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.

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

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

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 (grp), processor (proc) 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], $\rrbracket$ denotes parallel composition. If ($S_1 \rrbracket_{\gamma} S_2$) is executed with thread ID α, then S_2 runs with ID γ and S_1 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.

The semantics is built from the following.

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

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

- 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 equalities (M=N) and (x=M),
- formulae are closed under negation, conjunction, disjunction, 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 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 $\mathsf{tt} \vDash \phi$. We say ϕ is *unsatisfiable* if $\phi \vDash \mathsf{ff}$.

We require several binary relations between actions, detailed in the next subsection: *overlaps*, *strongly-overlaps*, *matches*, *strongly-matches*, *strongly-fences*, *blocks*, *sync-delays* and *co-delays*. We also require that there is a subsets of actions, distinguishing *read* and *release* actions, and an operator $\mathsf{merge} : \mathcal{A} \times \mathcal{A} \to 2^{\mathcal{A}}$.

1.2 Actions

We combine access and fence modes into a single order: $\mathsf{wk} \longrightarrow \mathsf{rlx} \rightrightarrows \begin{smallmatrix}\mathsf{rel}\\ \mathsf{acq}\end{smallmatrix} \rightrightarrows \mathsf{ra} \longrightarrow \mathsf{sc}$. 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]

Let $(\mathsf{W}^{\sqsupseteq\mathsf{rel}})$ and $(\mathsf{F}^{\sqsupseteq\mathsf{rel}})$ be *release* actions. Actions (R) are *read* actions.

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

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) $\mu_1, \mu_2 \neq \mathsf{wk}$,

(2b) if $\sigma_1 = \mathsf{grp}$ or $\sigma_2 = \mathsf{grp}$ then $\alpha_1 =_{\mathsf{grp}} \alpha_2$, and

(2c) if $\sigma_1 = \mathsf{proc}$ or $\sigma_2 = \mathsf{proc}$ then $\alpha_1 =_{\mathsf{proc}} \alpha_2$.

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. A *predicate transformer* is a function $\tau : \Phi \to \Phi$ such that

(x1) $\tau(\mathsf{ff})$ is ff,

(x2) $\tau(\psi_1 \wedge \psi_2)$ is $\tau(\psi_1) \wedge \tau(\psi_2)$,

(x3) $\tau(\psi_1 \vee \psi_2)$ is $\tau(\psi_1) \vee \tau(\psi_2)$,

(x4) if $\phi \vDash \psi$, then $\tau(\phi) \vDash \tau(\psi)$.

Definition 1.3. A *family of predicate transformers* for E consists of a predicate transformer τ^D for each $D \subseteq \mathcal{E}$, such that if $C \cap E \subseteq D$ then $\tau^C(\psi) \vDash \tau^D(\psi)$.

We write τ as an abbreviation of τ^E.

Definition 1.4. A *pomset with predicate transformers* is a tuple $(E, \lambda, \kappa, \tau, \checkmark, \trianglelefteq, \leq, \sqsubseteq, \mathsf{rmw})$ where

[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.

(M1) $E \subset \mathcal{E}$ is a set of *events*,
(M2) $\lambda : E \to \mathcal{A}$ defines a *label* for each event,
(M3) $\kappa : E \to \Phi$ defines a *precondition* for each event,
(M4) $\tau : 2^{\mathcal{E}} \to \Phi \to \Phi$ is a *family of predicate transformers* over E,
(M5) $\checkmark : \Phi$ defines a *termination condition*,
(M6) $\trianglelefteq : (E \times E)$ 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) rmw : $E \to E$ is a partial function capturing read-modify-write *atomicity*, such that
(M9a) if $d \xrightarrow{\text{rmw}} e$ then $\lambda(e)$ blocks $\lambda(d)$,
(M9b) if $d \xrightarrow{\text{rmw}} e$ then $d \leq e$ and $d \sqsubseteq e$,
(M9c) if $\lambda(c)$ overlaps $\lambda(d)$ then
(i) if $d \xrightarrow{\text{rmw}} e$ then $c \trianglelefteq e$ implies $c \trianglelefteq d$, $c \leq e$ implies $c \leq d$, $c \sqsubseteq e$ implies $c \sqsubseteq d$,
(ii) if $d \xrightarrow{\text{rmw}} e$ 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 rf : $E \times E$, capturing *reads-from*, such that
(C2) if $d \xrightarrow{\text{rf}} e$ then $\lambda(d)$ matches $\lambda(e)$,
(C6) if $d \xrightarrow{\text{rf}} e$ then $d \trianglelefteq e$,
(C7a) if $d' \leq d \xrightarrow{\text{rf}} e \leq e'$ 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 $e \leq d$,
(C8a) if $d \xrightarrow{\text{rf}} e$ then $d \sqsubseteq e$,
(C8b) if $d \xrightarrow{\text{rf}} e$ 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 *top-level* if
(T2) if $\lambda(e)$ is a read then there is some $d \xrightarrow{\text{rf}} e$,
(T3) $\kappa(e)$ is a tautology (for every $e \in E$),
(T5) $\checkmark$ is a tautology.

Note that for the IMM model, C8b is equivalent to:[2]

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

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

We lift terminology from actions to events. For example, we say that e writes x if $\lambda(e)$ writes x. We also drop quantifiers when clear from context, such as $(\forall e \in E)(\forall x \in \mathcal{X})$. We write $d < e$ when $d \leq e$ and $d \neq e$, and similarly for $\triangleleft$ and $\sqsubset$.

1.4 Semantics

See Figure 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:

- $e \rightarrow d$ arises from control/data/address *dependency* (S3),
- $e \rightarrow d$ arises from *sync-delays* (S7a),

[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 $\overset{\text{s}}{\sqsubseteq}$ for strong fulfillment and $\overset{\text{w}}{\sqsubseteq}$ for weak fulfillment.

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

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 $\kappa(e) \vDash \kappa_1(e)$,
(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) $\trianglelefteq\, \supseteq (\trianglelefteq_1 \cup \trianglelefteq_2)$,
(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 $\kappa(e) \vDash \kappa_1(e)$,
(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) $\checkmark \vDash \checkmark_1 \wedge \tau_1(\checkmark_2)$,
(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 $\kappa(e) \vDash \phi \wedge \kappa_1(e)$,
(I3b) if $e \in E_2 \setminus E_1$ then $\kappa(e) \vDash \neg\phi \wedge \kappa_2(e)$,
(I3c) if $e \in E_1 \cap E_2$ then $\kappa(e) \vDash (\phi \wedge \kappa_1(e)) \vee (\neg\phi \wedge \kappa_2(e))$,
(I4) $\tau^D(\psi) \vDash (\phi \wedge \tau_1^D(\psi)) \vee (\neg\phi \wedge \tau_2^D(\psi))$,
(I5) $\checkmark \vDash (\phi \wedge \checkmark_1) \vee (\neg\phi \wedge \checkmark_2)$.

