A Unified Memory Model for Heterogenous Systems

ANONYMOUS AUTHOR(S)

ACM Reference Format:
Anonymous Author(s). 2022. A Unified Memory Model for Heterogenous Systems. *Proc. ACM Program. Lang.* 0, POPL, Article 0 (January 2022), 24 pages.

1 MODEL

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,
- a set of *thread ids* $\mathcal{T}$, ranged over by α, γ.

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,
- references do not appear in expressions: $M[N/x] = M$,
- thread ids include the *top-level* id $\mathbf{0}$.

We model the following language.

$$\mu, \nu ::= \mathsf{wk} \mid \mathsf{rlx} \mid \mathsf{rel} \mid \mathsf{acq} \mid \mathsf{ra} \mid \mathsf{sc} \qquad\qquad \sigma, \rho ::= \mathsf{cta} \mid \mathsf{gpu} \mid \mathsf{sys}$$

$$\begin{aligned} S ::= {} & \mathtt{skip} \mid r := M \mid r := [L]^{\mu}_{\sigma} \mid [L]^{\mu}_{\sigma} := M \mid \mathsf{F}^{\mu}_{\sigma} \mid \mathtt{if}(M)\{S_1\}\mathtt{else}\{S_2\} \mid S_1 ; S_2 \\ & \mid S_1 \;{}_{\gamma}\!⇻\; S_2 \mid r := \mathtt{CAS}^{\mu,\nu}_{\sigma}([L], M, N) \mid r := \mathtt{FADD}^{\mu,\nu}_{\sigma}([L], M) \mid r := \mathtt{EXCHG}^{\mu,\nu}_{\sigma}([L], M) \end{aligned}$$

Access modes, μ, are weak (wk), relaxed (rlx), release (rel), acquire (acq), release-acquire (ra), and sequentially consistent (sc). Let expressions ($r := M$) only affect thread-local state and thus do not have a mode. Reads ($r := [L]^{\mu}_{\sigma}$) support wk, rlx, acq, sc. Writes ($[L]^{\mu}_{\sigma} := r$) support wk, rlx, rel, sc. Fences ($\mathsf{F}^{\mu}_{\sigma}$) support rel, acq, ra, sc. In the atomic update operations, μ is a read and ν is a write; we require that r does not occur in L.

Scopes, σ, are thread group (cta), processor (gpu) and system (sys).

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. If ($S_1 \;{}_{\gamma}\!⇻\; S_2$) is executed with thread ID α, then S_1 runs with ID γ and S_2 continues under ID α. Top level programs run with thread ID $\mathbf{0}$. In examples, we usually drop thread IDs. We use the symmetric $\parallel$ operator when there is no continuation after the parallel composition.

We use common syntax sugar, such as *extended expressions*, $\mathbb{M}$, which include memory locations. For example, if $\mathbb{M}$ includes a single occurrence of x, then $y := \mathbb{M}; S$ is shorthand for $r := x;$

2475-1421/2022/1-ART0
https://doi.org/

$y := \mathbb{M}[r/x]$; S. Each occurrence of x in an extended expression corresponds to an separate read. We also write $\texttt{if}(M)\{S\}$ as shorthand for $\texttt{if}(M)\{S\}\texttt{else}\{\texttt{skip}\}$.

The semantics is built from the following.

- a set of *events* $\mathcal{E}$, ranged over by e, d, c, and subsets ranged over by E, D, C,
- a set of *logical formulae* Φ, ranged over by ϕ, ψ, θ,
- a set of *actions* $\mathcal{A}$, ranged over by a, b,
- a family of *quiescence symbols* Q_x, indexed by location.

We require that

- registers include $\mathcal{S}_{\mathcal{E}} = \{s_e \mid e \in \mathcal{E}\}$ which do not appear in commands: $S[N/s_e] = S$,
- formulae include tt, ff, Q_x, and the equalities $(M{=}N)$ and $(x{=}M)$,
- formulae are closed under negation, conjunction, disjunction, and substitutions $[M/r]$, $[M/x]$, $[\phi/\mathsf{Q}_x]$,
- there is a relation $\models$ between formulae, capturing entailment,
- $\models$ has the expected semantics for $=$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and substitution.

We relax the first assumption in examples, assuming that each register is assigned at most once.

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

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

We also require that there is a subset of actions, distinguishing *read* actions. We require several binary relations between actions, detailed in the next subsection: *sync-delays*, *co-delays*, *strongly-matches*, *matches*, *blocks*, *overlaps*, *strongly-overlaps*, *strongly-fences*, We require that

$$\text{strongly-matches} \subseteq \text{matches} \subseteq \text{blocks} \subseteq \text{overlaps} \supseteq \text{strongly-overlaps}.$$

1.2 Actions

We combine access and fence modes into a single order:

$$\begin{tikzcd}[row sep=tiny] & & \mathsf{rel} \arrow[dr] & & \\ \mathsf{wk} \arrow[r] & \mathsf{rlx} \arrow[ur] \arrow[dr] & & \mathsf{ra} \arrow[r] & \mathsf{sc} \\ & & \mathsf{acq} \arrow[ur] & & \end{tikzcd}$$

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

Let actions be reads, writes and fences:

$$a, b \ ::= \ \alpha\mathsf{W}^{\mu}_{\sigma}xv \ \mid \ \alpha\mathsf{R}^{\mu}_{\sigma}xv \ \mid \ \alpha\mathsf{F}^{\mu}_{\sigma}$$

In examples, we systematically drop the default mode rlx and the default scope sys. In definitions, we drop elements of actions that are existentially quantified. We write $(\alpha\mathsf{A}^{\mu}_{\sigma}x)$ to stand for an *access*: either $(\alpha\mathsf{W}^{\mu}_{\sigma}x)$ or $(\alpha\mathsf{R}^{\mu}_{\sigma}x)$. We write $(\mathsf{W}^{\sqsupseteq\mathsf{rel}})$ to stand for either $(\mathsf{W}^{\mathsf{rel}})$ or $(\mathsf{W}^{\mathsf{sc}})$, and similarly for other actions and modes.

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 *overlaps* b if $a = (\mathsf{A}x)$ and $b = (\mathsf{A}x)$, regardless of access type or value.

We say a *co-delays* b if $(a, b) \in \{(\mathsf{W}x, \mathsf{W}x),\ (\mathsf{R}x, \mathsf{W}x),\ (\mathsf{W}x, \mathsf{R}x)\} \cup \{(\mathsf{A}^{\mathsf{sc}}, \mathsf{A}^{\mathsf{sc}})\}$.

We say a *sync-delays* b if $(a, b) \in \{(a, \mathsf{W}^{\sqsupseteq\mathsf{rel}}),\ (a, \mathsf{F}^{\sqsupseteq\mathsf{rel}}),\ (\mathsf{R}, \mathsf{F}^{\sqsupseteq\mathsf{acq}}),\ (\mathsf{R}^{\sqsupseteq\mathsf{acq}}, b),\ (\mathsf{F}^{\sqsupseteq\mathsf{acq}}, b),\ (\mathsf{F}^{\sqsupseteq\mathsf{rel}}, \mathsf{W}),\ (\mathsf{W}^{\sqsupseteq\mathsf{rel}}x, \mathsf{W}x)\}$.[1]

Actions (R) are *read* actions.

Definition 1.1. We assume two equivalences: $(=_{\mathsf{gpu}}) \subseteq (\mathcal{T} \times \mathcal{T})$ partitions threads by *processor*, and $(=_{\mathsf{cta}}) \subseteq (=_{\mathsf{gpu}})$ refines the processor partitioning into *thread groups*.

[1] For PTX, one could additionally include $(\mathsf{R}x, \mathsf{R}^{\sqsupseteq\mathsf{acq}}x)$, but this is not sound for Arm or IMM.

We say $(\alpha_1 \mathsf{A}^{\mu_1}_{\sigma_1} x)$ *strongly-overlaps* $(\alpha_2 \mathsf{A}^{\mu_2}_{\sigma_2} x)$ when either

(1) $\alpha_1 = \alpha_2$, or
(2a) $\mu_1, \mu_2 \sqsupseteq \mathsf{rlx}$,
(2b) if $\sigma_1 = \mathsf{cta}$ or $\sigma_2 = \mathsf{cta}$ then $\alpha_1 =_{\mathsf{cta}} \alpha_2$,
(2c) if $\sigma_1 = \mathsf{gpu}$ or $\sigma_2 = \mathsf{gpu}$ then $\alpha_1 =_{\mathsf{gpu}} \alpha_2$.

We say $(\alpha_1 \mathsf{F}^{\mu_1}_{\sigma_1})$ *strongly-fences* $(\alpha_2 \mathsf{F}^{\mu_2}_{\sigma_2})$ when $\mu_1 = \mu_2 = \mathsf{sc}$ and either (1) or (2) apply, from the definition of strongly-overlaps.

We say a *strongly-matches* b when a is a release, b is an acquire, and either a strongly-overlaps b or a strongly-fences b.

Note that for a CPUs, all action have scope sys and mode rlx or greater. For this subset of actions, *strongly-overlaps* is the same as *overlaps* and *strongly-fences* applies to any pair of sc fences.

1.3 Pomsets with Predicate Transformers

Definition 1.2. Let $\theta_\lambda = \bigwedge_{\{(e,v)\in(E\times\mathcal{V}) \mid \lambda(e)=(\mathsf{R}\,v)\}} (s_e{=}v)$ where $E = \mathsf{dom}(\lambda)$.
We say that ϕ is *λ-consistent* if $\phi \land \theta_\lambda$ is satisfiable. We say that it is *λ-inconsistent* otherwise.

Definition 1.3. A *λ-predicate transformer* is a function $\tau : \Phi \longrightarrow \Phi$ such that

(x1) $\tau(\psi_1 \land \psi_2) \equiv \tau(\psi_1) \land \tau(\psi_2)$,
(x2) $\tau(\psi_1 \lor \psi_2) \equiv \tau(\psi_1) \lor \tau(\psi_2)$,
(x3) if $\phi \vDash \psi$, then $\tau(\phi) \vDash \tau(\psi)$,
(x4) if ψ is λ-inconsistent then $\tau(\psi)$ is λ-inconsistent.

Definition 1.4. A *family of λ-predicate transformers* consists of a λ-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 $\tau(\psi)$ as an abbreviation of $\tau^E(\psi)$.

Definition 1.5. A *pomset with predicate transformers* is a tuple $(E, \lambda, \kappa, \tau, \checkmark, \trianglelefteq, \leq, \sqsubseteq, \mathsf{rmw})$ 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, such that
  (M3a) $\kappa(e)$ is λ-consistent,
(M4) $\tau : 2^{\mathcal{E}} \longrightarrow \Phi \longrightarrow \Phi$ is a *family of λ-predicate transformers*,
(M5) $\checkmark : \Phi$ is a *termination condition*, such that
  (M5a) $\checkmark \vDash \tau(\mathsf{tt})$,
(M6) $\trianglelefteq : (E \times E)$ is a partial order capturing *dependency*,
(M7) $\leq : (E \times E)$ is a partial order capturing *synchronization*,
(M8) $\sqsubseteq : (E \times E)$ is a partial order capturing *per-location order*, such that
  (M8a) if $\lambda(d)$ overlaps $\lambda(e)$ then $d \leq e$ implies $d \sqsubseteq e$,
(M9) $\mathsf{rmw} : E \longrightarrow E$ is a partial function capturing read-modify-write *atomicity*, such that
  (M9a) if $d \xrightarrow{\mathsf{rmw}} e$ then $\lambda(e)$ blocks $\lambda(d)$,
  (M9b) if $d \xrightarrow{\mathsf{rmw}} e$ then $d \leq e$ and $d \sqsubseteq e$,
  (M9c) if $\lambda(c)$ overlaps $\lambda(d)$ then
    (i) if $d \xrightarrow{\mathsf{rmw}} e$ then $c \trianglelefteq e$ implies $c \trianglelefteq d$, $c \leq e$ implies $c \leq d$, $c \sqsubseteq e$ implies $c \sqsubseteq d$,
    (ii) if $d \xrightarrow{\mathsf{rmw}} e$ then $d \trianglelefteq c$ implies $e \trianglelefteq c$, $d \leq c$ implies $e \leq c$, $d \sqsubseteq c$ implies $e \sqsubseteq c$.

