \vDash M{=}v$,
- (W5b) if $E = \emptyset$ then $\checkmark \vDash \mathsf{ff}$.

If $P \in \mathit{FENCE}(\mu)$ then
- (F1) if $d, e \in E$ then $d = e$,
- (F2) $\lambda(e) = \mathsf{F}^\mu$,
- (F4) $\tau^D(\psi) \vDash \psi$,
- (F5) if $E = \emptyset$ then $\checkmark \vDash \mathsf{ff}$.

If $P \in \mathit{SKIP}$ then $E = \emptyset$ and $\tau^D(\psi) \vDash \psi$.

If $P \in \mathcal{P}_1 \parallel \mathcal{P}_2$ then $(\exists P_1 \in \mathcal{P}_1)\ (\exists P_2 \in \mathcal{P}_2)$
- (P1) $E = (E_1 \uplus E_2)$,
- (P2) $\lambda = (\lambda_1 \cup \lambda_2)$,
- (P3a) if $e \in E_1$ then $\kappa(e) \vDash \kappa_1(e)$,
- (P3b) if $e \in E_2$ then $\kappa(e) \vDash \kappa_2(e)$,
- (P4) $\tau^D(\psi) \vDash \tau_1^D(\psi)$,
- (P5) $\checkmark \vDash \checkmark_1 \wedge \checkmark_2$,
- (P6) rf extends rf_1 and rf_2,
- (P7a) $\leq$ extends $\leq_1$ and $\leq_2$,
- (P7b) if $d \in E_1$, $e \in E_2$ and $d \xrightarrow{\mathsf{rf}} e$ then $d \leq e$.

If $P \in \mathcal{P}_1 ; \mathcal{P}_2$ then $(\exists P_1 \in \mathcal{P}_1)\ (\exists P_2 \in \mathcal{P}_2)$
let $\kappa_2'(e) = \tau_1^{\downarrow e}(\kappa_2(e))$, where $\downarrow e = \{c \mid c < e\}$
- (S1) $E = (E_1 \cup E_2)$,
- (S2) $\lambda = (\lambda_1 \cup \lambda_2)$,
- (S3a) if $e \in E_1 \setminus E_2$ then $\kappa(e) \vDash \kappa_1(e)$,
- (S3b) if $e \in E_2 \setminus E_1$ then $\kappa(e) \vDash \kappa_2'(e)$,
- (S3c) if $e \in E_1 \cap E_2$ then $\kappa(e) \vDash \kappa_1(e) \vee \kappa_2'(e)$,
- (S3d) if $\lambda_2(e)$ is a release then $\kappa(e) \vDash \checkmark_1$,
- (S4) $\tau^D(\psi) \vDash \tau_1^D(\tau_2^D(\psi))$,
- (S5) $\checkmark \vDash \checkmark_1 \wedge \tau_1(\checkmark_2)$,
- (S6) rf extends rf_1 and rf_2,
- (S7a) $\leq$ extends $\leq_1$ and $\leq_2$,
- (S7b) if $d \in E_1$, $e \in E_2$ and $d \xrightarrow{\mathsf{rf}} e$ then $d \leq e$,
- (S7c) if $\lambda_1(d)$ delays $\lambda_2(e)$ then $d \leq e$.