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

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

(F1) if $d, e \in E$ then $d = e$,
(F2) $\lambda(e) = \alpha\mathsf{F}^\mu_\sigma$,
(F4) $\tau^D(\psi) \vDash \psi$,
(F5) if $E = \emptyset$ then $\checkmark \vDash \mathsf{ff}$.

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

(R1) if $d, e \in E$ then $d = e$,
(R2) $\lambda(e) = \alpha\mathsf{R}^\mu_\sigma x v$,
(R4a) if $(E \cap D) \neq \emptyset$ then $\tau^D(\psi) \vDash v{=}s_e \Rightarrow \psi[s_e/r]$,
(R4b) if $E \neq \emptyset$ and $(E \cap D) = \emptyset$ then $\tau^D(\psi) \vDash (v{=}s_e \vee x{=}s_e) \Rightarrow \psi[s_e/r]$,
(R4c) if $E = \emptyset$ then $(\forall s)\tau^D(\psi) \vDash \psi[s/r]$,
(R5) if $E = \emptyset$ and $\mu \sqsupseteq \mathsf{acq}$ then $\checkmark \vDash \mathsf{ff}$.

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

(W1) if $d, e \in E$ then $d = e$,
(W2) $\lambda(e) = \alpha\mathsf{W}^\mu_\sigma x v$,
(W3) $\kappa(e) \vDash M{=}v$,
(W4) $\tau^D(\psi) \vDash \psi[M/x]$,
(W5a) if $E = \emptyset$ then $\checkmark \vDash \mathsf{ff}$,
(W5b) if $E \neq \emptyset$ then $\checkmark \vDash M{=}v$.

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

Fig. 1. Semantics of programs

- $e \dashrightarrow d$ arises from *co-delays* (s8a),
- $e \longrightarrow d$ arises from *matching* (c6), (c7a) and (c8a),
- $e \longrightarrow d$ arises from *strong fencing* (c7b),

- $e \dashrightarrow d$ arises from *blocking* (c8b).

Definition 1.5. Address Calculation.

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

(w1) if $d, e \in E$ then $d = e$,

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

(w3) $\kappa(e) \vDash L{=}\ell \wedge M{=}v$,

(w4a) if $E \neq \emptyset$ then $\tau^D(\psi) \vDash (L{=}\ell) \Rightarrow \psi[M/[\ell]]$,

(w4b) if $E = \emptyset$ then $(\forall k)\ \tau^D(\psi) \vDash (L{=}k) \Rightarrow \psi[M/[k]]$

(w5a) if $E \neq \emptyset$ then $\checkmark \vDash L{=}\ell \wedge M{=}v$,

(w5b) if $E = \emptyset$ then $\checkmark \vDash \mathsf{ff}$.

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

(r1) if $d, e \in E$ then $d = e$,

(r2) $\lambda(e) = \alpha \mathsf{R}^{\mu}_{\sigma}[\ell]v$

(r3) $\kappa(e) \wedge L{=}\ell$,

(r4a) $(\forall e \in E \cap D)\ \tau^D(\psi) \vDash (L{=}\ell \Rightarrow v{=}s_e) \Rightarrow \psi[s_e/r]$,

(r4b) $(\forall e \in E \setminus D)\ \tau^D(\psi) \vDash ((L{=}\ell \Rightarrow v{=}s_e) \vee (L{=}\ell \Rightarrow [\ell]{=}s_e)) \Rightarrow \psi[s_e/r]$,

(r4c) $(\forall s)$ if $E = \emptyset$ then $\tau^D(\psi) \vDash \psi[s/r]$,

(r5) if $E = \emptyset$ and $\mu \neq \mathsf{rlx}$ then $\checkmark \vDash \mathsf{ff}$.

Definition 1.6. If-closure

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

(w1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

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

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

(w4) $\tau^D(\psi) \vDash \theta_e \Rightarrow \psi[M/x]$,

(w5) $\checkmark \vDash \theta_e \Rightarrow M{=}v_e$,

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

(r1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

(r2) $\lambda(e) = \alpha \mathsf{R}^{\mu}_{\sigma} x v_e$

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

(r5) if $E = \emptyset$ and $\mu \neq \mathsf{rlx}$ then $\checkmark \vDash \mathsf{ff}$.

Definition 1.7. Both.

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

(w1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

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

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

(w4a) $\tau^D(\psi) \vDash \theta_e \Rightarrow (L{=}\ell) \Rightarrow \psi[M/[\ell]]$,

(w4b) $(\forall k)$ $\tau^D(\psi) \vDash (\bigwedge_{e \in E} \neg\theta_e) \Rightarrow (L{=}k) \Rightarrow \psi[M/[k]]$

(w5a) $\checkmark \vDash \theta_e \Rightarrow L{=}\ell_e \wedge M{=}v_e$,

(w5b) $\checkmark \vDash \bigvee_{e \in E} \theta_e$.

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

(r1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,

(r2) $\lambda(e) = \alpha \mathsf{R}^{\mu}_{\sigma}[\ell]v_e$

(r3) $\kappa(e) \vDash \theta_e \wedge L{=}\ell_e$,

(r4a) $(\forall e \in E \cap D)\ \tau^D(\psi) \vDash \theta_e \Rightarrow (L{=}\ell_e \Rightarrow v_e{=}s_e) \Rightarrow \psi[s_e/r]$,

(r4b) $(\forall e \in E \setminus D)\ \tau^D(\psi) \vDash \theta_e \Rightarrow ((L{=}\ell_e \Rightarrow v_e{=}s_e) \vee (L{=}\ell_e \Rightarrow [\ell]{=}s_e)) \Rightarrow \psi[s_e/r]$,

(r4c) $(\forall s)\ \tau^D(\psi) \vDash (\bigwedge_{e \in E} \neg\theta_e) \Rightarrow \psi[s/r]$,

(r5) if $E = \emptyset$ and $\mu \neq \mathsf{rlx}$ then $\checkmark \vDash \mathsf{ff}$.

Definition 1.8. Let READ' be defined as for READ, adding the constraint:

(r4d) if $(E \cap D) = \emptyset$ then $\tau^D(\psi) \vDash \psi$.