A pomset is a *candidate* if there is an injective relation $\mathsf{rf} : E \times E$, capturing *reads-from*, such that

(c2a) if $d \xrightarrow{\mathsf{rf}} e$ then $\lambda(d)$ matches $\lambda(e)$,
(c6) if $d \xrightarrow{\mathsf{rf}} e$ then $d \trianglelefteq e$,
(c7a) if $d' \leq d \xrightarrow{\mathsf{rf}} e \leq e'$ and $\lambda(d')$ strongly-matches $\lambda(e')$ then $d' \leq e'$,
(c7b) if $\lambda(d)$ strongly-fences $\lambda(e)$ then either $d \leq e$ or $e \leq d$, [Todo: Is this right?]
(c8a) if $d \xrightarrow{\mathsf{rf}} e$ then $d \sqsubseteq e$,

(c8b) if $d \xrightarrow{\mathsf{rf}} e$ and $\lambda(c)$ blocks $\lambda(e)$ then either $c \underset{\sim}{\sqsubseteq} d$ or $e \underset{\sim}{\sqsubseteq} c$, where $d' \underset{\sim}{\sqsubseteq} e'$ when $e' \sqsubseteq d'$ implies $d' = e'$ and $\lambda(d')$ strongly-overlaps $\lambda(e')$ implies $d' \sqsubseteq e'$.

A candidate pomset with rf is *complete* if

(c2b) if $\lambda(e)$ is a read then there is some $d \xrightarrow{\mathsf{rf}} e$,
(c3) $\kappa(e)$ is a tautology (for every $e \in E$),
(c5) $\checkmark$ is a tautology.

Note that for the IMM model, c8b is equivalent to:[2]

$$\text{if } d \xrightarrow{\mathsf{rf}} e \text{ and } \lambda(c) \text{ blocks } \lambda(e) \text{ then either } c \sqsubseteq d \text{ or } e \sqsubseteq c.$$

Let P range over pomsets, and $\mathcal{P}$ over sets of pomsets.

We drop quantifiers when clear from context, such as $(\forall e \in E)(\forall x \in \mathcal{X})$. We write $d < e$ when $d \le e$ and $d \neq e$, and similarly for $\lhd$ and $\sqsubset$. We sometimes use projection functions—for example, if $\lambda(e) = \alpha\mathsf{W}^{\mu}_{\sigma}xv$ then $\lambda_{\mathsf{thrd}}(e) = \alpha$, $\lambda_{\mathsf{mode}}(e) = \mu$, $\lambda_{\mathsf{scope}}(e) = \sigma$, $\lambda_{\mathsf{loc}}(e) = x$, $\lambda_{\mathsf{val}}(e) = v$.

1.4 Semantics

See Fig. 1.

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:

- $d \rightarrow e$ arises from control/data/address *dependency* (s3, definition of $\kappa'_2(d)$),
- $d \rightarrow e$ arises from *sync-delays* (s7a),
- $d \rightarrow e$ arises from *co-delays* (s8a),
- $d \rightarrow e$ arises from *matching* (c6), (c7a) and (c8a),
- $d \rightarrow e$ arises from *strong fencing* (c7b),
- $d \rightarrow e$ arises from *blocking* (c8b).

1.5 Address Calculation

Definition 1.6. If $P \in \mathit{WRITE}(L, M, \mu, \sigma)_\alpha$ then $(\exists \ell \in \mathcal{V})$ $(\exists v \in \mathcal{V})$

(w1) if $|E| \leqslant 1$,
(w2) $\lambda(e) = \alpha\mathsf{W}^{\mu}_{\sigma}[\ell]v$,
(w3) $\kappa(e) \equiv L{=}\ell \wedge M{=}v$,
(w4a) if $E \neq \emptyset$ then $\tau^D(\psi) \equiv (L{=}\ell) \Rightarrow \psi[M/[\ell]][M{=}v/\mathsf{Q}_{[\ell]}]$,
(w4b) if $E = \emptyset$ then $(\forall k)$ $\tau^D(\psi) \equiv (L{=}k) \Rightarrow \psi[M/[k]][\mathsf{ff}/\mathsf{Q}_{[k]}]$,
(w5a) if $E \neq \emptyset$ then $\checkmark \equiv L{=}\ell \wedge M{=}v$,
(w5b) if $E = \emptyset$ then $\checkmark \equiv \mathsf{ff}$.

If $P \in \mathit{READ}(r, L, \mu, \sigma)_\alpha$ then $(\exists \ell \in \mathcal{V})$ $(\exists v \in \mathcal{V})$

(R1) if $|E| \leqslant 1$,
(R2) $\lambda(e) = \alpha\mathsf{R}^{\mu}_{\sigma}[\ell]v$
(R3) $\kappa(e) \equiv L{=}\ell \wedge \mathsf{Q}_{[\ell]}$,
(R4a) if $e \in E \cap D$ then $\tau^D(\psi) \equiv (L{=}\ell \Rightarrow v{=}s_e) \Rightarrow \psi[s_e/r]$,
(R4b) if $e \in E \setminus D$ then $\tau^D(\psi) \equiv ((L{=}\ell \Rightarrow v{=}s_e) \vee (L{=}\ell \Rightarrow [\ell]{=}s_e)) \Rightarrow \psi[s_e/r]$,
(R4c) if $E = \emptyset$ then $(\forall s)$ $\tau^D(\psi) \equiv \psi[s/r]$,
(R5a) if $E \neq \emptyset$ or $\mu \sqsubseteq \mathsf{rlx}$ then $\checkmark \equiv \mathsf{tt}$.
(R5b) if $E = \emptyset$ and $\mu \sqsupseteq \mathsf{acq}$ then $\checkmark \equiv \mathsf{ff}$.

[2] If all accesses are morally strong with each other, weak fulfillment degenerates to

$$\forall\lambda(c) = (\mathsf{W}x) \text{ either } c \sqsubseteq d \text{ or } e \sqsubseteq c$$

If no accesses are morally strong with each other, weak fulfillment degenerates to

$$\nexists\lambda(c) = (\mathsf{W}x) \text{ both } d \sqsubset c \text{ and } c \sqsubset e$$

Note that the difference between strong and weak fulfillment is limited to $\sqsubseteq$. We sometimes write $\overset{\mathsf{s}}{\sqsubseteq}$ for strong fulfillment and $\overset{\mathsf{w}}{\sqsubseteq}$ for weak fulfillment.

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

If $P \in \mathit{PAR}(\mathcal{P}_1, \mathcal{P}_2)$ 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 $\kappa(e) \equiv \kappa_1(e)$,
(P3b) if $e \in E_2$ then $\kappa(e) \equiv \kappa_2(e)$,
(P4) $\tau^D(\psi) \equiv \tau_2^D(\psi)$,
(P5) $\checkmark \equiv \checkmark_1 \wedge \checkmark_2$,
(P6) $\trianglelefteq\ \supseteq (\trianglelefteq_1 \cup \trianglelefteq_2)$,
(P7) $\leq\ \supseteq (\leq_1 \cup \leq_2)$,
(P8) $\sqsubseteq\ \supseteq (\sqsubseteq_1 \cup \sqsubseteq_2)$,
(P9) $\mathsf{rmw} = (\mathsf{rmw}_1 \cup \mathsf{rmw}_2)$.

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

(S1) $E = (E_1 \cup E_2)$,
(S2) (S6) (S7) (S8) (S9) as for *PAR*,
(S3a) if $e \in E_1 \setminus E_2$ then $\kappa(e) \equiv \kappa_1(e)$,
(S3b) if $e \in E_2 \setminus E_1$ then $\kappa(e) \equiv \kappa'_2(e)$,
(S3c) if $e \in E_1 \cap E_2$ then $\kappa(e) \equiv \kappa_1(e) \vee \kappa'_2(e)$,
(S4) $\tau^D(\psi) \equiv \tau_1^D(\tau_2^D(\psi))$,
(S5) $\checkmark \equiv \checkmark_1 \wedge \tau_1(\checkmark_2)$,
(S7a) if $\lambda_1(d)$ sync-delays $\lambda_2(e)$ then $d \leq e$,
(S8a) if $\lambda_1(d)$ co-delays $\lambda_2(e)$ then $d \sqsubseteq e$,

where $\kappa'_2(e) = \tau_1(\kappa_2(e))$ if $\lambda(e)$ is a read—otherwise $\kappa'_2(e) = \tau_1^{\downarrow e}(\kappa_2(e))$, where $\downarrow e = \{c \mid c \triangleleft e\}$.

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

(I1) $E = (E_1 \cup E_2)$,
(I2) (I6) (I7) (I8) (I9) as for *PAR*,
(I3a) if $e \in E_1 \setminus E_2$ then $\kappa(e) \equiv \phi \wedge \kappa_1(e)$,
(I3b) if $e \in E_2 \setminus E_1$ then $\kappa(e) \equiv \neg\phi \wedge \kappa_2(e)$,
(I3c) if $e \in E_1 \cap E_2$ then $\kappa(e) \equiv (\phi \wedge \kappa_1(e)) \vee (\neg\phi \wedge \kappa_2(e))$,
(I4) $\tau^D(\psi) \equiv (\phi \wedge \tau_1^D(\psi)) \vee (\neg\phi \wedge \tau_2^D(\psi))$,
(I5) $\checkmark \equiv (\phi \wedge \checkmark_1) \vee (\neg\phi \wedge \checkmark_2)$.

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

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

(W1) $|E| \leqslant 1$,
(W2) $\lambda(e) = \alpha\mathsf{W}^\mu_\sigma x v$,
(W3) $\kappa(e) \equiv M{=}v$,
(W4a) if $E \neq \emptyset$ then $\tau^D(\psi) \equiv \psi[M/x][M{=}v/\mathsf{Q}_x]$,
(W4b) if $E = \emptyset$ then $\tau^D(\psi) \equiv \psi[M/x][\mathsf{ff}/\mathsf{Q}_x]$,
(W5a) if $E \neq \emptyset$ then $\checkmark \equiv M{=}v$.
(W5b) if $E = \emptyset$ then $\checkmark \equiv \mathsf{ff}$,

If $P \in \mathit{READ}(r, x, \mu, \sigma)_\alpha$ then $(\exists v \in \mathcal{V})$

(R1) $|E| \leqslant 1$,
(R2) $\lambda(e) = \alpha\mathsf{R}^\mu_\sigma x v$,
(R3) $\kappa(e) \equiv \mathsf{Q}_x$,
(R4a) if $e \in E \cap D$ then $\tau^D(\psi) \equiv v{=}s_e \Rightarrow \psi[s_e/r]$,
(R4b) if $e \in E \setminus D$ then $\tau^D(\psi) \equiv (v{=}s_e \vee x{=}s_e) \Rightarrow \psi[s_e/r]$,
(R4c) if $E = \emptyset$ then $(\forall s)\tau^D(\psi) \equiv \psi[s/r]$,
(R5a) if $E \neq \emptyset$ or $\mu \sqsubseteq \mathsf{rlx}$ then $\checkmark \equiv \mathsf{tt}$.
(R5b) if $E = \emptyset$ and $\mu \sqsupseteq \mathsf{acq}$ then $\checkmark \equiv \mathsf{ff}$.

If $P \in \mathit{FENCE}(\mu, \sigma)_\alpha$ then

(F1) $|E| \leqslant 1$,
(F2) $\lambda(e) = \alpha\mathsf{F}^\mu_\sigma$,
(F3) $\kappa(e) \equiv \mathsf{tt}$,
(F4) $\tau^D(\psi) \equiv \psi$,
(F5a) if $E \neq \emptyset$ then $\checkmark \equiv \mathsf{tt}$,
(F5b) if $E = \emptyset$ then $\checkmark \equiv \mathsf{ff}$.

$$
\begin{aligned}
\llbracket r := M \rrbracket_\alpha &= \mathit{LET}(r, M) & \llbracket \mathtt{skip} \rrbracket_\alpha &= \mathit{SKIP} \\
\llbracket r := x^\mu_\sigma \rrbracket_\alpha &= \mathit{READ}(r, x, \mu, \sigma)_\alpha & \llbracket S_1 \ {}_{\gamma}{\Vert\!\!\rightarrow}\ S_2 \rrbracket_\alpha &= \mathit{PAR}(\llbracket S_1 \rrbracket_\gamma, \llbracket S_2 \rrbracket_\alpha) \\
\llbracket x^\mu_\sigma := M \rrbracket_\alpha &= \mathit{WRITE}(x, M, \mu, \sigma)_\alpha & \llbracket S_1 \,;\, S_2 \rrbracket_\alpha &= \mathit{SEQ}(\llbracket S_1 \rrbracket_\alpha, \llbracket S_2 \rrbracket_\alpha) \\
\llbracket \mathsf{F}^\mu_\sigma \rrbracket_\alpha &= \mathit{FENCE}(\mu, \sigma)_\alpha & \llbracket \mathtt{if}(M)\{S_1\}\mathtt{else}\{S_2\} \rrbracket_\alpha &= \mathit{IF}(M{\neq}0, \llbracket S_1 \rrbracket_\alpha, \llbracket S_2 \rrbracket_\alpha)
\end{aligned}
$$

Fig. 1. Semantics of programs

1.6 If-closure

Definition 1.7. Let $E \subseteq \mathcal{E}$ and $\theta : E \to \Phi$ and $\Omega \in \Phi$. We say that θ *partitions* Ω if (1) if $\theta_e \wedge \theta_d$ is satisfiable then $e = d$, (2) $\Omega \equiv \bigvee_{e \in E} \theta_e$.

