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), 8 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: $M[N/x] = M$.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2021 Copyright held by the owner/author(s).
2475-1421/2021/10-ART0
https://doi.org/

We model the following language.

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

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

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], $\urcorner\!\!\urcorner$ 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:

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

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. 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 $a \bowtie_{\mathsf{co}} b$ or $a \ltimes_{\mathsf{sync}} b$ or $a \bowtie_{\mathsf{sc}} b$.

Let $\bowtie_{\mathsf{co}}$ capture write-write, read-write coherence: $\bowtie_{\mathsf{co}} = \{(\mathsf{W}x, \mathsf{W}x),\ (\mathsf{R}x, \mathsf{W}x),\ (\mathsf{W}x, \mathsf{R}x)\}$.

Let $\ltimes_{\mathsf{sync}}$ capture order due to synchronization: $\ltimes_{\mathsf{sync}} = \{(a, \mathsf{W}^{\sqsupseteq \mathsf{ra}}),\ (a, \mathsf{F}^{\sqsupseteq \mathsf{rel}}),\ (\mathsf{R}, \mathsf{F}^{\sqsupseteq \mathsf{acq}}),\ (\mathsf{R}x, \mathsf{R}^{\sqsupseteq \mathsf{ra}}x),\ (\mathsf{R}^{\sqsupseteq \mathsf{ra}}, a),\ (\mathsf{F}^{\sqsupseteq \mathsf{acq}}, a),\ (\mathsf{F}^{\sqsupseteq \mathsf{rel}}, \mathsf{W}),\ (\mathsf{W}^{\sqsupseteq \mathsf{ra}}x, \mathsf{W}x)\}$.

Let $\bowtie_{\mathsf{sc}}$ capture order due to sc access: $\bowtie_{\mathsf{sc}} = \{(\mathsf{W}^{\mathsf{sc}}, \mathsf{W}^{\mathsf{sc}}),\ (\mathsf{R}^{\mathsf{sc}}, \mathsf{W}^{\mathsf{sc}}),\ (\mathsf{W}^{\mathsf{sc}}, \mathsf{R}^{\mathsf{sc}}),\ (\mathsf{R}^{\mathsf{sc}}, \mathsf{R}^{\mathsf{sc}})\}$.

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) $\lambda : E \longrightarrow \mathcal{A}$ defines a *label* for each event,
(M3) $\kappa : E \longrightarrow \Phi$ defines a *precondition* for each event,
(M4) $\tau : 2^{\mathcal{E}} \longrightarrow \Phi \longrightarrow \Phi$ is a *family of predicate transformers* over E,
(M5) $\checkmark : \Phi$ defines a *termination condition*,
(M6) $\mathsf{rf} : E \longrightarrow E$ 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 $d \xrightarrow{\mathsf{rf}} e$.

We give the semantics of programs in Fig 1.

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

Proof. Induction on the definition of $\llbracket\cdot\rrbracket$. □

The semantics to be closed with respect to *augmentation* 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 1.4. 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) $\checkmark_2 \vDash \checkmark_1$,
(6) $\mathsf{rf}_2 = \mathsf{rf}_1$,
(7) $\leq_2 \supseteq \leq_1$.

Lemma 1.5. *If $P_1 \in \llbracket S\rrbracket$ and P_2 augments P_1 then $P_2 \in \llbracket S\rrbracket$.*

Proof. Induction on the definition of $\llbracket\cdot\rrbracket$. □

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

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

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 $(\exists v : E \longrightarrow \mathcal{V})\ (\exists \theta : E \longrightarrow \Phi)$

(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) $\checkmark \vDash \bigvee_{e \in E} \theta_e$.

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 $P \in \mathit{SKIP}$ 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 $P \in \mathit{IF}(\phi, \mathcal{P}_1, \mathcal{P}_2)$ 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) $\leq \subseteq (\leq_1 \cup \leq_2)$.