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 CONTEXTUAL REFINEMENT

Lemma 2.1. *$(\mathcal{P}_1 ; \mathcal{P}_2) ; \mathcal{P}_3 = \mathcal{P}_1 ; (\mathcal{P}_2 ; \mathcal{P}_3)$ and $\mathcal{P} ; \mathit{SKIP} = \mathcal{P} = \mathit{SKIP} ; \mathcal{P}$.*
$(\mathcal{P}_1 \mathbin{]\!]} \mathcal{P}_2) \mathbin{]\!]} \mathcal{P}_3 = \mathcal{P}_1 \mathbin{]\!]} (\mathcal{P}_2 \mathbin{]\!]} \mathcal{P}_3)$ and $\mathcal{P} \mathbin{]\!]} \mathit{SKIP} = \mathcal{P}$.

Proof. Straightforward calculation. Associativity of ; requires disjunction closure (x3). □

Definition 2.2. 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) $\tau_2^D(\psi) \vDash \tau_1^D(\psi)$,
(5) $\checkmark_2 \vDash \checkmark_1$,
(6) $\trianglelefteq_2 \supseteq \trianglelefteq_1$,
(7) $\leq_2 \supseteq \leq_1$,
(8) $\sqsubseteq_2 \supseteq \sqsubseteq_1$,
(9) $\mathsf{rmw}_2 \supseteq \mathsf{rmw}_1$.

Lemma 2.3. *If $P_1 \in [\![S]\!]$ and P_2 augments P_1 then $P_2 \in [\![S]\!]$.*

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

This should be okay:

Define contexts as follows.

$$\mathbb{C} ::= [-] \mid \mathtt{if}(M)\{\mathbb{C}\}\mathtt{else}\{S\} \mid \mathtt{if}(M)\{S\}\mathtt{else}\{\mathbb{C}\} \mid \mathbb{C}; S \mid S; \mathbb{C} \mid \mathbb{C} \mathbin{]\!]_\gamma} S \mid S \mathbin{]\!]_\gamma} \mathbb{C}$$

A *test* is a context that can dereference 0. We disallow dereferencing 0 in user programs.

Definition 2.4. Let $\mathcal{P}{\Downarrow}$ if there is some top-level pomset in $\mathcal{P}$ that includes $(\mathsf{W}[0])$.
Let $S_1 \geq S_2$ if $(\forall \mathbb{C})\ [\![\mathbb{C}[S_2]]\!]{\Downarrow}$ implies $[\![\mathbb{C}[S_1]]\!]{\Downarrow}$.
Let $\mathcal{P}_1 > \mathcal{P}_2$ if $\mathcal{P}_1 \geq \mathcal{P}_2$ and $\mathcal{P}_2 \not\geq \mathcal{P}_1$.
Let $\mathcal{P}_1 =_{\mathsf{obs}} \mathcal{P}_2$ if $\mathcal{P}_1 \geq \mathcal{P}_2$ and $\mathcal{P}_2 \geq \mathcal{P}_1$.

For example, $[\![x := 1; x := 1]\!] > [\![x := 1]\!]$. These are distinguished by the test:

$$[-] \mathbin{]\!]} r := x ; x := 2; s := x ; \mathtt{if}(r{=}s)\{[0] := 1\}$$

$$(\mathsf{R}x1) \dashrightarrow (\mathsf{W}x2) \dashrightarrow (\mathsf{R}x1) \qquad (\mathsf{W}[0]1) \tag{$\sqsubseteq$}$$

This is top-level when put in parallel with $(\mathsf{W}x1) \dashrightarrow (\mathsf{W}x1)$ from the left. But there is no pomset from the right $[\![x := 1]\!]$ that makes this top-level.

Contextual refinement is sensitive to register names: $[\![\mathtt{skip}]\!] > [\![r := 1]\!]$. These are distinguished by the context $[-] ; [0] := r$.

We should also have $[\![r := x ; r := x]\!] =_{\mathsf{obs}} [\![r := x]\!]$.

Lemma 2.5. *If $\mathcal{P}_1 \supseteq \mathcal{P}_2$ then $\mathcal{P}_1 \geq \mathcal{P}_2$.*

This is too fine:

Definition 2.6. Let $P_1 \Downarrow P_2$ if there is some top-level pomset in $\mathit{PAR}(P_1,\ P_2)$.
Let $\mathcal{P}_1 \geq \mathcal{P}_2$ if $(\forall P_2 \in \mathcal{P}_2)$ $(\exists P_1 \in \mathcal{P}_1)$ such that $\tau_1(\psi) \vDash \tau_2(\psi)$ and $(\forall P_3)$ $P_1 \Downarrow P_3$ implies $P_2 \Downarrow P_3$.
Let $\mathcal{P}_1 > \mathcal{P}_2$ if $\mathcal{P}_1 \geq \mathcal{P}_2$ and $\mathcal{P}_2 \not\geq \mathcal{P}_1$.
Let $\mathcal{P}_1 =_{\mathsf{obs}} \mathcal{P}_2$ if $\mathcal{P}_1 \geq \mathcal{P}_2$ and $\mathcal{P}_2 \geq \mathcal{P}_1$.

For this, we have $[\![r := x ; r := x]\!] > [\![r := x]\!]$. To see that $[\![r := x ; r := x]\!] \not\leq [\![r := x]\!]$, consider that the left side includes pomset $P_2 = ((\mathsf{R}x1)(\mathsf{R}x2))$. There is no pomset P_1 from the right such that $P_1 \Downarrow P_3$ implies $P_2 \Downarrow P_3$, for every P_3. Suppose we pick $P_1 = (\mathsf{R}x1)$, then $P_3 = (\mathsf{W}x1)$ fulfills P_1 but not P_2. The same is true if we pick $P_1 = (\mathsf{R}x2)$.

3 MERGE

Let $\mathsf{merge} : \mathcal{A} \times \mathcal{A} \to 2^{\mathcal{A}}$ be defined as follows. Let $\mathsf{merge}(\alpha \mathsf{R}^{\mu}_{\sigma} x v,\ \alpha \mathsf{R}^{\nu}_{\rho} x v) = \{\alpha \mathsf{R}^{\mu \sqcup \nu}_{\sigma \sqcup \rho} x v\}$, $\mathsf{merge}(\alpha \mathsf{W}^{\mu}_{\sigma} x v,\ \alpha \mathsf{W}^{\nu}_{\rho} x v) = \{\alpha \mathsf{W}^{\mu \sqcup \nu}_{\sigma \sqcup \rho} x v\}$, $\mathsf{merge}(\mathsf{W}^{\mu}_{\sigma} x v,\ \alpha \mathsf{R}^{\nu \sqsubseteq \mathsf{rlx}}_{\rho \sqsubseteq \sigma} x v) = \{\alpha \mathsf{W}^{\mu \sqcup \nu}_{\sigma} x v\}$, $\mathsf{merge}(\alpha \mathsf{F}^{\mu}_{\sigma},\ \alpha \mathsf{F}^{\nu}_{\rho}) = \{\alpha \mathsf{F}^{\mu \sqcup \nu}_{\sigma \sqcup \rho}\}$, and $\mathsf{merge}(a,\ b) = \emptyset$, otherwise.