If $P \in \mathit{WRITE}(x,\ M,\ \mu,\ \sigma)_\alpha$ then $(\exists v : E \to \mathcal{V})\ (\exists \theta : E \to \Phi)\ (\exists \Omega \in \{\mathsf{tt}, \mathsf{ff}\})$

(w1) θ partitions Ω,

(w2) $\lambda(e) = \alpha \mathsf{W}^{\mu}_{\sigma} x v_e$,

(w3) $\kappa(e) \equiv \theta_e \wedge M{=}v_e$,

(w4) $\tau^D(\psi) \equiv \bigwedge_{e \in E}(\theta_e \Rightarrow \psi[M/x][M{=}v_e/\mathsf{Q}_x])$
$\wedge \neg\Omega \Rightarrow \psi[M/x][\mathsf{ff}/\mathsf{Q}_x]$

(w5) $\checkmark \equiv \Omega \wedge \bigwedge_{e \in E}(\theta_e \Rightarrow M{=}v_e)$.

If $P \in \mathit{READ}(r,\ x,\ \mu,\ \sigma)_\alpha$ then $(\exists v : E \to \mathcal{V})\ (\exists \theta : E \to \Phi)\ (\exists \Omega \in \{\mathsf{tt}, \mathsf{ff}\})$

(R1) θ partitions Ω,

(R2) $\lambda(e) = \alpha \mathsf{R}^{\mu}_{\sigma} x v_e$

(R3) $\kappa(e) \equiv \theta_e \wedge \mathsf{Q}_x$,

(R4) $(\forall s)\tau^D(\psi) \equiv \bigwedge_{e \in E \cap D}(\theta_e \Rightarrow v_e{=}s_e \Rightarrow \psi[s_e/r])$
$\wedge \bigwedge_{e \in E \setminus D}(\theta_e \Rightarrow (v_e{=}s_e \vee x{=}s_e) \Rightarrow \psi[s_e/r])$
$\wedge \neg\Omega \Rightarrow \psi[s/r]$

(R5a) if $\mu \sqsubseteq \mathsf{rlx}$ then $\checkmark \equiv \mathsf{tt}$,

(R5b) if $\mu \sqsupseteq \mathsf{acq}$ then $\checkmark \equiv \Omega$.

1.7 Address Calculation and If-closure

Definition 1.8. If $P \in \mathit{WRITE}(L,\ M,\ \mu,\ \sigma)_\alpha$ then $(\exists \ell : E \to \mathcal{V})\ (\exists v : E \to \mathcal{V})\ (\exists \theta : E \to \Phi)$ $(\exists \Omega \in \{\mathsf{tt}, \mathsf{ff}\})$

(w1) θ partitions Ω,

(w2) $\lambda(e) = \alpha \mathsf{W}^{\mu}_{\sigma}[\ell] v_e$,

(w3) $\kappa(e) \equiv \theta_e \wedge L{=}\ell_e \wedge M{=}v_e$,

(w4) $(\forall k)\tau^D(\psi) \equiv \bigwedge_{e \in E}(\theta_e \Rightarrow (L{=}\ell) \Rightarrow \psi[M/x][M{=}v_e/\mathsf{Q}_x])$
$\wedge \neg\Omega \Rightarrow (L{=}k) \Rightarrow \psi[M/x][\mathsf{ff}/\mathsf{Q}_x]$

(w5) $\checkmark \equiv \Omega \wedge \bigwedge_{e \in E}(\theta_e \Rightarrow L{=}\ell_e \wedge M{=}v_e)$.

If $P \in \mathit{READ}(r,\ L,\ \mu,\ \sigma)_\alpha$ then $(\exists \ell : E \to \mathcal{V})\ (\exists v : E \to \mathcal{V})\ (\exists \theta : E \to \Phi)\ (\exists \Omega \in \{\mathsf{tt}, \mathsf{ff}\})$

(R1) θ partitions Ω,

(R2) $\lambda(e) = \alpha \mathsf{R}^{\mu}_{\sigma}[\ell] v_e$

(R3) $\kappa(e) \equiv \theta_e \wedge L{=}\ell_e \wedge \mathsf{Q}_{[\ell]}$,

(R4) $(\forall s)\tau^D(\psi) \equiv \bigwedge_{e \in E \cap D}(\theta_e \Rightarrow (L{=}\ell_e \Rightarrow v_e{=}s_e) \Rightarrow \psi[s_e/r])$
$\wedge \bigwedge_{e \in E \setminus D}(\theta_e \Rightarrow ((L{=}\ell_e \Rightarrow v_e{=}s_e) \vee (L{=}\ell_e \Rightarrow [\ell]{=}s_e)) \Rightarrow \psi[s_e/r])$
$\wedge \neg\Omega \Rightarrow \psi[s/r]$,

(R5a) if $\mu \sqsubseteq \mathsf{rlx}$ then $\checkmark \equiv \mathsf{tt}$,

(R5b) if $\mu \sqsupseteq \mathsf{acq}$ then $\checkmark \equiv \Omega$.

Definition 1.9. Let READ' be defined as for READ, adding the constraint:

(R4d) if $(E \cap D) = \emptyset$ then $\tau^D(\psi) \equiv \psi$.

If $P \in \mathit{FADD}(r, L, M, \mu, \nu)$ then $(\exists P_1 \in \mathit{SEQ}(\mathit{READ}'(r,\ L,\ \mu),\ \mathit{WRITE}(L,\ r{+}M,\ \nu)))$

(U1) if $\lambda_1(e)$ is a write then there is a read $\lambda_1(d)$ such that $\kappa(e) \vDash \kappa(d)$ and $d \xrightarrow{\mathsf{rmw}} e$.

If $P \in \mathit{EXCHG}(r, L, M, \mu, \nu)$ then $(\exists P_1 \in \mathit{SEQ}(\mathit{READ}'(r,\ L,\ \mu),\ \mathit{WRITE}(L,\ M,\ \nu)))$

(U1) if $\lambda_1(e)$ is a write then there is a read $\lambda_1(d)$ such that $\kappa(e) \vDash \kappa(d)$ and $d \xrightarrow{\mathsf{rmw}} e$.

If $P \in \mathit{CAS}(r, L, M, N, \mu, \nu)$ then $(\exists P_1 \in \mathit{SEQ}(\mathit{READ}'(r,\ L,\ \mu),\ \mathit{IF}(r{=}M,\ \mathit{WRITE}(L,\ N,\ \nu),\ \mathit{SKIP})))$

(U1) if $\lambda_1(e)$ is a write then there is a read $\lambda_1(d)$ such that $\kappa(e) \vDash \kappa(d)$ and $d \xrightarrow{\mathsf{rmw}} e$.

2 DEAD STORE ELIMINATION, STORE FORWARDING, AND MONOTONICITY

We validate "monotonicity" by updating the rules for read, write and fence to include $(\exists \nu \sqsupseteq \mu)$:

(R2) $\lambda(e) = \alpha \mathsf{R}^{\nu}_{\sigma} x v$, (W2) $\lambda(e) = \alpha \mathsf{W}^{\nu}_{\sigma} x v$, (F2) $\lambda(e) = \alpha \mathsf{F}^{\nu}_{\sigma}$.

One could do the same for scopes.

The semantics already validates:

- $[\![x := M;\, x := M]\!] \supseteq [\![x := M]\!]$

- $[\![s := x\,;\, r := x]\!] \supseteq [\![s := x\,;\, r := s]\!]$
- $[\![r := x]\!] \supseteq [\![\mathsf{skip}]\!]$

It does not validate:

- $[\![x := M;\, x := N]\!] \supseteq [\![x := N]\!]$
- $[\![x := M;\, r := x]\!] \supseteq [\![x := M;\, r := M]\!]$

The semantics of Fig. 1 validates elimination of irrelevant relaxed reads and redundant reads. Fig. 1 also validates elimination of writes of the same value. However, Fig. 1 does not validate general write elimination, where, for example, $(x := 1;\, x := 2)$ is refined to $x := 2$. Nor does it validate store forwarding, where, for example, $(x := 1;\, r := x)$ is refined to $(x := 1;\, r := 1)$.

Elimination can be justified in pomset by *merging* actions with different labels. A list of safe merges can be found in [Chakraborty and Vafeiadis 2017, §E] and [Kang 2019, §7.1]. For examples of unsafe merges and reorderings, see [Chakraborty and Vafeiadis 2017, §D]. See also [Chakraborty and Vafeiadis 2019, §6.2]

Read-read and fence-fence merges can be handled by "monotonicity": allowing actions to put down stronger modes in the model. Then they can merge on the nose.

Sad: read elimination can't be done the nice way using $\tau^D(\psi) \equiv x{=}r \Rightarrow \psi$ for R4c because there may be a release-acquire pair between the read and the matching write.

Let merge : $\mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$ be a partial function defined as follows.

$$\mathsf{merge}(a,\ b) = \begin{cases} a & \text{if } a = b \text{ or } a = (\alpha \mathsf{W}^{\mu}_{\sigma} x v) \text{ and } b = (\alpha \mathsf{R}^{\nu}_{\sigma} x v) \\ b & \text{if } a = b \text{ or } a = (\alpha \mathsf{W}^{\mu}_{\sigma} x v) \text{ and } b = (\alpha \mathsf{W}^{\nu}_{\sigma} x w) \\ \text{undefined} & \text{otherwise} \end{cases}$$

(If we have "monotonicity" then we can require $\mu = \nu$.)

If $a_0 = \mathsf{merge}(a_1,\ a_2)$, then a_1 and a_2 can coalesce, resulting in a_0. This allows optimizations such as $(x := 1;\, x := 2)$ to $(x := 2)$ and $(x := 1;\, r := x)$ to $(x := 1;\, r := 1)$. For associativity of sequential composition, it is important that merge always take an upper bound on the modes of the two actions. For example, it would invalidate associativity to allow $(\mathsf{W}xv) = \mathsf{merge}(\mathsf{W}xv,\ \mathsf{R}^{\mathsf{acq}}xv)$, although this is considered safe.

Then we can replace s2-s3 in Fig. 1 by:

(s2a) if $e \in E_1 \setminus E_2$ then $\lambda(e) = \lambda_1(e)$,
(s2b) if $e \in E_2 \setminus E_1$ then $\lambda(e) = \lambda_2(e)$,
(s2c) if $e \in E_1 \cap E_2$ then $\lambda(e) = \mathsf{merge}(\lambda_1(e),\ \lambda_2(e))$,
(s3a) if $e \in E_1 \setminus E_2$ then $\kappa(e) \equiv \kappa_1(e)$,
(s3b) if $e \in E_2 \setminus E_1$ then $\kappa(e) \equiv \kappa'_2(e)$,
(s3c) if $e \in E_1 \cap E_2$ then either

- $\lambda_1(e) = \lambda(e) = \lambda_2(e)$ and $\kappa(e) \equiv \kappa_1(e) \vee \kappa'_2(e)$,
- $\lambda_1(e) = \lambda(e) \neq \lambda_2(e)$ and $\kappa'_2(e) \equiv \kappa(e) \equiv \kappa_1(e)$ (write-read),
- $\lambda_1(e) \neq \lambda(e) = \lambda_2(e)$ and $\kappa_1(e) \equiv \kappa(e) \equiv \kappa'_2(e)$ (write-write).

Full merge: $\mathtt{if}(M)\{x := 1\};\ x := 2$ can become $x := 2$.

Partial merge: $x := 1;\ \mathtt{if}(M)\{x := 2\}$ can become $\mathtt{if}(M)\{x := 2\}\,\mathtt{else}\,\{x := 1\}$.

To get associativity, you need the ability to merge with multiple events.

$x := 1;\ \mathtt{if}(M)\{x := 2\}$ $\qquad\qquad$ $\mathtt{if}(!M)\{x := 2\}$

¬M | Wx1
M | Wx2
¬M | Wx2

This is asymmetric. We don't expect to merge all three events in the following:

$$\begin{array}{cc} \texttt{if(!}M\texttt{)\{}x\texttt{:=2\}} & x\texttt{:=1; if(}M\texttt{)\{}x\texttt{:=2\}} \\ \boxed{\neg M \mid \mathsf{W}x2} & \boxed{\neg M \mid \mathsf{W}x1}\ \boxed{M \mid \mathsf{W}x2} \end{array}$$

We could have a lot merging:

$$\begin{array}{cc} \texttt{if(}N\texttt{)\{}x\texttt{:=1; if(}M\texttt{)\{}x\texttt{:=3\}\}; if(}\neg N\texttt{)\{}x\texttt{:=2; if(}M\texttt{)\{}x\texttt{:=3\}\}} & \texttt{if(!}M\texttt{)\{}x\texttt{:=3\}} \\ \boxed{\neg M \land N \mid \mathsf{W}x1}\ \boxed{M \land N \mid \mathsf{W}x3}\ \boxed{\neg M \land \neg N \mid \mathsf{W}x2}\ \boxed{M \land \neg N \mid \mathsf{W}x3} & \boxed{\neg M \mid \mathsf{W}x3} \end{array}$$

Full merge: x `:=1; if(`M`){`r`:=`x`}` can become x `:=1; if(`M`){`r`:=1}`.