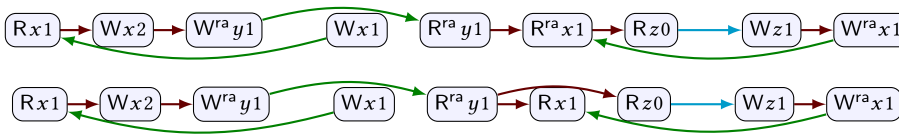
If $P \in \mathit{IF}(\phi, \mathcal{P}_1, \mathcal{P}_2)$ then $(\exists P_1 \in \mathcal{P}_1)\ (\exists P_2 \in \mathcal{P}_2)$
- (C1) $E = (E_1 \cup E_2)$,
- (C2) $\lambda = (\lambda_1 \cup \lambda_2)$,
- (C3a) if $e \in E_1 \setminus E_2$ then $\kappa(e) \vDash \phi \wedge \kappa_1(e)$,
- (C3b) if $e \in E_2 \setminus E_1$ then $\kappa(e) \vDash \neg\phi \wedge \kappa_2(e)$,
- (C3c) if $e \in E_1 \cap E_2$ then $\kappa(e) \vDash (\phi \wedge \kappa_1(e)) \vee (\neg\phi \wedge \kappa_2(e))$,
- (C4) $\tau^D(\psi) \vDash (\phi \wedge \tau_1^D(\psi)) \vee (\neg\phi \wedge \tau_2^D(\psi))$,
- (C5) $\checkmark \vDash (\phi \wedge \checkmark_1) \vee (\neg\phi \wedge \checkmark_2)$.
- (C6a) rf extends rf_1 and rf_2,
- (C6b) $\mathsf{rf} \subseteq (\mathsf{rf}_1 \cup \mathsf{rf}_2)$,
- (C7a) $\leq$ extends $\leq_1$ and $\leq_2$,
- (C7b) $\leq \subseteq (\leq_1 \cup \leq_2)$.

$$
\begin{aligned}
[\![r := M]\!] &= \mathit{LET}(r, M) & [\![\mathtt{skip}]\!] &= \mathit{SKIP} \\
[\![r := x^\mu]\!] &= \mathit{READ}(r, x, \mu) & [\![S_1 \parallel S_2]\!] &= [\![S_1]\!] \parallel [\![S_2]\!] \\
[\![x^\mu := M]\!] &= \mathit{WRITE}(x, M, \mu) & [\![S_1 ; S_2]\!] &= [\![S_1]\!] ; [\![S_2]\!] \\
[\![\mathsf{F}^\nu]\!] &= \mathit{FENCE}(\nu) & [\![\mathtt{if}(M)\{S_1\}\,\mathtt{else}\,\{S_2\}]\!] &= \mathit{IF}(M{\neq}0, [\![S_1]\!], [\![S_2]\!])
\end{aligned}
$$

Fig. 1. Semantics of programs

Downgrading messes up publication:

$$x := x + 1;\, y^{\mathsf{ra}} := 1 \parallel x := 1;\, \mathtt{if}(y^{\mathsf{ra}}\ \&\&\ x^{\mathsf{ra}})\{s := z\} \parallel z := 1;\, x^{\mathsf{ra}} := 1$$



Restrict to top level parallel composition.

We give an abstract view of Arm8 executions, leaving out many details. For example, the definition of A4a

Definition 1.4. An *Arm8 execution graph* is tuple $(E, \lambda, \mathsf{poloc}, \mathsf{lob})$ such that

(A1) $E \subseteq \mathcal{E}$ is a set of events,
(A2) $\lambda : E \rightarrow \mathcal{A}$ defines a label for each event,
(A3) $\mathsf{poloc} : E \times E$, is a per-thread, per-location total order, capturing *per-location program order*,
(A4) $\mathsf{lob} : E \times E$, is a per-thread partial order capturing *locally-ordered-before*, such that
  (A4a) $\mathsf{poloc} \cup \mathsf{lob}$ is acyclic.

An Arm8 execution graph G is *EC-valid for S via* $(\mathsf{co}, \mathsf{rf}, \mathsf{cb})$ if G is generated by S and