If $a_0 \in \mathsf{merge}(a_1,\ a_2)$, 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 $(\mathsf{W} x v) \in \mathsf{merge}(\mathsf{W} x v,\ \mathsf{R}^{\mathsf{acq}} x v)$, although this is considered safe.[3]

(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 $\lambda(e) \in \mathsf{merge}(\lambda_1(e),\ \lambda_2(e))$,

4 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.

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

$$\boxed{\alpha \mathsf{W} x 1} \to \boxed{\alpha \mathsf{W}^{\mathsf{rel}} y 1} \longrightarrow \boxed{\alpha \mathsf{R}^{\mathsf{acq}} y 1} \to \boxed{\alpha \mathsf{W}_{\mathsf{sys}} z 1} \dashrightarrow \boxed{\gamma \mathsf{R}^{\mathsf{acq}}_{\mathsf{sys}} z 1} \to \boxed{\gamma \mathsf{R} x 0} \qquad (\leq)$$

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

$$\boxed{\mathsf{W} x 1} \to \boxed{\mathsf{W}^{\mathsf{rel}} y 1} \longrightarrow \boxed{\mathsf{R}^{\mathsf{acq}} y 1} \to \boxed{\mathsf{W} z 1} \dashrightarrow \boxed{\mathsf{R}^{\mathsf{acq}} z 1} \to \boxed{\mathsf{R} x 0} \qquad (\leq)$$

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

$$\boxed{\mathsf{W} x 1} \to \boxed{\mathsf{W}^{\mathsf{rel}} y 1} \dashrightarrow \boxed{\mathsf{R} y 1} \to \boxed{\mathsf{W}^{\mathsf{rel}} z 1} \longrightarrow \boxed{\mathsf{R}^{\mathsf{acq}} z 1} \to \boxed{\mathsf{R} x 0} \qquad (\leq)$$

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

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

$$\boxed{\mathsf{W} x 1} \to \boxed{\mathsf{F}^{\mathsf{rel}}} \to \boxed{\mathsf{W} y 1} \dashrightarrow \boxed{\mathsf{R} y 1} \to \boxed{\mathsf{F}^{\mathsf{acq}}} \to \boxed{\mathsf{R} x 0} \qquad (\leq)$$

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

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

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

$$\begin{tikzcd} \boxed{\mathsf{W} x 1} \arrow[r] & \boxed{\mathsf{F}^{\mathsf{rel}}} \arrow[r] \arrow[rrr, Rightarrow, bend right] & \boxed{\mathsf{W} y 1} \arrow[r, dashed] & \boxed{\mathsf{R} y 1} \arrow[r] & \boxed{\mathsf{F}^{\mathsf{acq}}} \arrow[r] & \boxed{\mathsf{R} x 0} \end{tikzcd} \qquad (\leq)$$

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 $d' \leq_{\mathsf{po}} d \xrightarrow{\mathsf{rf}} e \leq_{\mathsf{po}} e'$ and $\lambda(d')$ strongly-matches $\lambda(e')$ then $d' \leq e'$.

[3] A list of safe merge operations 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].

where $\leq_{\mathrm{po}}$ is the obvious restriction of $\leq$ to actions on the same thread.

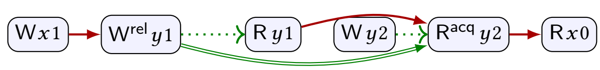
- With either (c7a′) or (c7a″) is it too strong to require $\leq$ that be transitive? In particular:
  - if we restrict to $\leq_{\mathrm{po}}$, the closure condition (c7a″) could add order between actions on the same thread via cross-thread reads.
  - How does transitivity interact with scopes?

Anton proposes:

(m9b′) if $d \xrightarrow{\mathsf{rmw}} e$ then $d \sqsubseteq e$,
(c7a‴) if $d' \leq d$ $(\xrightarrow{\mathsf{rf}}; (\xrightarrow{\mathsf{rmw}}; \xrightarrow{\mathsf{rf}})^*)$ $e \leq e'$ and $\lambda(d')$ strongly-matches $\lambda(e')$ then $d' \leq e'$.

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).

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



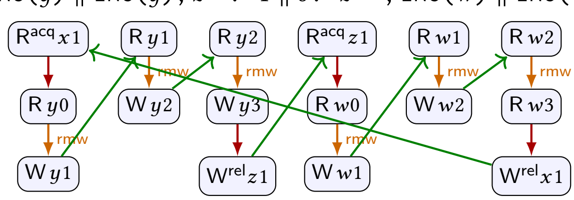
(≤)

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:

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

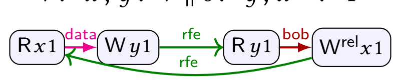
$$r := x^{\mathsf{acq}}\,;\; \mathtt{INC}\,(y) \parallel \mathtt{INC}\,(y) \parallel \mathtt{INC}\,(y)\,;\; z^{\mathsf{rel}} := 1 \parallel s := z^{\mathsf{acq}}\,;\; \mathtt{INC}\,(w) \parallel \mathtt{INC}\,(w) \parallel \mathtt{INC}\,(w)\,;\; x^{\mathsf{rel}} := 1$$

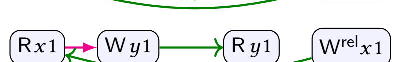


5 RELATING IMM AND PTX

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:

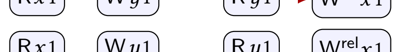
$$r := x\,;\; y := r \parallel s := y\,;\; x^{\mathsf{rel}} := 1$$





(⊴)

(≤)



(⊑)

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.

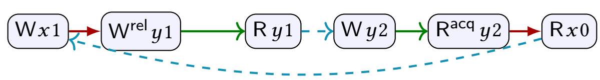
The problem is related to the one we have already discussed in the context of the C++ model – if you don't have acquire reads in the program, then you can erase release annotations from writes. In this regard, PTX is closer to PL memory models than to hardware ones.

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).