Partial merge: `if(`M`){`x`:=1};` r`:=`x can become `if(`M`){`x`:=1;` r`:=1} else {`r`:=`x`}`.

I don't think we need multi-merge for write-read. Reads only affect the world via the predicate transformer. Any conditional surrounding a read is baked into the predicate transformer, and so does not to persist in the preconditions of the actions themselves after the merge. Consider r`:=1;` x`:=2; if(`M`){`r`:=`x`}`. This can safely transform to r`:=1;` x`:=2; if(`M`){`r`:=2}`.

In the example below, the reads should *not* merge. Although the second read can merge with the write.

$$\begin{array}{cc} \texttt{if(!}M\texttt{)\{}x\texttt{:=1\}; if(}M\texttt{)\{}r\texttt{:=}x\texttt{\}} & \texttt{if(!}M\texttt{)\{}s\texttt{:=}x\texttt{\}} \\ \boxed{\neg M \mid \mathsf{W}x1}\ \boxed{M \mid \mathsf{R}x1} & \boxed{\neg M \mid \mathsf{R}x1} \end{array}$$

Another example:

$$\begin{array}{cc} x\texttt{:=1; if(}M\texttt{)\{}r\texttt{:=}x\texttt{\}} & \texttt{if(!}M\texttt{)\{}s\texttt{:=}x\texttt{\}} \\ \boxed{\mathsf{W}x1} & \boxed{\neg M \mid \mathsf{R}x1} \end{array}$$

Another example:

$$\begin{array}{cc} x\texttt{:=1} & \texttt{if(}M\texttt{)\{}r\texttt{:=}x\texttt{\}; if(!}M\texttt{)\{}s\texttt{:=}x\texttt{\}} \\ \boxed{\mathsf{W}x1} & \boxed{\mathsf{R}x1} \end{array}$$

Idea for multi-merge. Use $E_1' \subseteq E_1$, with a surjective function $\pi : E_1 \longrightarrow E_1'$ that shows how writes merge.

- Require that $(\forall d \in E_1')\ \pi(d) = d$.
- Require that if $c \in (E_1 \setminus E_1')$ then $\pi(c) \in E_2$—and therefore $\pi(c) \in (E_1' \cap E_2)$.
- Take $E = E_1' \cup E_2$.

Require that the writes that coalesce have disjoint preconditions.

- if $\pi(c) = \pi(c')$ then $\kappa_1(c) \land \kappa_1(c')$ is unsatisfiable

Then each of them has to merge into the same write $e \in E_2$ using the merge function and combining the predicates as specified above.

(s2a) if $e \in E_1' \setminus E_2$ then $\lambda(e) = \lambda_1(e)$,
(s2b) if $e \in E_2 \setminus E_1'$ then $\lambda(e) = \lambda_2(e)$,
(s2c) if $e \in (E_1' \cap E_2)$ and $c \in E_1$ and $\pi(c) = e$ then $\lambda(e) = \mathsf{merge}(\lambda_1(c),\ \lambda_2(e))$,
(s3a) if $e \in E_1' \setminus E_2$ then $\kappa(e) \equiv \kappa_1(e)$,
(s3b) if $e \in E_2 \setminus E_1'$ then $\kappa(e) \equiv \kappa_2'(e)$,
(s3c) if $e \in (E_1' \cap E_2)$ then

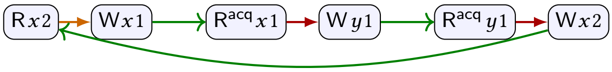
- $\kappa(e) \equiv \kappa_2'(e) \lor \bigvee_{c \in C} \kappa_1(c)$, where $C = \{c \in E_1 \mid \pi(c) = e \text{ and } \lambda_1(c) = \lambda_2(e)\}$,
- if $\pi(c) = e$ and $\lambda_1(c) = \lambda(e) \neq \lambda_2(e)$ then $\kappa_2'(c) \equiv \kappa(e)$ (write-read),
- if $\pi(c) = e$ and $\lambda_1(c) \neq \lambda(e) = \lambda_2(e)$ then $\kappa_1(c) \equiv \kappa(e)$ (write-write).

3 EXAMPLE FROM JAM PAPER

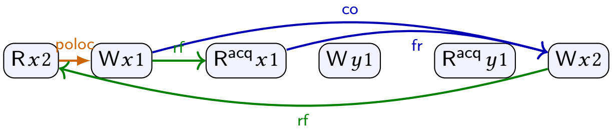
From [Bender and Palsberg 2019, §3.3]. With partial coherence/weak fulfillment you need to be careful that RMWs are totally ordered (if that's a property you want). May not come for free.

From [Bender and Palsberg 2019, §B]: "Here we demonstrate that it is possible to construct a program that is only forbidden due to the total coherence order"

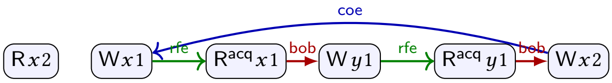
$$r := x\,;\ x := 1 \parallel r := x^{\mathsf{acq}}\,;\ y := 1 \parallel r := y^{\mathsf{acq}}\,;\ x := 2$$ (TOTAL-CO)



(✗Arm8)



(✗Arm8)

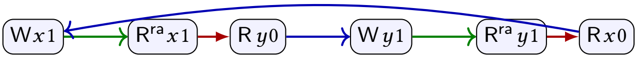


(✗Arm8)

4 IRIW

Status of IRIW is unclear in our model, since we allow everything allowed by power...

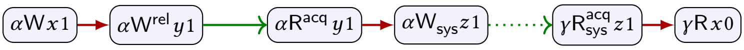
$$x := 1 \parallel r := x^{\mathsf{ra}}\,;\ s := y \parallel y := 1 \parallel s := y^{\mathsf{ra}}\,;\ r := x$$



5 SYNC EXAMPLES

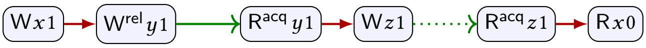
The first of these is seen in hardware. All are allowed by PTX. Showing rf that is not included in the order using a dotted arrow.

$$x := 1;\ y^{\mathsf{rel}} := 1 \parallel r := y^{\mathsf{acq}};\ z_{\mathsf{sys}} := r \mathrel{\leftarrow\!\!\parallel}_{\gamma} r := z^{\mathsf{acq}}_{\mathsf{sys}};\ s := x$$



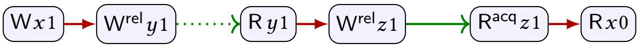
($\leq$)

$$x := 1;\ y^{\mathsf{rel}} := 1 \parallel r := y^{\mathsf{acq}};\ z := r \parallel r := z^{\mathsf{acq}};\ s := x$$



($\leq$)

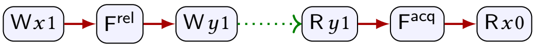
$$x := 1;\ y^{\mathsf{rel}} := 1 \parallel r := y;\ z^{\mathsf{rel}} := r \parallel r := z^{\mathsf{acq}};\ s := x$$



($\leq$)

To get publication using fences we need an additional closure property for rf on sync order:

$$x := 1;\ \mathsf{F}^{\mathsf{rel}};\ y := 1 \parallel r := y;\ \mathsf{F}^{\mathsf{acq}};\ s := x$$



($\leq$)

Previous def of candidate requires:

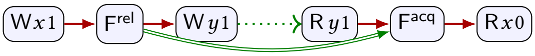
(c7a) if $d \xrightarrow{\mathsf{rf}} e$ and $\lambda(d)$ strongly-matches $\lambda(e)$ then $d \leq e$.

This is not good enough for fences. A possible fix is the following closure condition:

(c7a′) if $d' \le d \xrightarrow{\mathsf{rf}} e \le e'$ and $\lambda(d')$ strongly-matches $\lambda(e')$ then $d' \le e'$.

With that we have the following, using $\Longrightarrow$ for edges induced by closure when $d' \neq d$ or $e' \neq e$:

$$x := 1;\ \mathsf{F}^{\mathsf{rel}};\ y := 1 \parallel r := y;\ \mathsf{F}^{\mathsf{acq}};\ s := x$$



($\le$)

This seems to work for the above examples, but it could be too strong in general.

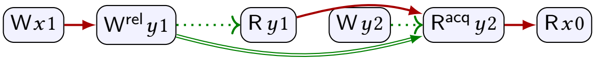
- One possibility is to restrict to preceding and following things in the same thread:
  (c7a″) if $d' \le_{\mathsf{po}} d \xrightarrow{\mathsf{rf}} e \le_{\mathsf{po}} e'$ and $\lambda(d')$ strongly-matches $\lambda(e')$ then $d' \le e'$.
  where $\le_{\mathsf{po}}$ is the obvious restriction of $\le$ to actions on the same thread.
- With either (c7a′) or (c7a″) is it too strong to require $\le$ that be transitive? In particular:
  - if we restrict to $\le_{\mathsf{po}}$, the closure condition (c7a″) could add order between actions on the same thread via cross-thread reads.
  - How does transitivity interact with scopes?

Anton proposes:

(M9b′) if $d \xrightarrow{\mathsf{rmw}} e$ then $d \sqsubseteq e$,
(c7a‴) if $d' \le d\ (\xrightarrow{\mathsf{rf}}; (\xrightarrow{\mathsf{rmw}}; \xrightarrow{\mathsf{rf}})^*)\ e \le e'$ and $\lambda(d')$ strongly-matches $\lambda(e')$ then $d' \le e'$.

The following behavior is allowed by Arm, IMM, and C11, but forbidden by PTX. PTX forbids it since acquire reads work as fences for po-previous reads from the same location (symmetrically to release writes for po-latter writes to the same location in IMM, C11, and PTX).

$$x := 1;\ y^{\mathsf{rel}} := 1 \parallel r := y;\ y := 2;\ s := y^{\mathsf{acq}};\ t := x$$



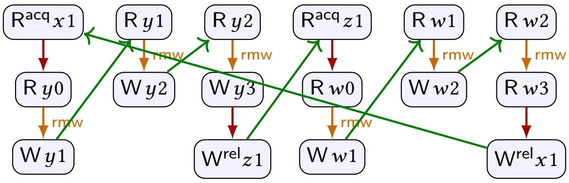
($\le$)

To allow this on for IMM, we need to drop $(\mathsf{R}x, \mathsf{R}^{\sqsupseteq\mathsf{acq}}x)$ from sync-delays.

The following is allowed by C11, but not IMM or PTX. The goal here is to construct a cycle $a \xrightarrow{\mathsf{rf}} b \xrightarrow{\mathsf{hb}} c \xrightarrow{\mathsf{rf}} d \xrightarrow{\mathsf{hb}} a$ where rf will be included in synch-relation. In relational notation, the cycle has the following form:

$$\left(\mathsf{rmw}; (\mathsf{rfe}; \mathsf{rmw})^2; \mathsf{ppo}; [\mathsf{W}^{\mathsf{rel}}]; \mathsf{rfe}; [\mathsf{R}^{\mathsf{acq}}]; \mathsf{ppo}\right)^2$$

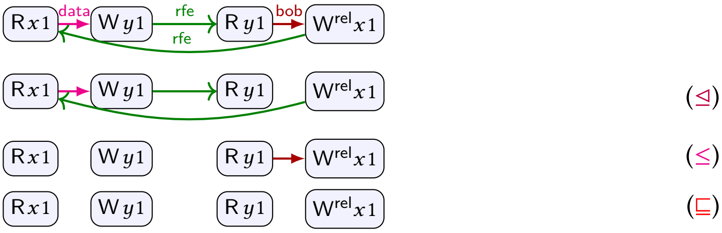
$$r := x^{\mathsf{acq}};\ \mathtt{INC}(y) \parallel \mathtt{INC}(y) \parallel \mathtt{INC}(y);\ z^{\mathsf{rel}} := 1 \parallel s := z^{\mathsf{acq}};\ \mathtt{INC}(w) \parallel \mathtt{INC}(w) \parallel \mathtt{INC}(w);\ x^{\mathsf{rel}} := 1$$



6 RELATING IMM AND PTX

It looks like we cannot prove compilation correctness from IMM to PTX. (In this email I assume that all threads are in the same CTA, so any relation is a morally strong one if it is applicable.) The problem is in the LB-data-rel example:

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



IMM forbids it, but PTX allows it. The point is that IMM mixes dependencies and release/acquire-induced po-order in its NoOOTA axiom, whereas PTX doesn't — release/acquire are only used to have coherence.