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 $P \in \mathit{LET}(r, M)$ 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 $E \neq \emptyset$ and $(E \cap D) = \emptyset$ then $\tau^D(\psi) \vDash (v{=}r \vee x{=}r) \Rightarrow \psi$.
- (R4c) if $E = \emptyset$ 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) $\tau^D(\psi) \vDash \psi[M/x]$,
- (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 $\checkmark \vDash \mathsf{ff}$.

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

Fig. 1. Semantics of programs

If $P \in READ(r, x, \mu)$ 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

For simplicity, we restrict to top level parallel composition and ignore fences[1].

2.1 Arm executions

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

(A1) $E \subseteq \mathcal{E}$ is a set of events,
(A2) $\lambda : E \to \mathcal{A}$ 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.

The definition of lob is complex. Comparing with our definition of sequential composition, it is sufficient to note that lob includes

(L1) read-write dependencies, required by s3,
(L2) synchronization delay of $\ltimes_{\mathsf{sync}}$, required by s7c,
(L3) sc access delay of $\bowtie_{\mathsf{sc}}$, required by s7c,
(L4) write-write and read-to-write coherence delay of $\ltimes_{\mathsf{co}}$, required by s7c,

and that lob does *not* include

(L5) read-read control dependencies, required by s3,
(L6) write-to-read order of rf, required by s7b,
(L7) write-to-read coherence delay of $\ltimes_{\mathsf{co}}$, required by s7c.

Definition 2.2. Execution G is $(\mathsf{co}, \mathsf{rf}, \mathsf{gcb})$*-valid*, under *External Global Consistency* (EGC) if

(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) $\mathsf{gcb} \supseteq (\mathsf{co} \cup \mathsf{rf})$ is a linear order such that
  (A7a) if $d \xrightarrow{\mathsf{rf}} e$ and $\lambda(c)$ blocks $\lambda(e)$ then either $c \xrightarrow{\mathsf{gcb}} d$ or $e \xrightarrow{\mathsf{gcb}} c$,
  (A7b) if $e \xrightarrow{\mathsf{lob}} c$ then either $e \xrightarrow{\mathsf{gcb}} c$ or $(\exists d)$ $d \xrightarrow{\mathsf{rf}} e$ and $d \xrightarrow{\mathsf{poloc}} e$ but not $d \xrightarrow{\mathsf{lob}} c$.

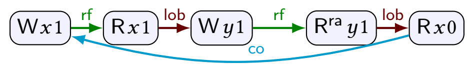
Execution G is $(\mathsf{co}, \mathsf{rf}, \mathsf{cb})$*-valid* under *External Consistency* (EC) if

(A5) and (A6), as for EGC,
(A8) $\mathsf{cb} \supseteq (\mathsf{co} \cup \mathsf{lob})$ is a linear order such that if $d \xrightarrow{\mathsf{rf}} e$ then either
  (A8a) $d \xrightarrow{\mathsf{cb}} e$ and if $\lambda(c)$ blocks $\lambda(e)$ then either $c \xrightarrow{\mathsf{cb}} d$ or $e \xrightarrow{\mathsf{cb}} c$, or
  (A8b) $d \xleftarrow{\mathsf{cb}} e$ and $d \xrightarrow{\mathsf{poloc}} e$ and $(\nexists c)$ $\lambda(c)$ blocks $\lambda(e)$ and $d \xrightarrow{\mathsf{poloc}} c \xrightarrow{\mathsf{poloc}} e$.

[1]Fences are not actions in Arm8, which complicates the theorem statements.

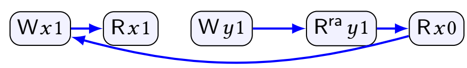
[Alglave et al. 2021] explain EGC and EC using the following example.[2]

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



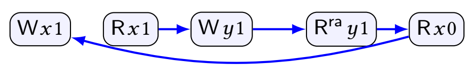
(✓Arm8)

EGC drops lob-order in the first thread using 2.4, since (Wx1) is not lob-ordered before (Wy1).