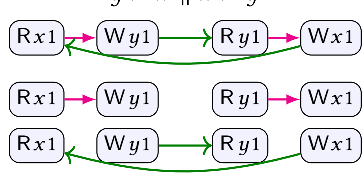
$$x := 1;\ y^{\mathsf{rel}} := 1 \parallel r := y;\ y := 2;\ s := y^{\mathsf{acq}};\ t := x$$



6 THIN AIR

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

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



($\trianglelefteq$)

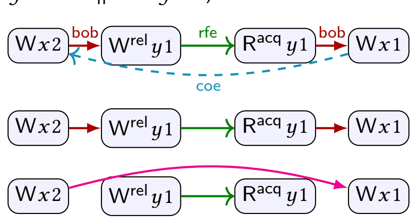
($\leq$)

($\sqsubseteq$)

7 IMM EXAMPLES

Disallowed by IMM:

$$x := 2;\ y^{\mathsf{rel}} := 1 \parallel r := y^{\mathsf{acq}};\ x := 1 \qquad \text{(PUB-REL-ACQ-COE)}$$



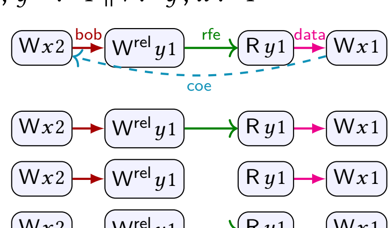
(✗IMM)

($\trianglelefteq\ =\ \leq$)

($\sqsubseteq$)

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

$$x := 2;\ y^{\mathsf{rel}} := 1 \parallel r := y;\ x := 1 \qquad \text{(PUB-REL-RLX-COE)}$$



(✓IMM)

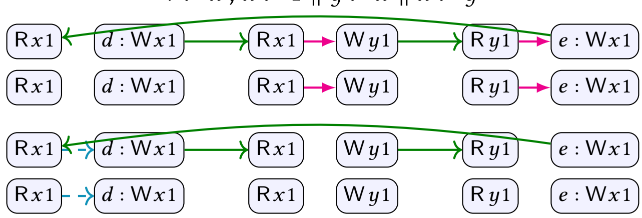
($\trianglelefteq$)

($\leq$)

($\sqsubseteq$)

Example from talk:

$$r := x\,;\; x := 1 \parallel y := x \parallel x := y \qquad \text{(ARM7-WEAK)}$$



(⊴)
(≤)
($\sqsubseteq_S$)
($\sqsubseteq_W$)

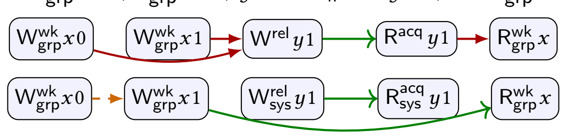
8 PTX EXAMPLES

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

In examples, all threads in different grps.

$(\mathsf{R}\,x0)$ must be forbidden. Before fulfilling the read:

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

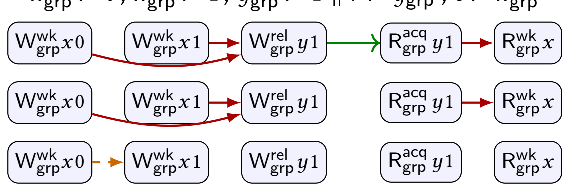


(⊴ = ≤)
(⊑)

$(\mathsf{W}\,x1) \precsim (\mathsf{R}\,x)$ is required by c8b, enforcing publication.

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

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



(⊴)
(≤)
(⊑)

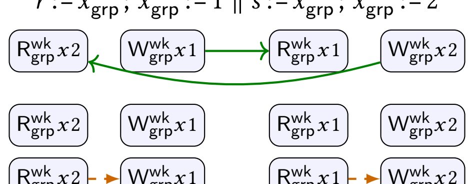
We do not have $(\mathsf{W}^{\mathsf{rel}} y1) \leq (\mathsf{R}^{\mathsf{acq}} y1)$ since c7a only requires order for things that are morally strong.

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

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

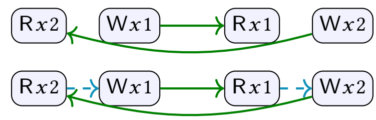
PTX allows TC16 for events that are not mutually strong (TC16_{WK}), but disallows it when events are mutually strong (TC16_{SYS}). Note that ≤ imposes no requirements here. Fulfillment imposes no order. This example shows that c8b cannot be strengthened to replace $\precsim$ with ⊑.

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



(⊴)
(≤)
(⊑)

$$r := x\,;\ x := 1 \parallel s := x\,;\ x := 2 \qquad (\textsc{tc16}_{\textsc{sys}})$$

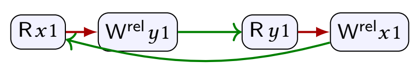


($\trianglelefteq$ = $\leq$)

($\sqsubseteq$)

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.

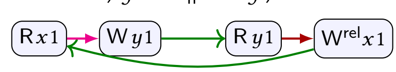
$$r := x\,;\ y^{\mathsf{rel}} := 1 \parallel s := y\,;\ x^{\mathsf{rel}} := 1 \qquad (\textsc{lb-rel})$$



($\trianglelefteq$ = $\leq$)

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.

$$r := x\,;\ y := r \parallel s := y\,;\ x^{\mathsf{rel}} := 1 \qquad (\textsc{lb-data-rel})$$

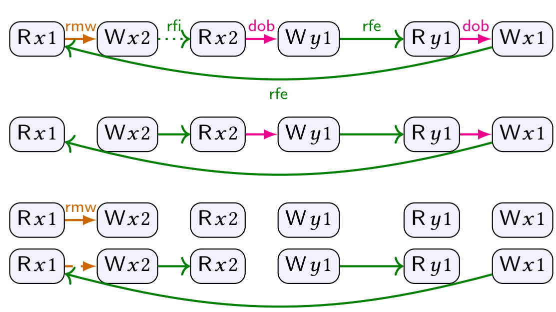


($\trianglelefteq$ = $\leq$)

9 RFI EXAMPLES

Bad example:

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



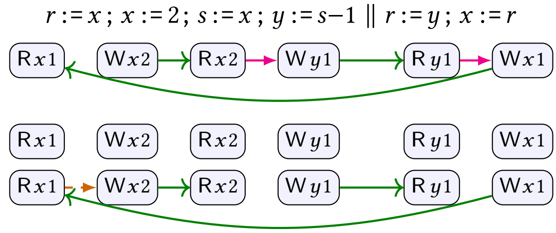
(✓Arm8)

($\trianglelefteq$)

($\leq$)

($\sqsubseteq$)

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



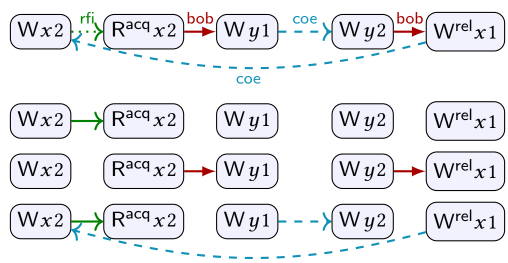
($\trianglelefteq$)

($\leq$)

($\sqsubseteq$)

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

$$x := 2;\ r := x^{\mathsf{acq}};\ y := 1 \parallel y := 2;\ x^{\mathsf{rel}} := 1 \qquad (\textsc{rfi-coe-coe})$$



(✓Arm8)