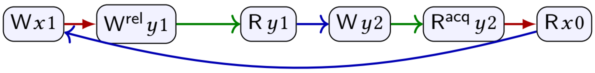
The problem is related to the one we have already discussed in the context of the C++ model – if you don't have acquire reads in the program, then you can erase release annotations from writes. In this regard, PTX is closer to PL memory models than to hardware ones.

AFAIU for the same reason we won't be able to show compilation correctness from the Pomset model to PTX even directly, if the Pomset model mixes release/acquire induced order with dependencies in the same causality relation.

The previous example in the section shows that IMM's acquires are stronger than PTX for this pattern. The next example shows that acquiring reads in PTX are stronger than in IMM for a different pattern. Thus the acquires in PTX and IMM are incomparable.

The following behavior is allowed by IMM and C11, but forbidden by PTX. PTX forbids it since acquire reads work as fences for po-previous reads from the same location (symmetrically to release writes for po-latter writes to the same location in IMM, C11, and PTX).

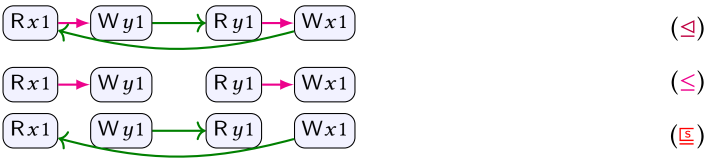
$$x := 1\,;\ y^{\mathsf{rel}} := 1 \parallel r := y\,;\ y := 2\,;\ s := y^{\mathsf{acq}}\,;\ t := x$$



7 THIN AIR

Need $\trianglelefteq$ to prevent thin air on rlx:

$$y := x \parallel x := y$$

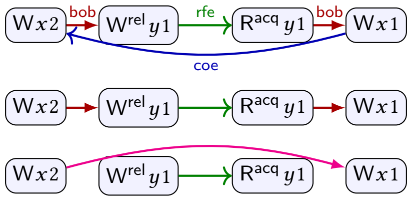


8 IMM EXAMPLES

Disallowed by IMM:

$$x := 2;\ y^{\mathsf{rel}} := 1 \parallel r := y^{\mathsf{acq}};\ x := 1$$

(PUB-REL-ACQ-COE)



(✗IMM)

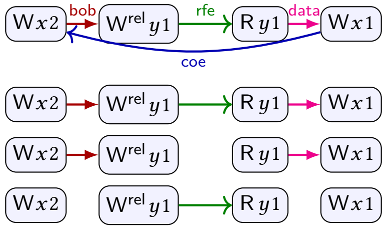
($\trianglelefteq$ = $\leq$)

($\underline{\sqsubset\mathsf{s}}$)

Allowed by IMM, but not by Power/ARMv7/ARMv8/TSO:

$$x := 2;\ y^{\mathsf{rel}} := 1 \parallel r := y;\ x := 1$$

(PUB-REL-RLX-COE)



(✓IMM)

($\trianglelefteq$)

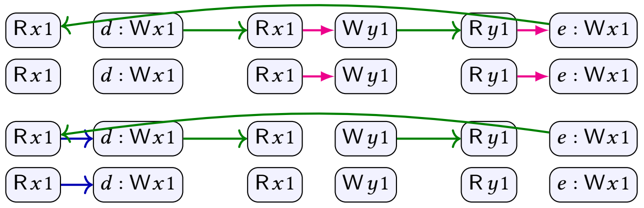
($\leq$)

($\underline{\sqsubset\mathsf{s}}$)

Example from talk:

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

(ARM7-WEAK)



($\trianglelefteq$)

($\leq$)

($\underline{\sqsubset\mathsf{s}}$)

($\underline{\sqsubset\mathsf{w}}$)

9 PTX EXAMPLES

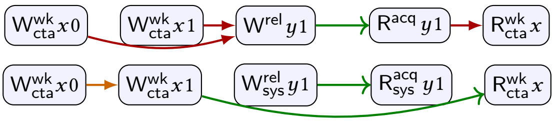
Based on [Lustig et al. 2019; NVIDIA 2020].

In examples, all threads in different ctas.

(Rx0) must be forbidden. Before fulfilling the read:

$$x^{\mathsf{wk}}_{\mathsf{cta}} := 0;\ x^{\mathsf{wk}}_{\mathsf{cta}} := 1;\ y^{\mathsf{rel}} := 1 \parallel r := y^{\mathsf{acq}};\ s := x^{\mathsf{wk}}_{\mathsf{cta}}$$

(PUB1$_{\mathsf{SYS}}$)



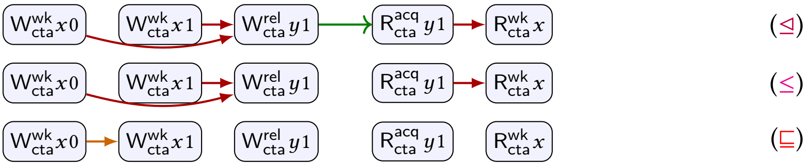
($\trianglelefteq$ = $\leq$)

($\sqsubseteq$)

(Wx1) $\underset{\sim}{\sqsubseteq}$ (Rx) is required by c8b, enforcing publication.

$(\mathsf{R}\,x0)$ must be allowed:

$$x^{\mathsf{wk}}_{\mathsf{cta}} := 0;\ x^{\mathsf{wk}}_{\mathsf{cta}} := 1;\ y^{\mathsf{rel}}_{\mathsf{cta}} := 1 \parallel r := y^{\mathsf{acq}}_{\mathsf{cta}};\ s := x^{\mathsf{wk}}_{\mathsf{cta}} \qquad (\text{PUB1}_{\text{CTA}})$$


$\mathsf{W}^{\mathsf{wk}}_{\mathsf{cta}}x0$
$\mathsf{W}^{\mathsf{wk}}_{\mathsf{cta}}x1$
$\mathsf{W}^{\mathsf{rel}}_{\mathsf{cta}}y1$
$\mathsf{R}^{\mathsf{acq}}_{\mathsf{cta}}y1$
$\mathsf{R}^{\mathsf{wk}}_{\mathsf{cta}}x$
$(\trianglelefteq)$
$(\leq)$
$(\sqsubseteq)$

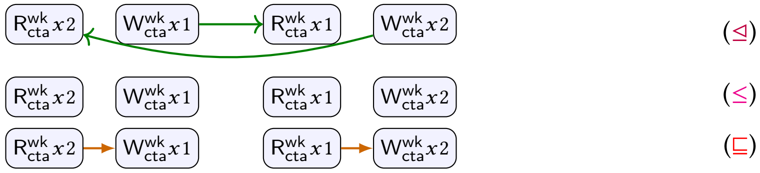
We do not have $(\mathsf{W}^{\mathsf{rel}}y1) \leq (\mathsf{R}^{\mathsf{acq}}y1)$ since c7a only requires order for things that are morally strong.

Another example that may be of interest (nothing morally strong). Can this $(\mathsf{R}\,x0)$?

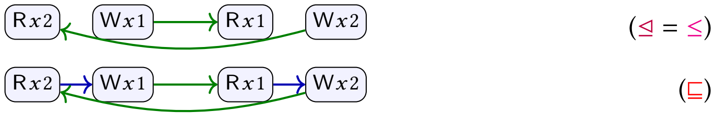
$$x := 0;\ x := 1 \parallel y := x \parallel \mathtt{if}(y)\{r := x\}$$

PTX allows TC16 for events that are not mutually strong (TC16_{WK}), but disallows it when events are mutually strong (TC16_{SYS}). Note that $\leq$ imposes no requirements here. Fulfillment imposes no order. This example shows that c8b cannot be strengthened to replace $\underset{\sim}{\sqsubseteq}$ with $\sqsubseteq$.

$$r := x^{\mathsf{wk}}_{\mathsf{cta}};\ x^{\mathsf{wk}}_{\mathsf{cta}} := 1 \parallel s := x^{\mathsf{wk}}_{\mathsf{cta}};\ x^{\mathsf{wk}}_{\mathsf{cta}} := 2 \qquad (\text{TC16}_{\text{WK}})$$

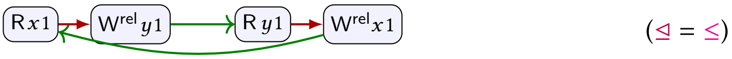

$\mathsf{R}^{\mathsf{wk}}_{\mathsf{cta}}x2$
$\mathsf{W}^{\mathsf{wk}}_{\mathsf{cta}}x1$
$\mathsf{R}^{\mathsf{wk}}_{\mathsf{cta}}x1$
$\mathsf{W}^{\mathsf{wk}}_{\mathsf{cta}}x2$
$(\trianglelefteq)$
$(\leq)$
$(\sqsubseteq)$

$$r := x;\ x := 1 \parallel s := x;\ x := 2 \qquad (\text{TC16}_{\text{SYS}})$$



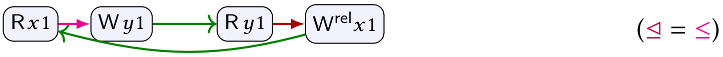
About Release-Acquire semantics. Anton confirms that the following example is allowed in C11, but disallowed in the IMM. It is apparently allowed in C11 with the intention to allow releasing writes to be downgraded to relaxed in the case that only fulfill relaxed reads.

$$r := x;\ y^{\mathsf{rel}} := 1 \parallel s := y;\ x^{\mathsf{rel}} := 1 \qquad (\text{LB-REL})$$


$\mathsf{R}\,x1$
$\mathsf{W}^{\mathsf{rel}}y1$
$\mathsf{R}\,y1$
$\mathsf{W}^{\mathsf{rel}}x1$
$(\trianglelefteq = \leq)$

Another example from Anton. This is allowed in PTX because it does not include synchronization in the no-tar axiom, only in coherence and causality.

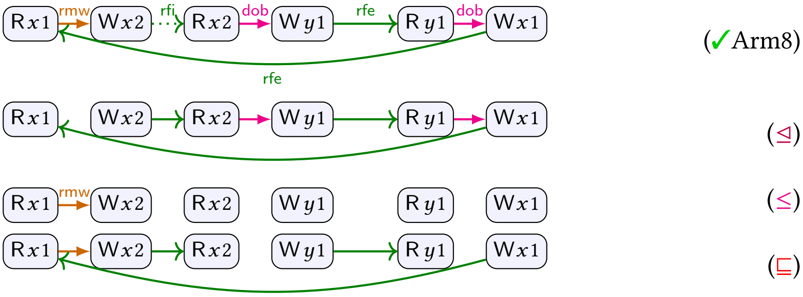
$$r := x;\ y := r \parallel s := y;\ x^{\mathsf{rel}} := 1 \qquad (\text{LB-DATA-REL})$$



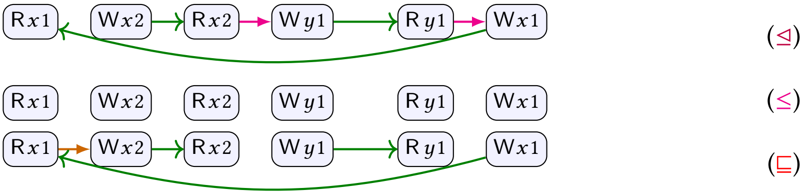
10 RFI EXAMPLES

Bad example:

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



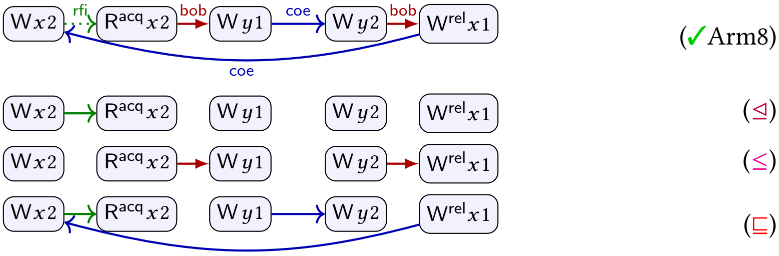
$$r := x\,;\; x := 2\,;\; s := x\,;\; y := s{-}1 \parallel r := y\,;\; x := r$$



Anton example 1 (Allowed by ARM) `[rfi-coe-coe]`

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

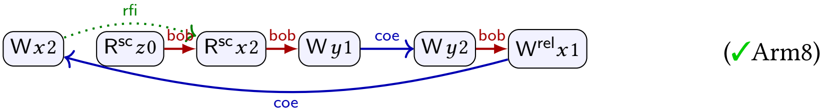
(RFI-COE-COE)


W x2
rfi
$\mathsf{R}^{\mathsf{acq}} x2$
bob
W y1
coe
W y2
bob
$\mathsf{W}^{\mathsf{rel}} x1$
coe
(✓Arm8)
(⊴)
(≤)
(⊑)