(A5) $\mathsf{co} : E \times E$, is a per-location total order on writes, capturing *coherence*,
(A6) $\mathsf{rf} : E \times E$, is a surjective and injective relation on reads, capturing *reads-from*, such that
  (A6a) if $d \xrightarrow{\mathsf{rf}} e$ then $\lambda(d)$ matches $\lambda(e)$,
  (A6b) $\mathsf{poloc} \cup \mathsf{co} \cup \mathsf{rf} \cup \mathsf{fr}$ is acyclic, where $e \xrightarrow{\mathsf{fr}} c$ if $e \xleftarrow{\mathsf{rf}} d \xrightarrow{\mathsf{co}} c$, for some d,
(A7) $\mathsf{cb} \supseteq (\mathsf{co} \cup \mathsf{lob})$ is a linear order such that if $d \xrightarrow{\mathsf{rf}} e$ then either
  (A7a) $d \xrightarrow{\mathsf{cb}} e$ and if $\lambda(c)$ blocks $\lambda(e)$ then either $c \xrightarrow{\mathsf{cb}} d$ or $e \xrightarrow{\mathsf{cb}} c$, or
  (A7b) $d \xleftarrow{\mathsf{cb}} e$ and $d \xrightarrow{\mathsf{poloc}} e$ and $(\nexists c)\ \lambda(c)$ blocks $\lambda(e)$ and $d \xrightarrow{\mathsf{poloc}} c \xrightarrow{\mathsf{poloc}} e$.

An Arm8 execution graph G is *EGC-valid for S via* $(\mathsf{co}, \mathsf{rf}, \mathsf{gcb})$ if G is generated by S and

(A5) and (A6), as for EC,
(A8) $\mathsf{gcb} \supseteq (\mathsf{co} \cup \mathsf{rf})$ is a linear order such that if $d \xrightarrow{\mathsf{rf}} e$ then either
  (A8a) if $d \xrightarrow{\mathsf{rf}} e$ and c blocks e then either $c \xrightarrow{\mathsf{gcb}} d$ or $e \xrightarrow{\mathsf{gcb}} c$,
  (A8b) if $d \xrightarrow{\mathsf{lob}} e$ then either $d \xrightarrow{\mathsf{gcb}} e$ or $(\exists c)\ c \xrightarrow{\mathsf{rf}} d$ and $c \xrightarrow{\mathsf{poloc}} d$ but not $c \xrightarrow{\mathsf{lob}} e$.

Lemma 1.5. *Suppose G is an execution graph that is* EC*-valid via* $(\mathsf{co}, \mathsf{rf}, \mathsf{cb})$*. Then there a permutation cb' of cb such that G is* EC*-valid via* $(\mathsf{co}, \mathsf{rf}, \mathsf{cb}')$ *and* $\mathsf{cb}' \supseteq \mathsf{fr}$*, where* fr *is defined in* A6b*.*

Proof. We show that any cb order that contradicts fr is incidental.

By definition of fr, $e \xleftarrow{\mathsf{rf}} d \xrightarrow{\mathsf{co}} c$, for some d. Since $\mathsf{cb} \supseteq \mathsf{co}$, we know that $d \xrightarrow{\mathsf{co}} c$.

If A7a applies to $d \xrightarrow{\mathsf{rf}} e$, then $e \xrightarrow{\mathsf{cb}} c$, since it cannot be that $c \xrightarrow{\mathsf{co}} d$.

Suppose A7b applies to $d \xrightarrow{\mathsf{rf}} e$ and c is from a different thread. Because it is a different thread, we cannot have $e \xrightarrow{\mathsf{lob}} c$, and thus the order in cb is incidental.

Suppose A7b applies to $d \xrightarrow{\mathsf{rf}} e$ and c is from the same thread. Since $c \xrightarrow{\mathsf{co}} d$, it cannot be that $c \xrightarrow{\mathsf{poloc}} d$, using A6b. It also cannot be that $d \xrightarrow{\mathsf{poloc}} c \xrightarrow{\mathsf{poloc}} e$. It must be that $e \xrightarrow{\mathsf{poloc}} c$. By A4a, we cannot have $e \xrightarrow{\mathsf{lob}} c$, and thus the order in cb is incidental. $\square$