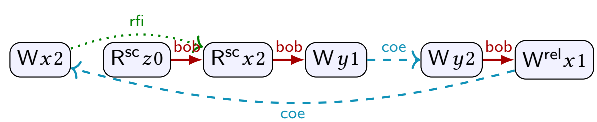
(⊴)

(≤)

(⊑)

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

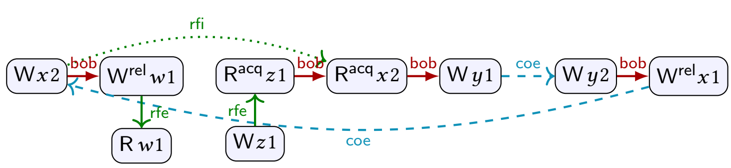
$$x := 2;\ s := z^{\mathsf{sc}};\ r := x^{\mathsf{sc}};\ y := 1 \parallel y := 2;\ x^{\mathsf{rel}} := 1 \qquad (\textsc{rfi-acq-coe-coe})$$



(✓Arm8)

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

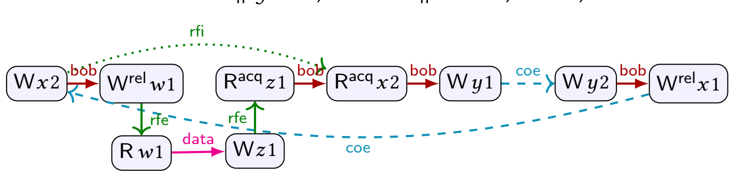
$$x := 2;\ w^{\mathsf{rel}} := 1;\ s := z^{\mathsf{acq}};\ r := x^{\mathsf{acq}};\ y := 1 \qquad (\textsc{rfi-ra-coe-coe2})$$
$$\parallel y := 2;\ x^{\mathsf{rel}} := 1 \parallel r := w;\ z := 1;$$



(✓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:

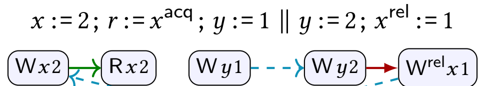
$$x := 2;\ w^{\mathsf{rel}} := 1;\ s := z^{\mathsf{acq}};\ r := x^{\mathsf{acq}};\ y := 1 \qquad (\textsc{rfi-ra-data-coe-coe})$$
$$\parallel y := 2;\ x^{\mathsf{rel}} := 1 \parallel r := w;\ z := r;$$



(✗Arm8)

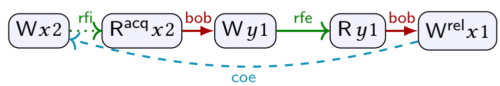
To allow this, weaken ra to rlx when read fulfilled by relaxed write of same thread (don't need to allow this when the write is part of an RMW).

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



RF variant [rfi-rfe-coe]:

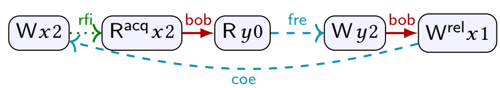
$$x := 2;\ r := x^{\mathsf{acq}};\ y := 1 \parallel s := y;\ x^{\mathsf{rel}} := 1 \qquad \text{(RFI-RFE-COE)}$$



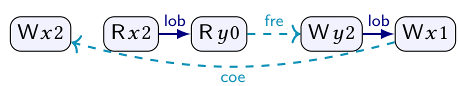
(✓Arm8)

TSO variant [rfi-fre-coe2]:

$$x := 2;\ r := x^{\mathsf{acq}};\ s := y \parallel y := 2;\ x^{\mathsf{rel}} := 1 \qquad \text{(RFI-COE-COE2)}$$



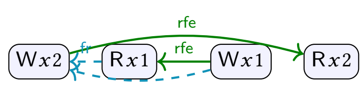
(✓Arm8)



(✓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]
- $d \xrightarrow{\text{po}} e$ when $c \xrightarrow{\text{rfe}} d \xrightarrow{\text{po}} e$

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

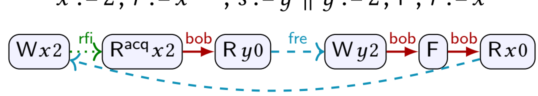
- [W]; po; [W]
- $d \xrightarrow{\text{po}} e$ when $c \xrightarrow{\text{rfe}} d \xrightarrow{\text{po}} e$, from `(range(rfe) * _)`
- [R]; po; [W], from `(rfi^-1; lob)`

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]:

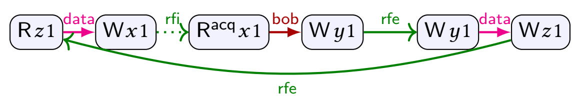
$$x := 2;\ r := x^{\mathsf{acq}};\ s := y \parallel y := 2;\ \mathsf{F};\ r := x \qquad \text{(RFI-FRE-FRE)}$$



(✓Arm8)

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

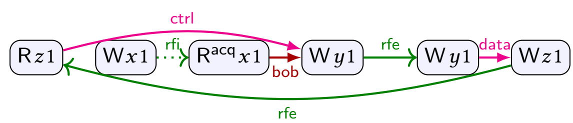
$$x := z;\ r := x^{\mathsf{acq}};\ y := 1 \parallel z := y \qquad (\text{DATA-RFI-RFE-RFE})$$



(✗Arm8)

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

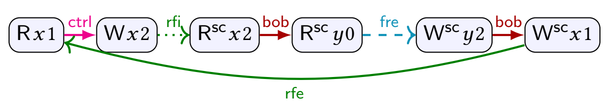
$$\texttt{if}(z)\{\};\ x := 1;\ r := x^{\mathsf{acq}};\ y := 1 \parallel z := y \qquad (\text{CTRL-RFI-RFE-RFE})$$



(✗Arm8)

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

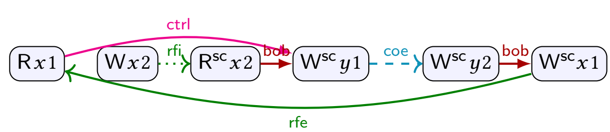
$$\texttt{if}(x)\{\};\ x := 2;\ r := x^{\mathsf{sc}};\ s := y^{\mathsf{sc}} \parallel y^{\mathsf{sc}} := 2;\ x^{\mathsf{sc}} := 1 \qquad (\text{CTRL-RFI-FRE-RFE})$$



(✓Arm8)

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

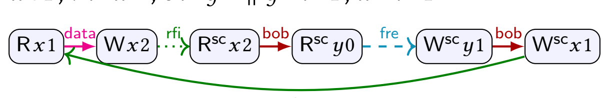
$$\texttt{if}(x)\{\};\ x := 2;\ r := x^{\mathsf{sc}};\ y^{\mathsf{sc}} := 1 \parallel y^{\mathsf{sc}} := 2;\ x^{\mathsf{sc}} := 1 \qquad (\text{CTRL-RFI-COE-RFE})$$