Internal reads survive acquires `[rfi-acq-coe-coe]` (where SC read = `LDAR`)

$$x := 2\,;\; s := z^{\mathsf{sc}}\,;\; r := x^{\mathsf{sc}}\,;\; y := 1 \parallel y := 2\,;\; x^{\mathsf{rel}} := 1$$

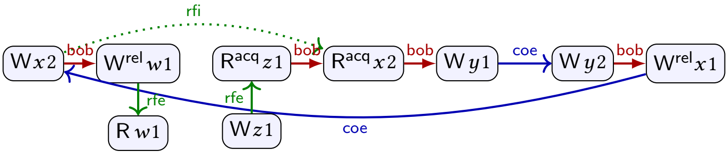
(RFI-ACQ-COE-COE)


rfi
W x2
$\mathsf{R}^{\mathsf{sc}} z0$
bob
$\mathsf{R}^{\mathsf{sc}} x2$
bob
W y1
coe
W y2
bob
$\mathsf{W}^{\mathsf{rel}} x1$
coe
(✓Arm8)

And release-acquire pairs [rfi-ra-coe-coe] (where acquiring read = LDAPR)

$$\begin{array}{l} x := 2;\; w^{\mathsf{rel}} := 1;\; s := z^{\mathsf{acq}};\; r := x^{\mathsf{acq}};\; y := 1 \\ \quad \parallel\; y := 2;\; x^{\mathsf{rel}} := 1 \;\parallel\; r := w;\; z := 1; \end{array}$$

(RFI-RA-COE-COE2)

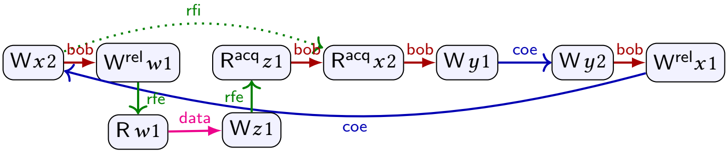


(✓Arm8)

But not if either acquire is strengthened to SC (where SC read = LDAR). The execution is also disallowed if an external thread places order between the ra accesses:

$$\begin{array}{l} x := 2;\; w^{\mathsf{rel}} := 1;\; s := z^{\mathsf{acq}};\; r := x^{\mathsf{acq}};\; y := 1 \\ \quad \parallel\; y := 2;\; x^{\mathsf{rel}} := 1 \;\parallel\; r := w;\; z := r; \end{array}$$

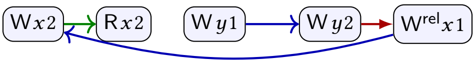
(RFI-RA-DATA-COE-COE)



(✗Arm8)

To allow this, weaken ra to rlx when read fulfilled by relaxed write of same thread (don't need to allow this when the write is part of an RMW).

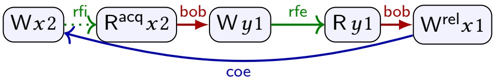
$$x := 2;\; r := x^{\mathsf{acq}};\; y := 1 \;\parallel\; y := 2;\; x^{\mathsf{rel}} := 1$$



RF variant [rfi-rfe-coe]:

$$x := 2;\; r := x^{\mathsf{acq}};\; y := 1 \;\parallel\; s := y;\; x^{\mathsf{rel}} := 1$$

(RFI-RFE-COE)

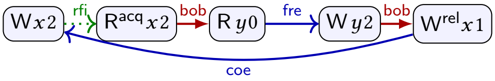


(✓Arm8)

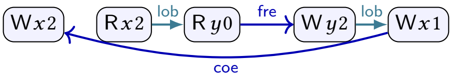
TSO variant [rfi-fre-coe2]:

$$x := 2;\; r := x^{\mathsf{acq}};\; s := y \;\parallel\; y := 2;\; x^{\mathsf{rel}} := 1$$

(RFI-COE-COE2)



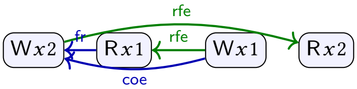
(✓Arm8)



(✓TSO)

Note that TSO does not order W to R in local order, even in poloc. Nonetheless, TSO disallows the following because of local visibility in first thread.

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



(✗TSO)

[Higham and Kawash 2000] describe TSO as a linearization of partial order including:

- poloc
- lws = po; [W]
- $d \overset{po}{\dashrightarrow} e$ when $c \xrightarrow{rfe} d \overset{po}{\dashrightarrow} e$

[Alglave et al. 2020] describe TSO as linearization of partial order satisfying internal visibility and including

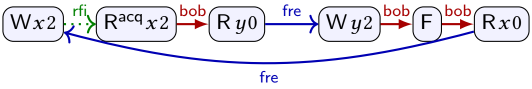
- [W]; po; [W]
- $d \overset{po}{\dashrightarrow} e$ when $c \xrightarrow{rfe} d \overset{po}{\dashrightarrow} e$, from `(range(rfe) * _)`
- [R]; po; [W], from `(rfi^-1; lob)`

Ignoring fences and RMWs:

```
let rec lob = po \ ([W]; po; [R])
let IM0 = loc & ((IW * (M\IW)) | ((W\FW) * FW))
let gc-req = (W * _) | ((R * _) & ((range(rfe) * _) | (rfi^-1; lob))
let preorder-gcb = IM0 | lob & gc-req
```

Double FRE variant `[rfi-fre-fre]`:

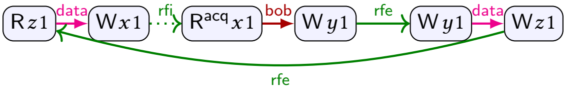
$$x := 2;\ r := x^{\mathsf{acq}};\ s := y \parallel y := 2;\ \mathtt{F};\ r := x \qquad \text{(RFI-FRE-FRE)}$$



(✓Arm8)

It does not seem possible to do this only with rfe. ARM disallows this `[data-rfi-rfe-rfe]`:

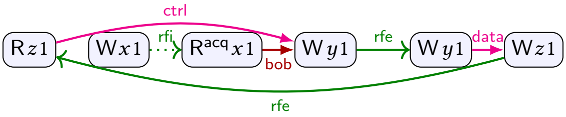
$$x := z;\ r := x^{\mathsf{acq}};\ y := 1 \parallel z := y \qquad \text{(DATA-RFI-RFE-RFE)}$$



(✗Arm8)

It also disallows `[ctrl-rfi-rfe-rfe]`:

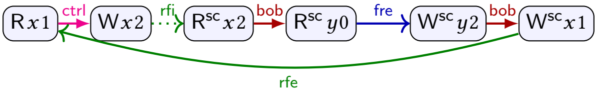
$$\mathtt{if}(z)\{\};\ x := 1;\ r := x^{\mathsf{acq}};\ y := 1 \parallel z := y \qquad \text{(CTRL-RFI-RFE-RFE)}$$



(✗Arm8)

ARM allows some counterintuitive results for SC access `[ctrl-rfi-fre-rfe]`:

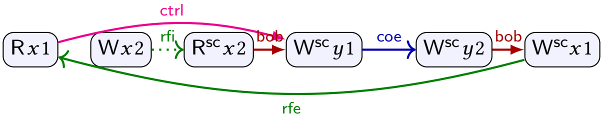
$$\mathtt{if}(x)\{\};\ x := 2;\ r := x^{\mathsf{sc}};\ s := y^{\mathsf{sc}} \parallel y^{\mathsf{sc}} := 2;\ x^{\mathsf{sc}} := 1 \qquad \text{(CTRL-RFI-FRE-RFE)}$$



(✓Arm8)

Not possible with coe `[ctrl-rfi-coe-rfe]`:

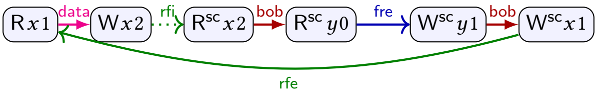
$$\mathtt{if}(x)\{\};\ x := 2;\ r := x^{\mathsf{sc}};\ y^{\mathsf{sc}} := 1 \parallel y^{\mathsf{sc}} := 2;\ x^{\mathsf{sc}} := 1 \qquad \text{(CTRL-RFI-COE-RFE)}$$



(✗Arm8)

This is not allowed with a data dependency instead of a control dependency `[data-rfi-fre-rfe]`:

$$x := x{+}1;\ r := x^{\mathsf{sc}};\ s := y^{\mathsf{sc}} \parallel y^{\mathsf{sc}} := 1;\ x^{\mathsf{sc}} := 1 \qquad \text{(DATA-RFI-FRE-RFE)}$$

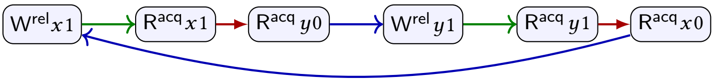


(✗Arm8)

11 SC EXAMPLES

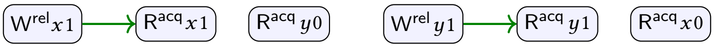
Example 11.1. Consider IRIW with all ra access:

$$x^{\mathsf{rel}} := 1 \parallel r := x^{\mathsf{acq}};\ s := y^{\mathsf{acq}} \parallel y^{\mathsf{rel}} := 1 \parallel r := y^{\mathsf{acq}};\ s := x^{\mathsf{acq}} \qquad \text{(IRIW-ACQ-ACQ)}$$

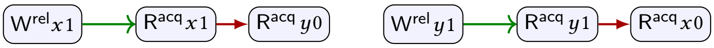


(✓POWER,C11)

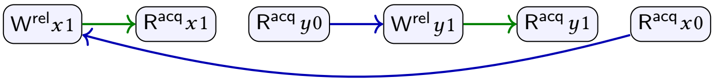
We allow this execution:



($\trianglelefteq$)



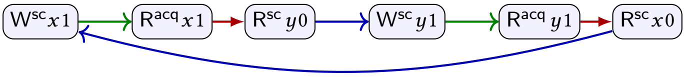
($\leq$)



($\sqsubseteq$)

IRIW-ACQ-SC1, is allowed by trailing-sync compilation to power [Lahav et al. 2017, §1].

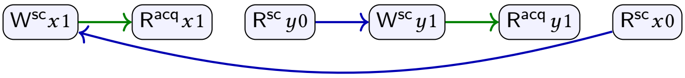
$$x^{\mathsf{sc}} := 1 \parallel r := x^{\mathsf{acq}};\ s := y^{\mathsf{sc}} \parallel y^{\mathsf{sc}} := 1 \parallel r := y^{\mathsf{acq}};\ s := x^{\mathsf{sc}} \qquad \text{(IRIW-ACQ-SC1)}$$



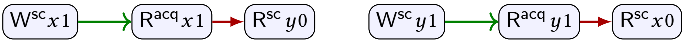
(✓POWER,✗C11)

To model this it is convenient that synchronization is not included in dependency order:

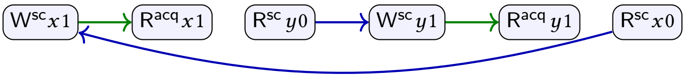
- add sc bullet to def of $\underset{\sim}{\sqsubseteq}$ in C8b,
- add SC access to *sync-delays*.



($\trianglelefteq$)



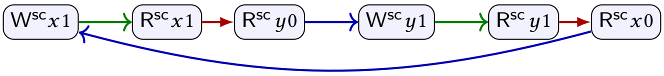
($\leq$)



($\sqsubseteq$)

This correctly forbids the all sc version:

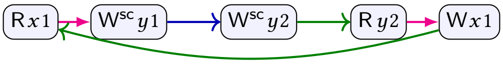
$$x^{\mathsf{sc}} := 1 \parallel r := x^{\mathsf{sc}};\ s := y^{\mathsf{sc}} \parallel y^{\mathsf{sc}} := 1 \parallel r := y^{\mathsf{sc}};\ s := x^{\mathsf{sc}} \qquad \text{(IRIW-SC-SC)}$$



($\trianglelefteq$)

Example 11.2. Thin air with an SC antidependency:

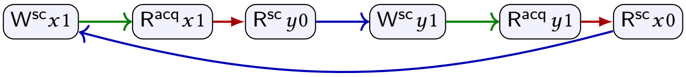
$$y^{\mathsf{sc}} := x \parallel y^{\mathsf{sc}} := 2 \parallel x := y{-}1$$



($\trianglelefteq$)

IRIW-ACQ-SC2 is allowed by trailing-sync compilation to power [Lahav et al. 2017, §1].

$$x^{\mathsf{sc}} := 1 \parallel r := x^{\mathsf{acq}}\,;\, s := y^{\mathsf{sc}} \parallel y^{\mathsf{sc}} := 1 \parallel r := y^{\mathsf{acq}}\,;\, s := x^{\mathsf{sc}} \tag{IRIW-ACQ-SC2}$$

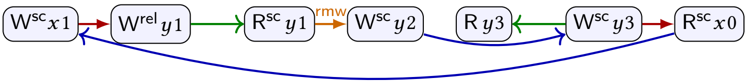

$\mathsf{W}^{\mathsf{sc}}x1$
$\mathsf{R}^{\mathsf{acq}}x1$
$\mathsf{R}^{\mathsf{sc}}y0$
$\mathsf{W}^{\mathsf{sc}}y1$
$\mathsf{R}^{\mathsf{acq}}y1$
$\mathsf{R}^{\mathsf{sc}}x0$

(✓POWER,RC11)

This example is hard to get right for power because it must be allowed with ra reads, but disallowed with sc reads. This seems unsolvable: To allow the version with ra, we would need to weaken the order between the reads in each thread for the ra case, and that would break publication.