(gcb)

EC drops rf-order in the first thread using A8b.



(cb)

2.2 Arm Compilation 1

Podkopaev et al. [2019] lowers to Arm8 as follows: Relaxed access is implemented using `ldr/str`, non-relaxed access using `ldar/stlr`. In this section, we consider a suboptimal strategy, which lowers non-relaxed reads to (`dmb.sy; ldar`).

We do not distinguish control dependencies from address dependencies, and therefore L5 forces us to drop all dependencies between reads. To achieve this, we modify the definition of κ_2' in Fig 1.

Definition 2.3. Let $[\![\cdot]\!]_{\mathsf{RR}}$ be as defined in Fig 1, replacing the definition of κ_2' with:

$$\kappa_2'(e) = \begin{cases} \tau_1(\kappa_2(e)) & \text{if } \lambda(e) \text{ is a read} \\ \tau_1^{\downarrow e}(\kappa_2(e)) & \text{otherwise, where } {\downarrow}e = \{c \mid c < e\} \end{cases}$$

THEOREM 2.4. *Suppose G_1 is $(\mathsf{co}_1, \mathsf{rf}_1, \mathsf{gcb}_1)$-valid for S under the suboptimal lowering that maps non-relaxed reads to (`dmb.sy; ldar`). Then there is a top-level pomset $P_2 \in [\![S]\!]_{\mathsf{RR}}$ such that $E_2 = E_1$, $\lambda_2 = \lambda_1$, $\mathsf{rf}_2 = \mathsf{rf}_1$, and $\le_2 = \mathsf{gcb}_1$.*

PROOF. First, we establish some lemmas about Arm8.

LEMMA 2.5. *Suppose G is $(\mathsf{co}, \mathsf{rf}, \mathsf{gcb})$-valid. Then $\mathsf{gcb} \supseteq \mathsf{fr}$.*

PROOF. Using the definition of fr from A6b, we have $e \xleftarrow{\mathsf{rf}} d \xrightarrow{\mathsf{co}} c$, and therefore $\lambda(c)$ blocks $\lambda(e)$. Applying A7a, we have that either $c \xrightarrow{\mathsf{gcb}} d$ or $e \xrightarrow{\mathsf{gcb}} c$. Since gcb includes co, we have $d \xrightarrow{\mathsf{gcb}} c$, and therefore it must be that $e \xrightarrow{\mathsf{gcb}} c$. □

LEMMA 2.6. *Suppose G is $(\mathsf{co}, \mathsf{rf}, \mathsf{gcb})$-valid and $c \xrightarrow{\mathsf{poloc}} e$, where $\lambda(c)$ blocks $\lambda(e)$. Then $c \xrightarrow{\mathsf{gcb}} e$.*

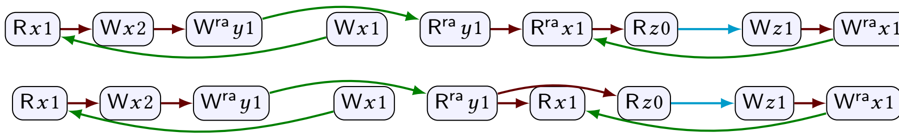
PROOF. By way of contradiction, assume $e \xrightarrow{\mathsf{gcb}} c$. If $c \xrightarrow{\mathsf{rf}} e$ then by A7 we must also have $c \xrightarrow{\mathsf{gcb}} e$, contradicting the assumption that gcb is a total order. Otherwise that there is some $d \neq c$ such that $d \xrightarrow{\mathsf{rf}} e$, and therefore $d \xrightarrow{\mathsf{gcb}} e$. By transitivity, $d \xrightarrow{\mathsf{gcb}} c$. By the definition of fr, we have $e \xrightarrow{\mathsf{fr}} c$. But this contradicts A6b, since $c \xrightarrow{\mathsf{poloc}} e$. □