(✗Arm8)

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

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

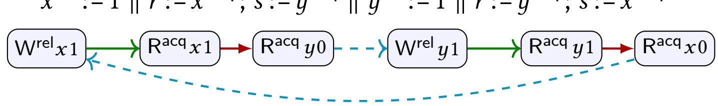


(✗Arm8)

10 SC EXAMPLES

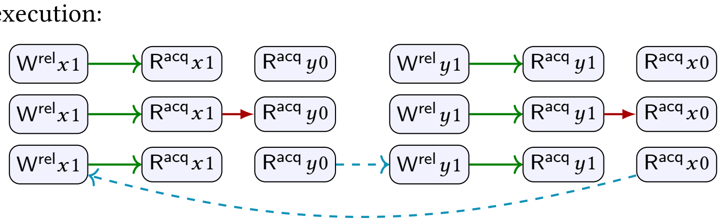
Example 10.1. Consider IRIW with all ra access:

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



(✓POWER,C11)

We allow this execution:



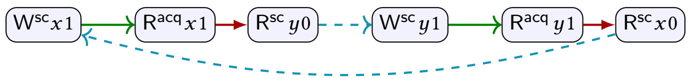
($\trianglelefteq$)

($\leq$)

($\sqsubseteq$)

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

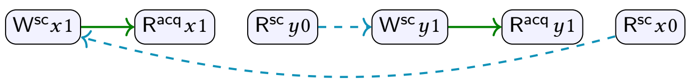
$$x^{\mathsf{sc}} := 1 \parallel r := x^{\mathsf{acq}};\ s := y^{\mathsf{sc}} \parallel y^{\mathsf{sc}} := 1 \parallel r := y^{\mathsf{acq}};\ s := x^{\mathsf{sc}} \tag{IRIW-ACQ-SC1}$$



(✓POWER,✗C11)

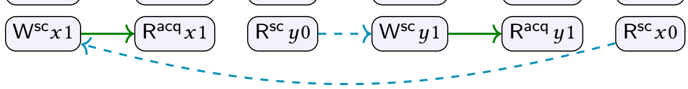
To model this it is convenient that synchronization is not included in dependency order:

- add sc bullet to def of $\sqsubseteq$ in c8b,
- add SC access to *sync-delays*.



($\trianglelefteq$)

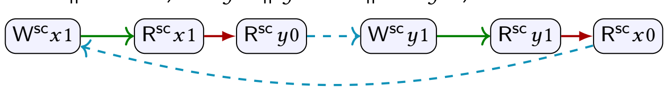
($\leq$)



($\sqsubseteq$)

This correctly forbids the all sc version:

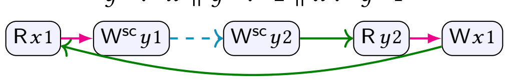
$$x^{\mathsf{sc}} := 1 \parallel r := x^{\mathsf{sc}};\ s := y^{\mathsf{sc}} \parallel y^{\mathsf{sc}} := 1 \parallel r := y^{\mathsf{sc}};\ s := x^{\mathsf{sc}} \tag{IRIW-SC-SC}$$



($\trianglelefteq$)

Example 10.2. Thin air with an SC antidependency:

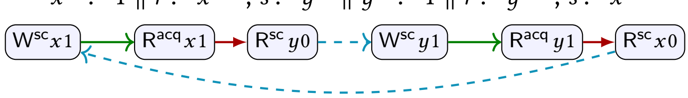
$$y^{\mathsf{sc}} := x \parallel y^{\mathsf{sc}} := 2 \parallel x := y - 1$$



($\trianglelefteq$)

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

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

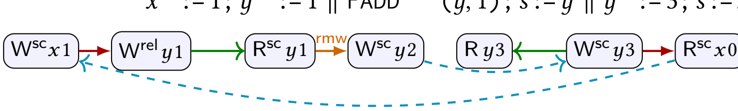


(✓POWER,RC11)

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].

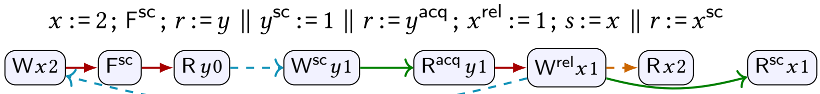
$$x^{\mathsf{sc}} := 1;\ y^{\mathsf{rel}} := 1 \parallel \mathsf{FADD}^{\mathsf{sc,sc}}(y, 1);\ s := y \parallel y^{\mathsf{sc}} := 3;\ s := x^{\mathsf{sc}} \tag{Z6.U}$$



(✓POWER,RC11)

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

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

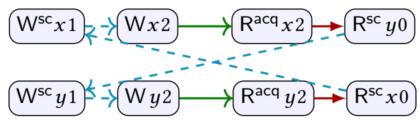


(✓RC11)

Fulfillment of ($\mathsf{R}x2$) requires that either ($\mathsf{W}^{\mathsf{rel}}x1$) ⇢ ($\mathsf{W}x2$) or ($\mathsf{R}x2$) ⇢ ($\mathsf{W}^{\mathsf{rel}}x1$). 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:

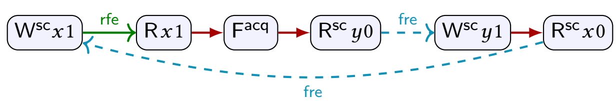
$$x^{\mathsf{sc}} := 1\,;\ x := 2 \parallel y^{\mathsf{sc}} := 1\,;\ y := 2 \parallel r := x^{\mathsf{acq}}\,;\ s := y^{\mathsf{sc}} \parallel r := y^{\mathsf{acq}}\,;\ s := x^{\mathsf{sc}} \qquad \text{(IRIW-SC-RLX-ACQ)}$$



(✓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].

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

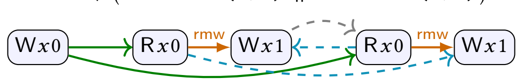


(✗Arm8)

11 ADDITIONAL RMW EXAMPLES

It is not possible for two RMWs to see the same write.

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

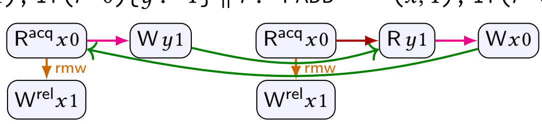


(RMW0)

The gray arrow is required the RMW atomicity axioms.