Leading sync is also unsound in C11 with RMW [Lahav et al. 2017, §2.1].

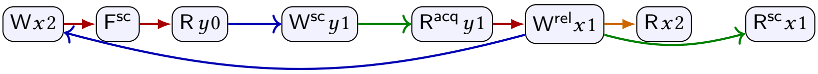
$$x^{\mathsf{sc}} := 1\,;\, y^{\mathsf{rel}} := 1 \parallel \mathsf{FADD}^{\mathsf{sc,sc}}(y, 1)\,;\, s := y \parallel y^{\mathsf{sc}} := 3\,;\, s := x^{\mathsf{sc}} \tag{Z6.U}$$


$\mathsf{W}^{\mathsf{sc}}x1$
$\mathsf{W}^{\mathsf{rel}}y1$
$\mathsf{R}^{\mathsf{sc}}y1$
rmw
$\mathsf{W}^{\mathsf{sc}}y2$
$\mathsf{R}\,y3$
$\mathsf{W}^{\mathsf{sc}}y3$
$\mathsf{R}^{\mathsf{sc}}x0$

(✓POWER,RC11)

Leading sync is also unsound in C11 with SC fences [Lahav et al. 2017, §A.1].

$$x := 2\,;\, \mathsf{F}^{\mathsf{sc}}\,;\, r := y \parallel y^{\mathsf{sc}} := 1 \parallel r := y^{\mathsf{acq}}\,;\, x^{\mathsf{rel}} := 1\,;\, s := x \parallel r := x^{\mathsf{sc}} \tag{RSYNC+RSC}$$

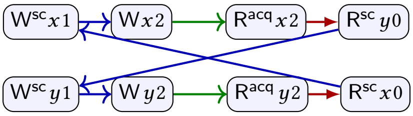

$\mathsf{W}x2$
F^{sc}
$\mathsf{R}\,y0$
$\mathsf{W}^{\mathsf{sc}}y1$
$\mathsf{R}^{\mathsf{acq}}y1$
$\mathsf{W}^{\mathsf{rel}}x1$
$\mathsf{R}x2$
$\mathsf{R}^{\mathsf{sc}}x1$

(✓RC11)

Fulfillment of $(\mathsf{R}x2)$ requires that either $(\mathsf{W}^{\mathsf{rel}}x1) \longrightarrow (\mathsf{W}x2)$ or $(\mathsf{R}x2) \longrightarrow (\mathsf{W}^{\mathsf{rel}}x1)$. It's interesting that in the pomset, $(\mathsf{R}^{\mathsf{sc}}x1)$ is not needed to get a cycle.

There is a long discussion of this in [Bender and Palsberg 2019, §5.2, Fig. 17], where they also discuss this example:

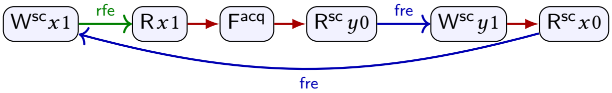
$$x^{\mathsf{sc}} := 1\,;\, x := 2 \parallel y^{\mathsf{sc}} := 1\,;\, y := 2 \parallel r := x^{\mathsf{acq}}\,;\, s := y^{\mathsf{sc}} \parallel r := y^{\mathsf{acq}}\,;\, s := x^{\mathsf{sc}} \tag{IRIW-SC-RLX-ACQ}$$


$\mathsf{W}^{\mathsf{sc}}x1$
$\mathsf{W}x2$
$\mathsf{R}^{\mathsf{acq}}x2$
$\mathsf{R}^{\mathsf{sc}}y0$
$\mathsf{W}^{\mathsf{sc}}y1$
$\mathsf{W}y2$
$\mathsf{R}^{\mathsf{acq}}y2$
$\mathsf{R}^{\mathsf{sc}}x0$

(✓RC11)

[Lahav et al. 2017, §A.2] claims that Arm8 allows this [RWC+acq+sc], but herd7 rejects it. Reason: they are citing the flowing/pop model [Flur et al. 2016] rather than [Pulte et al. 2018].

$$x^{\mathsf{sc}} := 1 \parallel r := x\,;\, \mathsf{F}^{\mathsf{acq}}\,;\, s := y^{\mathsf{sc}} \parallel y^{\mathsf{sc}} := 1\,;\, r := x^{\mathsf{sc}} \tag{RWC+ACQ+SC}$$


$\mathsf{W}^{\mathsf{sc}}x1$
rfe
$\mathsf{R}x1$
$\mathsf{F}^{\mathsf{acq}}$
$\mathsf{R}^{\mathsf{sc}}y0$
fre
$\mathsf{W}^{\mathsf{sc}}y1$
$\mathsf{R}^{\mathsf{sc}}x0$
fre

(✗Arm8)

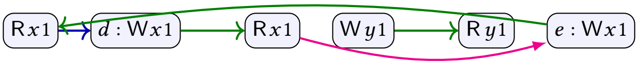
12 TWO ORDER IDEA

The two order idea from OOPSLA talk is:

- Require: $d \sqsubseteq e$ when $d \trianglelefteq e$ and they conflict

This does not work for the IMM or ARMv7, but it may work for Power, TSO, ARMv8. That would be nice. Let's write $\sqsubsetneq$ for this notion, with strong fulfillment.

With this there is a cycle in ARM7-WEAK (weak/strong fulfillment not relevant here):



$(\sqsubsetneq)$

Anton says: ARM7-WEAK is forbidden by Power, TSO, ARMv8, but allowed by ARMv7. Maybe it isn't that important to support it anymore.

There is also a cycle in PUB-REL-RLX-COE. Anton says: I checked Power/ARMv7 models in this regard. They disallow the behavior (as well as ARMv8 and TSO), so we can in principle strengthen IMM to forbid it as well. For that, we may add axiom to IMM forbidding cycles in $\mathsf{co} \cup([\mathsf{W}];\mathsf{rfe}^{?};([\mathsf{R}^{\mathsf{acq}}]\cup \mathsf{po};[\mathsf{FW}^{\mathsf{rel}}]);\mathsf{ar}^{*};[\mathsf{W}])$. This works if we have acquire/release accesses on the path since they are compiled with fences to Power.

13 OLD NOTES

Goal is to capture POWER, not ARM-V7. See §13.3, below. So cannot use IMM out of the box.

Introduce *weak order* $\sqsubseteq$[3].

Definition 13.1 (2.1). A *(memory order) pomset* is a tuple $(E, \leq, \sqsubseteq, \lambda, \xrightarrow{\mathsf{rmw}})$:

- E is a set of *states*,
- $\leq \subseteq (E \times E)$ and $\sqsubseteq \subseteq (E \times E)$ are partial orders,
- $\lambda : E \longrightarrow (\Phi \times \mathcal{A})$ is a *labeling*, from which we derive functions $\kappa : E \longrightarrow \Phi$ and $\lambda : E \longrightarrow \mathcal{A}$,
- if $d \ (\leq \cup \sqsubseteq)\ e$ then $\kappa(e)$ implies $\kappa(d)$, and
- $\bigwedge_e \kappa(e)$ is satisfiable.

Additional stuff:

- if $d \ (\leq ; \sqsubset)\ e$ then $d \neq e$, and
- if $d \ (\leq ; \sqsubseteq ; \leq)\ e$, d is SC, and e is SC, then $d \leq e$.

RMW:

- If $d \xrightarrow{\mathsf{rmw}} e$, then $d \leq e$.
- If $\exists x.\ c$ and e write x, $c \sqsubseteq e$, and $d \xrightarrow{\mathsf{rmw}} e$, then $c \sqsubseteq d$.
- If $\exists x.\ c$ and e write x, $d \sqsubseteq c$, and $d \xrightarrow{\mathsf{rmw}} e$, then $e \sqsubseteq c$.

Update the definitions to use $\sqsubseteq$ instead of $\leq$ in two places:

- the last item defining fulfillment, and
- item 5b defining prefixing.

Definition 13.2 (2.4). We say d *fulfills* e *on* x if

- d writes v to x,
- e reads v from x,
- $d \leq e$, and
- if c writes to x then either $c \sqsubseteq d$ or $e \sqsubseteq c$.

Definition 13.3 (2.10). Let $(\phi \mid a) \Rightarrow \mathcal{P}$ be the set $\nabla\mathcal{P}'$ where $P' \in \mathcal{P}'$ when there is some $P \in \mathcal{P}$ that satisfies items 1-4 of Definition 2.8 such that:

5a. if e writes then either $d <' e$ or $\kappa'(e)$ implies $\kappa(e)$,
5b. if d and e are actions in conflict, then $d \sqsubset' e$,
5c. if d is an acquire or e is a release, then $d <' e$,
5d. if d is an SC write and e is an SC read, then $d <' e$,
5e. if d reads, and e is an acquiring fence, then $d <' e$, and
5f. if d is a releasing fence, and e writes, then $d <' e$.

[3] Note we can *not* require

- if $d \ (\leq \cup \sqsubseteq)\ e$ then $\sigma(d)$ subsumes $\sigma(e)$.

This does not hold, for example, in $[\![x := 1\,;\, x := 2]\!]$.

Weak order is only required to relate actions on the same location. In augmentation minimal pomsets, it is a subset of eco (only relates writes that are read). The irreflexivity requirement in the definition is thus comparable to requiring that $\leq \cup \sqsubseteq_x$ is a partial order, for every x. It is *not* the case that $\leq \cup \sqsubseteq$ is a partial order.

Note that we have a kind of semi-transitivity here, but only per variable.

- If $c \leq_x d \sqsubseteq_x e$ then $c \sqsubseteq_x e$.
- If $c \sqsubseteq_x d \leq_x e$ then $c \sqsubseteq_x e$.

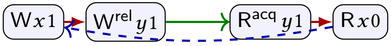
With the requirements of fulfillment, we have that $d \leq e$ implies $d \sqsubseteq e$ when the actions conflict—there is a caveat for unread writes, where no order is forced.

Here are some examples of the main text. To better visualize, we use different arrowheads for strong and weak order. We use a single color for strong order, and separate colors for each variable in weak order.

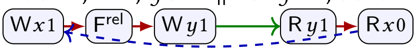
13.1 Many Examples

Here is fencing behavior mixing with release/acquire:

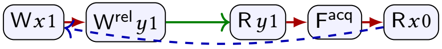
$$x := 1;\ y^{\mathsf{ra}} := 1 \parallel r := y^{\mathsf{acq}};\ s := x$$



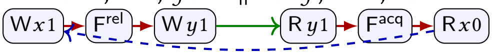
$$x := 1;\ \mathsf{F}^{\mathsf{rel}};\ y := 1 \parallel r := y^{\mathsf{acq}};\ s := x$$



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

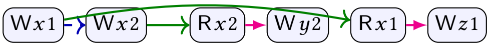


$$x := 1;\ \mathsf{F}^{\mathsf{rel}};\ y := 1 \parallel r := y;\ \mathsf{F}^{\mathsf{acq}};\ s := x$$



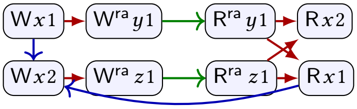
Coherence C does not allow:

$$x := 1;\ x := 2 \parallel y := x;\ z := x$$



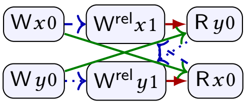
Coherence Java allows:

$$x := 1;\ y^{\mathsf{ra}} := 1 \parallel x := 2;\ z^{\mathsf{ra}} := 1 \parallel r := y^{\mathsf{sc}};\ r := z^{\mathsf{sc}};\ r := x;\ r := x$$



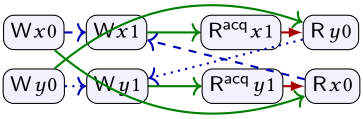
Store buffering

$$x := 0;\ y := 0;\ (x^{\mathsf{ra}} := 1;\ y := r \parallel y^{\mathsf{ra}} := 1;\ x := r)$$



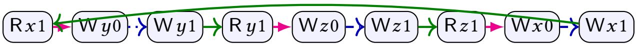
IRIW

$$x := 0;\ x := 1 \parallel r := x^{\mathsf{acq}};\ s := y$$
$$\parallel y := 0;\ y := 1 \parallel r := y^{\mathsf{acq}};\ s := x$$