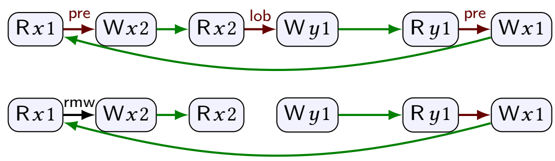
Theorem 1.6. *Suppose G_1 is* EC*-valid for S via* $(\mathsf{co}_1, \mathsf{rf}_1, \mathsf{cb}_1)$ *and that* $\mathsf{cb}_1 \supseteq \mathsf{fr}_1$*. Then there is a top-level pomset $P_2 \in [\![S]\!]$ such that $E_2 = E_1$, $\lambda_2 = \lambda_1$, $\mathsf{rf}_2 = \mathsf{rf}_1$, and $\leq_2 = \mathsf{cb}_1$.*

Proof. We show that all the order required in the pomset is also required by Arm8. Dependency order required by s3 is also required by lob. Synchronization order required $\ltimes_{\mathsf{sync}}$ and $\bowtie_{\mathsf{sc}}$ in s7c is also required by lob. Write-to write coherence required by $\ltimes_{\mathsf{co}}$ in s7c is also required in cb, by A7. Read-to-write coherence required by $\ltimes_{\mathsf{co}}$ in s7c is also required in cb, by assumption. (By Lemma

1.5, there is no loss of generality). M7a holds since cb_1 is consistent with co_1 and fr_1. s7b follows from A7b. ☐

Bad example:

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

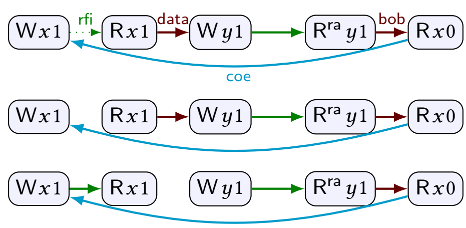


(✓Arm8)

(≤)

Armed cats example (changed address to data dependency):

$$x := 1\,;\ r := x\,;\ y := r \parallel 1 := y^{\mathsf{ra}}\,;\ s := x$$



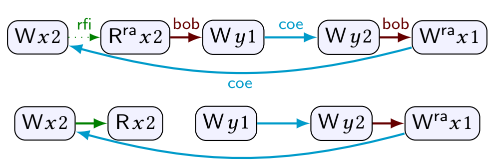
(✓Arm8)

(✓EC)

(✓EGC)

Anton example 1 `[rfi-coe-coe]`

$$x := 2\,;\ r := x^{\mathsf{ra}}\,;\ y := 1 \parallel y := 2\,;\ x^{\mathsf{ra}} := 1$$ (RFI-COE-COE)



(✓Arm8)

1.5 Full Versions

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

(W1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,
(W2) $\lambda(e) = \mathsf{W}^{\mu} x v_e$,
(W3) $\kappa(e) \vDash \theta_e \wedge M{=}v_e$,
(W4) $\tau^D(\psi) \vDash \theta_e \Rightarrow \psi[M/x]$,
(W5a) $\checkmark \vDash \theta_e \Rightarrow M{=}v_e$,
(W5b) $\checkmark \vDash \bigvee_{e \in E} \theta_e$.

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

(R1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,
(R2) $\lambda(e) = \mathsf{R}^{\mu} x v_e$
(R3) $\kappa(e) \vDash \theta_e$,
(R4a) $(\forall e \in E \cap D)\ \tau^D(\psi) \vDash \theta_e \Rightarrow v_e{=}s_e \Rightarrow \psi[s_e/r]$,
(R4b) $(\forall e \in E \setminus D)\ \tau^D(\psi) \vDash \theta_e \Rightarrow (v_e{=}s_e \vee x{=}s_e) \Rightarrow \psi[s_e/r]$,
(R4c) $(\forall s)\ \tau^D(\psi) \vDash (\bigwedge_{e \in E} \neg\theta_e) \Rightarrow \psi[s/r]$.

1.6 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 1.7. A *pomset* over $\mathcal{A}$ is a tuple $(E, \leq, \lambda)$ where

- $E \subset \mathcal{E}$ is a set of *events*,
- $\le\, \subseteq (E \times E)$ is the *causality* partial order,
- $\lambda : E \rightarrow \mathcal{A}$ 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 1.8. Action $(\mathsf{W}xv)$ *matches* $(\mathsf{R}xw)$ when $v = w$. Action $(\mathsf{W}xv)$ *blocks* $(\mathsf{R}xw)$, for any v, w.