Lee et al. [2020] introduce PS2.0 to refine the treatment of RMWs in the promising semantics (PS). Their examples have the expected results here, with far less work. First they recall that PS requires quantification over multiple futures in order to disallow executions such as CDRF:

$$r := \mathsf{FADD}^{\mathsf{acq,rel}}(x, 1)\,;\ \mathtt{if}(r{=}0)\{y := 1\} \parallel r := \mathsf{FADD}^{\mathsf{acq,rel}}(x, 1)\,;\ \mathtt{if}(r{=}0)\{\mathtt{if}(y)\{x := 0\}\}$$

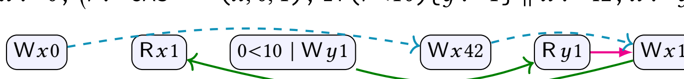


(CDRF)

This execution is clearly impossible, due to the cycle above. In this diagram, we have not drawn order adjacent to the writes of the RMWs, since this is not necessary to produce the cycle. If CDRF is allowed then DRF-RA fails.

PS does not support global value range analysis, as modeled by GA+E below. Our semantics permits GA+E:

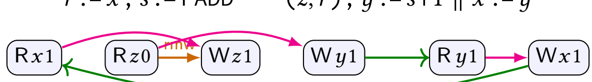
$$x := 0\,;\ \big(r := \mathsf{CAS}^{\mathsf{rlx,rlx}}(x, 0, 1)\,;\ \mathtt{if}(r{<}10)\{y := 1\} \parallel x := 42\,;\ x := y\big)$$



(GA+E)

PS also does not support register promotion, as modeled by RP below. Our semantics permits RP:

$$r := x\,;\ s := \mathsf{FADD}^{\mathsf{rlx,rlx}}(z, r)\,;\ y := s{+}1 \parallel x := y$$

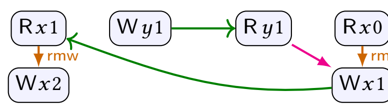


(RP)

These following examples are from "Modular Data-Race-Freedom Guarantees in the Promising Semantics" to appear in PLDI21.

CDRF shows that our semantics is not too permissive for ra-RMWs. But what about rlx-RMWs. The following execution is allowed by Arm8, and PS2.0, but disallowed by PS2.1.

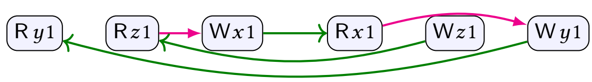
$$r := \mathsf{FADD}^{\mathsf{rlx},\mathsf{rlx}}(x, 1)\, ;\, y := 1 \parallel r := y\, ;\, s := \mathsf{FADD}^{\mathsf{rlx},\mathsf{rlx}}(x, r)$$



(RMW-W)

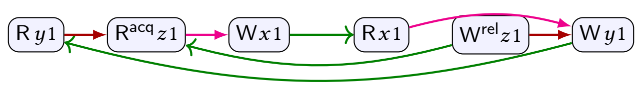
If this $\{z\}$-DRF-RA?

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



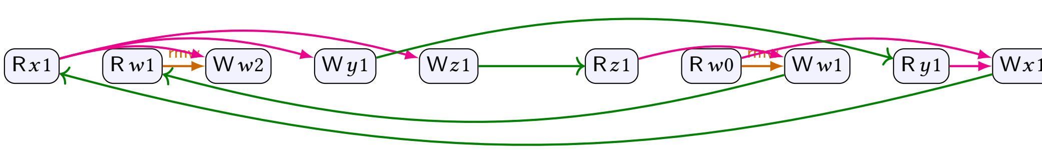
(NAIVE-LDRF-RA-FAIL)

Interpreting $\{z\}$ as ra:



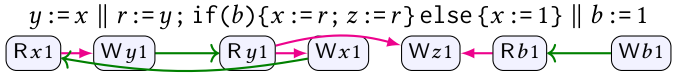
Our semantics already disallows LDRF-FAIL-PS, which is similar to OOTA4.

$$\mathtt{if}(x)\{\mathsf{FADD}(w, 1)\, ;\, y := 1\, ;\, z := 1\} \parallel \mathtt{if}(!z)\{x := 1\}\,\mathtt{else}\,\{\mathtt{if}(!\mathsf{FADD}(w, 1))\{x := y\}\}$$



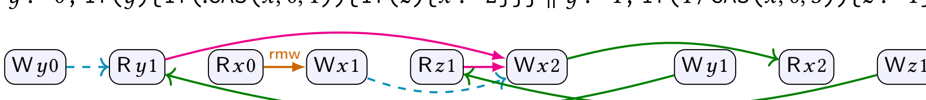
(LDRF-FAIL-PS)

$$y := x \parallel r := y\, ;\, \mathtt{if}(b)\{x := r\, ;\, z := r\}\,\mathtt{else}\,\{x := 1\} \parallel b := 1 \qquad \text{(OOTA4)}$$



If RMWs simply use the same semantics as read and write, then we allow LDRF-PF-FAIL, which is used to show failure of LDRF-SC.

$$y := 0\, ;\, \mathtt{if}(y)\{\mathtt{if}(!\mathsf{CAS}(x, 0, 1))\{\mathtt{if}(z)\{x := 2\}\}\} \parallel y := 1\, ;\, \mathtt{if}(1 \neq \mathsf{CAS}(x, 0, 3))\{z := 1\}$$



(LDRF-PF-FAIL)

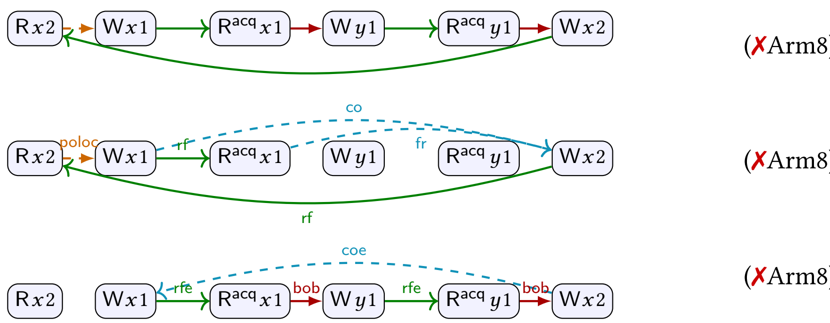
To disallow this, we need to retain the dependency $(\mathsf{R}x2) \rightarrow (\mathsf{W}z1)$. For this, we need to avoid the substitution for x. This is clearer in the LICS semantics. You just use L6 rather than L5 for the independent case on RMWs.

12 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"

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



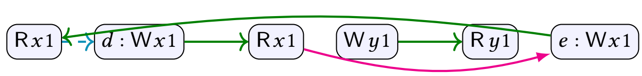
13 TWO ORDER IDEA

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 $\sqsubseteq_2$ for this notion, with strong fulfillment.

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



($\sqsubseteq_2$)

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.

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

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, 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

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

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.

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