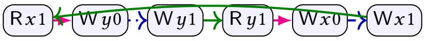
Three variable MCA example Allowed.

$$\mathtt{if}(x)\{y := 0\};\ y := 1 \| \mathtt{if}(y)\{z := 0\};\ z := 1 \| \mathtt{if}(z)\{x := 0\};\ x := 1$$



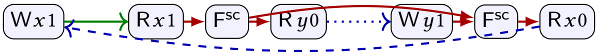
Two variable MCA example Allowed.

$$\mathtt{if}(x)\{y := 0\};\ y := 1 \| \mathtt{if}(y)\{x := 0\};\ x := 1$$



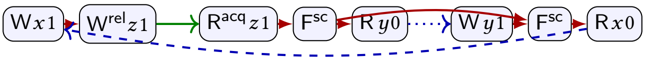
[Lahav et al. 2017, Fig. 5]

$$x := 1 \parallel r := x;\ \mathsf{F}^{\mathsf{sc}};\ r := y \parallel y := 1;\ \mathsf{F}^{\mathsf{sc}};\ r := x$$



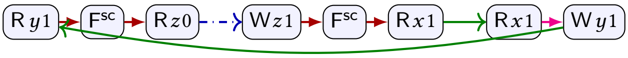
[Lahav et al. 2017, Fig. 6]

$$x := 1;\ z^{\mathsf{ra}} := 1;\ \parallel r := z^{\mathsf{acq}};\ \mathsf{F}^{\mathsf{sc}};\ r := y \parallel y := 1;\ \mathsf{F}^{\mathsf{sc}};\ r := x$$



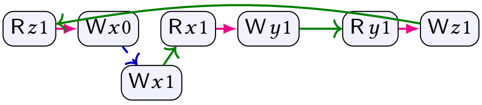
PSC is part of AR [Podkopaev et al. 2019, Ex. 3.9]:

$$r := y;\ \mathsf{F}^{\mathsf{sc}};\ r := z \parallel z := 1;\ \mathsf{F}^{\mathsf{sc}};\ r := x \parallel \mathtt{if}(x{\neq}0)\{y := 1\}$$



Detour Example [Podkopaev et al. 2019, Ex. 3.7]:

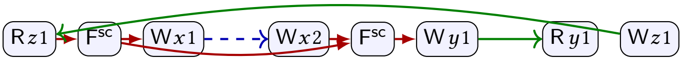
$$x := z{-}1;\ y := x \parallel x := 1 \parallel z := y$$



13.2 Another fencing example

[Podkopaev et al. 2019, §D]: The following execution graph is not consistent in the promise-free declarative model of [Kang et al. 2017]. Nevertheless, its mapping to POWER (obtained by simply replacing Fsc with Fsync) is POWER-consistent and po ∪ rf is acyclic (so it is Strong-POWER-consistent). Note that, using promises, the promising semantics allows this behavior.

$$r := z;\ \mathsf{F}^{\mathsf{sc}};\ x := 1 \parallel x := 2;\ \mathsf{F}^{\mathsf{sc}};\ y := 1 \parallel r := y;\ z := 1$$



Allowed behavior on POWER... Is there a dependency in the last thread? If so, this is a problem.

[Podkopaev et al. 2019, §8]: To establish the correctness of compilation of the promising semantics to POWER, Kang et al. [2017] followed the approach of Lahav and Vafeiadis [2016]. This approach reduces compilation correctness to POWER to (i) the correctness of compilation to the POWER model strengthened with $\mathsf{po} \cup \mathsf{rf}$ acyclicity; and (ii) the soundness of local reorderings of memory accesses. To establish (i), Kang et al. [2017] wrongly argued that the strengthened POWER-consistency of mapped promise-free execution graphs imply the promise-free consistency of the source execution graphs. This is not the case due to SC fences, which have relatively strong semantics in the promise-free declarative model (see [Podkopaev et al. 2018, Appendix D] for a counter example). Nevertheless, our proof shows that the compilation claim of Kang et al. [2017] is correct.

13.3 Power versus ARM7

[Lahav and Vafeiadis 2016, §5]: Characterizing ppo of power:

$$[\mathsf{RU}];(\mathsf{deps} \cup \mathsf{poloc})^{+};[\mathsf{WU}] \subseteq \mathsf{ppo} \tag{PPO lower}$$

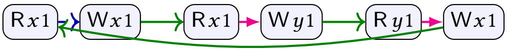
$$\mathsf{ppo} \cap \mathsf{po}_{\mathsf{imm}} \subseteq (\mathsf{deps} \cup \mathsf{poloc})^{+} \tag{PPO upper}$$

R_{imm} denotes the relation consisting of all immediate R-edges, i.e., pairs $(a, b) \in R$ such that for every c, $(c, b) \in R$ implies $(c, a) \in R^{?}$, and $(a, c) \in R$ implies $(b, c) \in R^{?}$.

[Podkopaev et al. 2019, After example 3.6]: Note that we do not include fri in ppo since it is not preserved in ARMv7 [Alglave et al. 2014] (unlike in x86-TSO, POWER, and ARMv8). Thus, as ARMv7 (as well as the Flowing and POP models of ARM in [Flur et al. 2016]), IMM allows the weak behavior from [Lahav and Vafeiadis 2016, §6].

[Lahav and Vafeiadis 2016, §6]: Consider the program in Fig. 4.

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



Note that no reorderings or eliminations can be applied to this program. In the second and the third threads, reordering is forbidden because of the dependency between the load and the subsequent store. On the first thread, there is no dependency, but since the load and the store access the same location, their reordering is generally unsound, as it allows the load to read from the (originally subsequent) store. Moreover, this program cannot return a = 1 under a (po ∪ rf)-acyclic model, because the only instance of the constant 1 in the program occurs after the load of x in the first thread. Nevertheless, this behavior is allowed under both the axiomatic ARMv7 model of Alglave et al. [4] and the ARMv8 Flowing and POP models of Flur et al. [12]

The axiomatic ARMv7 model [4] is the same as the Power model presented in Section 5, with the only difference being the definition of ppo (preserved program order). In particular, this model does not satisfy (ppo-lower-bound) because

$$[\mathsf{RU}];\mathsf{poloc};[\mathsf{WU}] \not\subseteq \mathsf{ppo}.$$

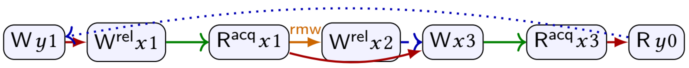
Hence, the first thread's program order in the example above is not included in ppo, and there is no happens-before cycle. For the same reason, our proof for Power does not carry over to ARM.

In the ARMv8 Flowing model [12], consider the topology where the first two threads share a queue and the third thread is separate. The following execution is possible: (1) the first thread issues a load request from x and immediately commits the x := 1 store; (2) the second thread then issues a load request from x, which gets satisfied by the x := 1 store, and then (3) issues a store to y := 1; (4) the store to y gets reordered with the x-accesses, and flows to the third thread; (5) the third thread then loads y = 1, and also issues a store x := 1, which flows to the memory; (6) the load of x flows to the next level and gets satisfied by the x := 1 store of the third thread; and (7) finally the x := 1 store of the first thread also flows to the next level. The POP model is strictly weaker than the Flowing model, and thus also allows this outcome.

13.4 Fences and RMW

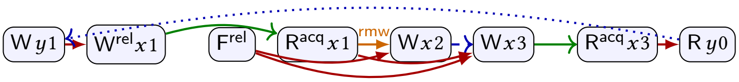
[Podkopaev et al. 2019, Remark 2, After example 3.1]: Aim: allow the splitting of release writes and RMWs into release fences followed by relaxed operations. In RC11 [Lahav et al. 2017], as well as in C/C++11 [Batty et al. 2011], this rather intuitive transformation, as we found out, is actually unsound.

$$y := 1;\ x^{\mathsf{ra}} := 1 \parallel \mathtt{INC}^{\mathsf{ra,ra}}(x)\,;\ x := 3 \parallel r := x^{\mathsf{acq}};\ s := y$$



(R)C11 disallows the annotated behavior, due in particular to the release sequence formed from the release exclusive write to x in the second thread to its subsequent relaxed write. However, if we split the increment to fencerel; a := FADDacq,rlx(x, 1) (which intuitively may seem stronger), the release sequence will no longer exist, and the annotated behavior will be allowed. IMM overcomes this problem by strengthening sw in a way that ensures a synchronization edge for the transformed program as well

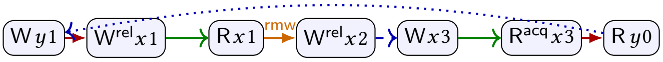
$$y := 1;\ x^{\mathsf{ra}} := 1 \parallel \mathsf{F}^{\mathsf{rel}};\ \mathtt{INC}^{\mathsf{ra,rlx}}(x)\,;\ x := 3 \parallel r := x^{\mathsf{acq}};\ s := y$$



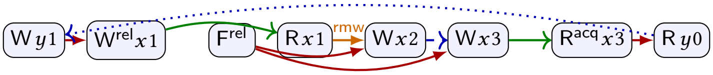
We seem to disallow both of these out of the box.

In the case of a relaxed read in the RMW, the outcome is allowed in both cases:

$$y := 1;\ x^{\mathsf{ra}} := 1 \parallel \mathtt{INC}^{\mathsf{rlx,ra}}(x)\,;\ x := 3 \parallel r := x^{\mathsf{acq}};\ s := y$$



$$y := 1;\ x^{\mathsf{ra}} := 1 \parallel \mathsf{F}^{\mathsf{rel}};\ \mathtt{INC}^{\mathsf{rlx,rlx}}(x)\,;\ x := 3 \parallel r := x^{\mathsf{acq}};\ s := y$$



REFERENCES

Jade Alglave, Will Deacon, Richard Grisenthwaite, Antoine Hacquard, and Luc Maranget. 2020. Armed cats: Formal Concurrency Modelling at Arm. Draft. , 49 pages.

John Bender and Jens Palsberg. 2019. A formalization of Java's concurrent access modes. *Proc. ACM Program. Lang.* 3, OOPSLA (2019), 142:1–142:28. https://doi.org/10.1145/3360568

Soham Chakraborty and Viktor Vafeiadis. 2017. Formalizing the concurrency semantics of an LLVM fragment. In *Proceedings of the 2017 International Symposium on Code Generation and Optimization, CGO 2017, Austin, TX, USA, February 4-8, 2017*, Vijay Janapa Reddi, Aaron Smith, and Lingjia Tang (Eds.). ACM, 100–110. http://dl.acm.org/citation.cfm?id=3049844

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

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

Shaked Flur, Kathryn E. Gray, Christopher Pulte, Susmit Sarkar, Ali Sezgin, Luc Maranget, Will Deacon, and Peter Sewell. 2016. Modelling the ARMv8 architecture, operationally: concurrency and ISA. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, Rastislav Bodík and Rupak Majumdar (Eds.). ACM, 608–621. https://doi.org/10.1145/2837614.2837615

Lisa Higham and Jalal Kawash. 2000. Memory Consistency and Process Coordination for SPARC Multiprocessors. In *High Performance Computing - HiPC 2000, 7th International Conference, Bangalore, India, December 17-20, 2000, Proceedings (Lecture Notes in Computer Science, Vol. 1970)*, Mateo Valero, Viktor K. Prasanna, and Sriram Vajapeyam (Eds.). Springer, 355–366. https://doi.org/10.1007/3-540-44467-X_32

Jeehoon Kang. 2019. *Reconciling Low-Level Features of C with Compiler Optimizations.* Ph.D. Dissertation. Seoul National University, Seoul, South Korea. https://sf.snu.ac.kr/jeehoon.kang/thesis/

Ori Lahav and Viktor Vafeiadis. 2016. Explaining Relaxed Memory Models with Program Transformations. In *FM 2016: Formal Methods - 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9995)*, John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou (Eds.). Springer, 479–495. https://doi.org/10.1007/978-3-319-48989-6_29

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

Daniel Lustig, Sameer Sahasrabuddhe, and Olivier Giroux. 2019. A Formal Analysis of the NVIDIA PTX Memory Consistency Model. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck (Eds.). ACM, 257–270. https://doi.org/10.1145/3297858.3304043

NVIDIA. 2020. Parallel Thread Execution ISA Version 7.1. https://docs.nvidia.com/cuda/parallel-thread-execution/index.html#memory-consistency-model.

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

Christopher Pulte, Shaked Flur, Will Deacon, Jon French, Susmit Sarkar, and Peter Sewell. 2018. Simplifying ARM concurrency: multicopy-atomic axiomatic and operational models for ARMv8. *PACMPL* 2, POPL (2018), 19:1–19:29. https://doi.org/10.1145/3158107