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

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

Definition 1.9. Actions a and b are *reorderable* ($a \bowtie b$) if either both are reads or they are accesses to different locations. Formally $\bowtie\, = \{(\mathsf{R}xv, \mathsf{R}yw)\} \cup \{(\mathsf{R}xv, \mathsf{W}yw), (\mathsf{W}xv, \mathsf{R}yw), (\mathsf{W}xv, \mathsf{W}yw) \mid x \ne y\}$.

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 1.10. If $P \in \mathit{NIL}$ 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 $d \le_1 e$ then $d \le e$,
3. if $d \le_2 e$ then $d \le e$,
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 $P \in (a \rightarrow \mathcal{P}_2)$ 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 $d \le_2 e$ then $d \le e$,
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 $d \le e$ or $a \bowtie \lambda_2(e)$.

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

1. $E = E_1$,
2. $\lambda(e) = \lambda_1(e)$,
3. if $d \le_1 e$ then $d \le e$,
4. if $\lambda_1(e)$ is a read then e is fulfilled (Def 1.8).

Definition 1.11. For a language fragment, the semantics is:

$$\llbracket x^{\mu} := v\, ;\, S \rrbracket = (\mathsf{W}xv) \rightarrow \llbracket S \rrbracket \qquad \llbracket \mathsf{skip} \rrbracket = \llbracket 0 \rrbracket = \mathit{NIL}$$

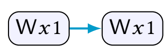
$$\llbracket r := x^{\mu}\, ;\, S \rrbracket = \textstyle\bigcup_v (\mathsf{R}xv) \rightarrow \llbracket S \rrbracket \qquad \llbracket S_1 \parallel S_2 \rrbracket = \llbracket S_1 \rrbracket \parallel \llbracket S_2 \rrbracket$$

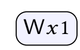
In this semantics, both `skip` and 0 map to the empty pomset. Parallel composition is disjoint union, inheriting labeling and order from the two sides. Prefixing may add a new action (on the left) to an existing pomset (on the right), inheriting labeling and order from the right.

It is worth noting that if $\bowtie$ is taken to be the empty relation, then top-level pomsets of Def 1.7 correspond to sequentially consistent executions up to mumbling [Brookes 1996].

Example 1.12. 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.

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

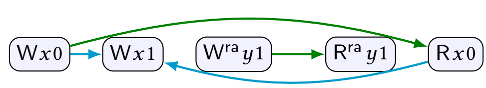




In the left pomset, the order between the events is enforced by clause 6, since the actions are in conflict.

Example 1.13. 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 **??**.

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,
- $e \rightarrow d$ arises from fulfillment, where e *matches* d,
- $e \rightarrow d$ arises either from fulfillment, where e *blocks* d, or from prefixing, where e was prefixed before d and their actions *conflict*,
- $e \rightarrow d$ arises from control/data/address *dependency*,
- $e \rightarrow d$ arises from *synchronized access*.

Definition 1.14. $\mathcal{P}_1$ *refines* $\mathcal{P}_2$ if $\mathcal{P}_1 \subseteq \mathcal{P}_2$.

Example 1.15. Ex 1.12 shows that $[\![x := 1]\!]$ refines $[\![x := 1\,;\ x := 1]\!]$.

1.7 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 1.16. A *pomset with preconditions* is a pomset (Def 1.7) together with $\kappa : E \longrightarrow \Phi$.