We show that all the order required in the pomset is also required by Arm8. M7a holds since cb_1 is consistent with co_1 and fr_1. As noted above lob includes the order required by s3 and s7c. We need only show that the order removed from 2.4 can also be removed from the pomset. In order for to remove order from e to c, we must have $d \xrightarrow{\mathsf{rf}} e$ and $d \xrightarrow{\mathsf{poloc}} e$ but not $d \xrightarrow{\mathsf{lob}} c$. Because of our suboptimal lowering, it must be that e is a relaxed read; otherwise the `dmb.sy` would require $d \xrightarrow{\mathsf{lob}} c$. Thus we know that s7c does not require order from e to c. By chaining R4b and W4, any dependence on the read can by satisfied without introducing order in s3. □

[2]We have changed an address dependency in the first thread to a data dependency.

Downgrading messes up publication:

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



2.3 Arm Compilation 2

Definition 2.7. Let $[\![\cdot]\!]_{\mathsf{rf}}$ be as defined for $[\![\cdot]\!]_{\mathsf{RR}}$ in Def 2.3 and Fig 1, with two changes in the definition of sequential composition:

- remove (s7b),
- add (s7d) if $\lambda_1(c)$ blocks $\lambda_2(e)$ then $d \xrightarrow{\mathsf{rf}} e$ implies $c \le d$,
- replace $\bowtie_{\mathsf{co}}$ by $\ltimes_{\mathsf{lws}}$ in Def 1.1 of *delays*, where $\ltimes_{\mathsf{lws}} = \{(\mathsf{W}x, \mathsf{W}x),\ (\mathsf{R}x, \mathsf{W}x)\}$.

Note that Lem 1.3 fails for $[\![\cdot]\!]_{\mathsf{rf}}$, since $d \xrightarrow{\mathsf{rf}} e$ may not imply $d \le e$ when d and e come from different sides of a sequential composition. This means that rf must be verified during pomset construction, rather than post-hoc. If one wants a post-hoc verification technique for rf, it is possible to include program order (po) in the pomset.

Lemma 2.8. *P is top-level iff $d \xrightarrow{\mathsf{rf}} e$ implies either*

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

Theorem 2.9. *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. M7a holds since cb_1 is consistent with co_1 and fr_1. s7b follows from A8b. As noted above, lob includes the order required by s3 and s7c. □

The generality of Thm 2.9 is not limited by the assumption that $\mathsf{cb}_1 \supseteq \mathsf{fr}_1$:

Lemma 2.10. *Suppose G 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 $d \xrightarrow{\mathsf{co}} c$.

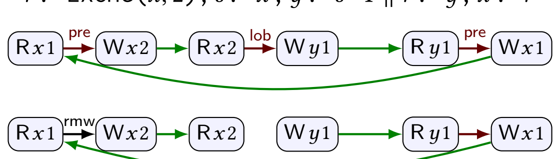
If A8a applies to $d \xrightarrow{\mathsf{rf}} e$, then $e \xrightarrow{\mathsf{cb}} c$, since it cannot be that $c \xrightarrow{\mathsf{co}} d$.

Suppose A8b 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 A8b 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. □

Bad example:

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

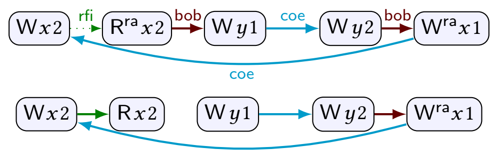


(✓Arm8)

(≤)

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

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



(✓Arm8)

REFERENCES

Jade Alglave, Will Deacon, Richard Grisenthwaite, Antoine Hacquard, and Luc Maranget. 2021. Armed Cats: Formal Concurrency Modelling at Arm. *TOPLAS* (2021). To Appear.

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