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), 7 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.

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

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

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 $\parallel$ 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:

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

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 rlx $\sqsubseteq$ ra $\sqsubseteq$ sc and rel $\sqsubseteq$ ar and acq $\sqsubseteq$ 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 $(E, \lambda, \kappa, \tau, \checkmark, \mathsf{rf}, \le)$ 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) $\le : E \times E$, is a partial order capturing *causality*, such that
  - (M7a) if $d \xrightarrow{\mathsf{rf}} e$ and $\lambda(c)$ blocks $\lambda(e)$ then either $c \le d$ or $e \le c$.

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 $d \le e$.*

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

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

See Fig ??.

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 $(\phi \Rightarrow \theta_1) \wedge (\neg\phi \Rightarrow \theta_2)$.

1.4 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) $(\forall s)\ \tau^D(\psi) \vDash (\bigwedge_{e \in E} \neg\theta_e) \Rightarrow \psi[s/r]$.

2 ARM

Restrict to top level parallel composition.

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 $P \in \mathcal{P}_1 \mathbin{|\!|\!|} \mathcal{P}_2$ 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) $\le$ extends $\le_1$ and $\le_2$,
- (P7b) if $d \in E_1$, $e \in E_2$ and $d \xrightarrow{\mathsf{rf}} e$ then $d \le e$.

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) $\le$ extends $\le_1$ and $\le_2$,
- (S7b) if $d \in E_1$, $e \in E_2$ and $d \xrightarrow{\mathsf{rf}} e$ then $d \le e$,
- (S7c) if $\lambda_1(d)$ delays $\lambda_2(e)$ then $d \le e$.

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) $\le$ extends $\le_1$ and $\le_2$,
- (C7b) $\le \subseteq (\le_1 \cup \le_2)$.

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

Fig. 1. Semantics of programs

2.1 Arm executions

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

Definition 2.1. An *Arm8 execution graph* is tuple $(E, \lambda, \textsf{poloc}, \textsf{lob})$ such that

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

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

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

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

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

2.2 Arm Compilation 1

Arm does not enforce read-read control dependencies. So we need to modify the definition of of $\downarrow e$:

$$\downarrow e = \begin{cases} \{c \mid c < e\} & \text{if } \lambda(e) \text{ is a write} \\ E_1 & \text{otherwise} \end{cases}$$

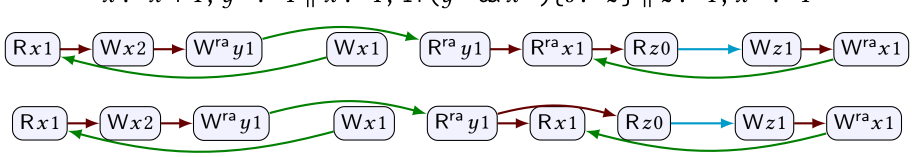
Podkopaev et al. [2019] lowers to Arm8 as follows: Relaxed access is implemented using `ldr`/`str`, non-relaxed access using `ldar`/`stlr`, acquire and other fences using `dmb.ld`/`dmb.sy`.

THEOREM 2.2. *Our model lowers to arm correctly if non-relaxed reads are lowered to* `dmb st; ldar`.

PROOF. Use EGC. □

Downgrading messes up publication:

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



2.3 Arm Compilation 2

Two changes in the definition of sequential composition:

- Replace (s7b) with: if $\lambda_1(c)$ blocks $\lambda_2(e)$ then $d \xrightarrow{\textsf{rf}} e$ 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: $d \le e$ and if $\lambda(c)$ blocks $\lambda(e)$ then either $c \le d$ or $e \le c$,
- internal fulfillment: $d \xrightarrow{\mathsf{po}} e$ and $(\nexists c)\ \lambda(c)$ blocks $\lambda(e)$ and $d \xrightarrow{\mathsf{po}} c \xrightarrow{\mathsf{po}} e$.

Lemma 2.3. *Suppose G is an execution graph that is* EC*-valid via* (co, rf, cb)*. Then there a permutation* cb′ *of* cb *such that G is* EC*-valid via* (co, rf, cb′) *and* cb′ $\supseteq$ fr*, where* fr *is defined in* A6b*.*

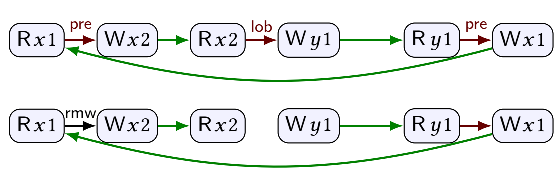
Proof. We show that any cb order that contradicts fr is incidental.
By definition of fr, $e \xleftarrow{\mathsf{rf}} d \xrightarrow{\mathsf{co}} c$, for some d. Since cb $\supseteq$ co, we know that $d \xrightarrow{\mathsf{co}} c$.
If A7a applies to $d \xrightarrow{\mathsf{rf}} e$, then $e \xrightarrow{\mathsf{cb}} c$, 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. □

Theorem 2.4. *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* $\le_2 = \mathsf{cb}_1$*.*

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 2.3, 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:

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

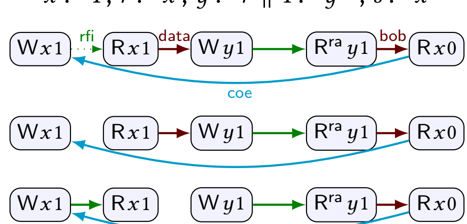


(✓Arm8)

(≤)

Armed cats example (changed address to data dependency):

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



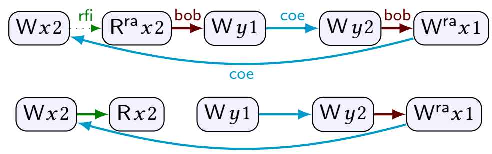
(✓Arm8)

(✓EC)

(✓EGC)

Anton example 1 [rfi-coe-coe]

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



(✓Arm8)

REFERENCES

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

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, 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