Definition 1.17. 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 1.18. If $P \in \mathit{NIL}$ 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 1.10,
(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 $P \in \mathit{IF}(\phi, \mathcal{P}_1, \mathcal{P}_2)$ then $(\exists P_1 \in \mathcal{P}_1)$ $(\exists P_2 \in \mathcal{P}_2)$
1–5) as for $\parallel$ in Def 1.10 (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 $P \in \mathit{WR}(x, M, \mathcal{P}_2)$ then $(\exists P_2 \in \mathcal{P}_2)$ $(\exists v \in \mathcal{V})$
1–6) as for $(\mathsf{W}xv) \longrightarrow \mathcal{P}_2$ in Def 1.10,
(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 $P \in \mathit{RD}(r, x, \mathcal{P}_2)$ then $(\exists P_2 \in \mathcal{P}_2)$ $(\exists v \in \mathcal{V})$
1–6) as for $(\mathsf{R}xv) \longrightarrow \mathcal{P}_2$ in Def 1.10,
(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 $P \in \mathit{TOP}(\mathcal{P})$ then $(\exists P_1 \in \mathcal{P})$
1–4) as for TOP in Def 1.10,
(5) if $\lambda_1(e)$ is a write, $\kappa_1(e)[\mathsf{tt}/\mathsf{Q}][\mathsf{tt}/\mathsf{W}]$ is a tautology,
(6) if $\lambda_1(e)$ is a read, $\kappa_1(e)[\mathsf{tt}/\mathsf{Q}][\mathsf{ff}/\mathsf{W}]$ is a tautology.

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

Definition 1.19. For a language fragment, the semantics is:

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

Example 1.20. 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 $v = 1$:

$$\begin{gathered} y := r \\ \boxed{r{=}1 \mid \mathsf{W}y1} \end{gathered}$$

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

$$\begin{gathered} r := x\,;\, y := r \\ \boxed{\mathsf{R}x0} \rightarrow \boxed{r{=}0 \Rightarrow r{=}1 \mid \mathsf{W}y1} \end{gathered}$$

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:

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

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

Example 1.21. Control dependencies are similar, for example for any $P \in [\![r := x \,;\, \texttt{if}(r)\{y := 1\}]\!]$, there must be an $v \in \mathcal{V}$ and $P' \in [\![\texttt{if}(r)\{y := 1\}]\!]$ such as:

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

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

The rest of the reasoning is the same as Ex 1.20.

Example 1.22. 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 1.23. Consider $P \in [\![\texttt{if}(r{=}1)\{y := r\}\,\texttt{else}\,\{y := 1\}]\!]$, so there must be $P_1 \in [\![y := r]\!]$, and $P_2 \in [\![y := 1]\!]$, such as:

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

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

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:

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

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

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

1.8 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 1.24. A *predicate transformer* is a function $\tau : \Phi \longrightarrow \Phi$ such that

- $\tau(\mathsf{ff})$ is ff,
- $\tau(\psi_1 \land \psi_2)$ is $\tau(\psi_1) \land \tau(\psi_2)$,
- $\tau(\psi_1 \lor \psi_2)$ is $\tau(\psi_1) \lor \tau(\psi_2)$,
- 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 §??, 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 1.25. 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 1.26. A *pomset with predicate transformers* is a pomset with preconditions (Def 1.18), together with a family of predicate transformers for E.

Definition 1.27. If $P \in \mathit{ABORT}$ then $E = \emptyset$ and

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

If $P \in \mathit{SKIP}$ then $E = \emptyset$ and

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

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

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

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

1–8) as for *IF* in Def 1.18,

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

If $P \in \mathcal{P}_1 \mathbin{]\!\!|} \mathcal{P}_2$ then $(\exists P_1 \in \mathcal{P}_1)$ $(\exists P_2 \in \mathcal{P}_2)$

1–P3b) as for $\|$ in Def 1.18,

(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 $\|$ in Def 1.10 (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 $P \in \mathit{WRITE}(x, M, \mu)$ 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 $P \in \mathit{READ}(r, x, \mu)$ then $(\exists v \in \mathcal{V})$

L1) if $d, e \in E$ then $d = e$,

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

L3) $\kappa(e) \vDash \mathsf{tt}$,

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 1.18,
(7) $\tau^{E_1}(\mathsf{s}) \models \mathsf{s}$, for every quiescence symbol s.

Definition 1.28. The semantics of commands is:

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

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

$$[\![r := x^\mu]\!] = \mathit{READ}(r,\ x,\ \mu) \qquad [\![S_1 \parallel S_2]\!] = [\![S_1]\!] \parallel [\![S_2]\!]$$

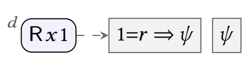
$$[\![r := M]\!] = \mathit{LET}(r,\ M) \qquad [\![S_1\,;\,S_2]\!] = [\![S_1]\!]\,;\,[\![S_2]\!]$$

$$[\![\mathsf{F}^\mu]\!] = \mathit{FENCE}(\mu)$$

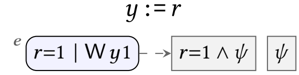
Most of these definitions are straightforward adaptations of §1.7, but the treatment of sequential composition is new. This uses the usual rule for composition of predicate transformers (but preserving the indexing set). For the pomset, we take the union of their events, preserving actions, but crucially 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 1.29. For read to write dependency, consider:

$r := x$

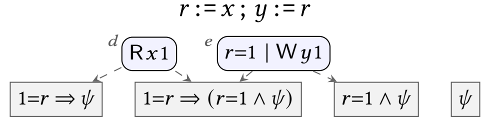


$y := r$



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:

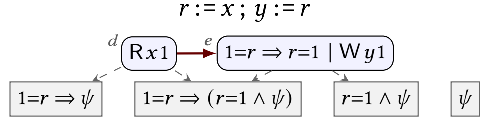
$r := x\,;\, y := r$



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:

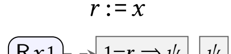
$r := x\,;\, y := r$



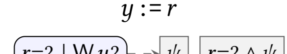
This pomset's preconditions do not depend on a bound register, so can contribute to a top-level pomset.

Example 1.30. If the read and write choose different values:

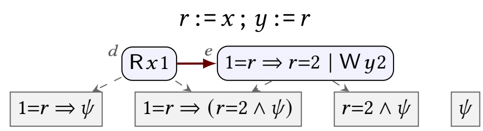
$r := x$



$y := r$

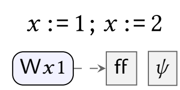


Putting these together with order, we have the following, which cannot be part of a top-level pomset:



Example 1.31. S4 includes $M{=}v$ to ensure that spurious merges do not go undetected. Consider the following.

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



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 1.32. 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:

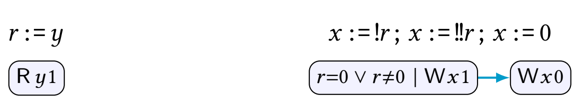
L4) $\tau^D(\psi) \vDash \psi[v/r]$,
L5) $\tau^C(\psi) \vDash (\forall r)\psi$.

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 1.25, which states that the predicate transformer for a small subset of E must imply the transformer for a larger subset.

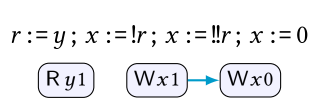
Sadly, this definition fails associativity.

Consider the following, eliding transformers:

Associating to the right and merging:



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



If, instead, we associate to the left:

Sequencing and merging:

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

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

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

Our solution is to Skolemize. We have proven associativity of Def 1.27 in Agda. The proof requires that predicate transformers distribute through disjunction (Def 1.24). 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: $\tau(\psi_1 \lor \psi_2) = (\forall r)(\psi_1 \lor \psi_2) \neq ((\forall r)(\psi_1)) \lor ((\forall r)(\psi_2)) = \tau(\psi_1) \lor \tau(\psi_2)$.

1.9 The Road Ahead

The final semantic functions for load, store, and thread initialization are given in Fig **??**, at the end of the paper. In §**??**–**??**, we explain this definition by looking at its constituent parts, building on Def 1.27. In §**??**, we add *quiescence*, which encodes coherence, release-acquire access, and SC access. In §**??**, we add peculiarities that are necessary for efficient implementation on Arm8. In §**??**, 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 1.27, 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.

REFERENCES

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

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

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

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

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

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