1 MODEL

1.1 Preliminaries

The syntax is built from

- a set of *thread ids* $\mathcal{T}$, ranged over by α, γ,
- a set of *values* $\mathcal{V}$, ranged over by v, w, ℓ, k,
- a set of *registers* $\mathcal{R}$, ranged over by r, s,
- a set of *expressions* $\mathcal{M}$, ranged over by M, N, L.

Memory references are tagged values, written $[\ell]$. Let $\mathcal{X}$ be the set of memory references, ranged over by x, y, z.

We require that

- values and registers are disjoint,
- values include at least the constants 0 and 1,
- expressions include at least registers and values,
- expressions do *not* include references: $M[N/x] = M$,
- there are registers $\mathcal{S}_{\mathcal{E}} = \{s_e \mid e \in \mathcal{E}\}$,
- registers $\mathcal{S}_{\mathcal{E}}$ do not appear in programs: $S[N/s_e] = S$.

Alternative to the last assumption, we sometimes assume each register is assigned at most once.[1]

We model the following language.

$$\sigma, \rho ::= \mathsf{cta} \mid \mathsf{gpu} \mid \mathsf{sys}$$

$$\mu ::= \mathsf{wk} \mid \mathsf{rlx} \mid \mathsf{ra}$$

$$\nu ::= \mathsf{rel} \mid \mathsf{acq} \mid \mathsf{fsc}$$

$$S ::= \mathtt{skip} \mid r := M \mid r := [L]^{\mu}_{\sigma} \mid [L]^{\mu}_{\sigma} := M \mid \mathsf{F}^{\nu}_{\sigma} \mid \mathtt{if}(M)\{S_1\}\,\mathtt{else}\,\{S_2\} \mid S_1 ; S_2 \mid S_1 \ {}_{\gamma}\!\!\Vdash\!\!\rightarrow S_2$$
$$\mid r := \mathsf{CAS}([L]^{\mu_1,\mu_2}_{\sigma}, M, N) \mid r := \mathsf{FADD}([L]^{\mu_1,\mu_2}_{\sigma}, M) \mid r := \mathsf{EXCHG}([L]^{\mu_1,\mu_2}_{\sigma}, M)$$

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

Access modes, μ, are weak (wk), are relaxed (rlx), release-acquire (ra), and sequentially consistent (sc). ra/sc accesses are collectively known as *synchronized accesses*.

Fence modes, ν, are release (rel), acquire (acq), and sequentially consistent (fsc).

Commands, aka *statements*, S, include memory accesses at a given mode, as well as the usual structural constructs. Following [Ferreira et al. 1996], $\Vdash\!\!\rightarrow$ denotes parallel composition. If $(S_1 \ {}_{\gamma}\!\!\Vdash\!\!\rightarrow S_2)$ is executed with id α, then S_1 runs with id γ and S_1 continues under id α. Top level programs run with thread id 0. In examples, we usually drop thread ids. We use the symmetric $\|$ 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,
- an equivalence (?) $\mathsf{ms} \subseteq (\mathcal{A} \times \mathcal{A})$, defining *moral strength*.

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

We require that:

- actions include writes ($\alpha\mathsf{W}^{\mu}_{\sigma} x v$), reads ($\alpha\mathsf{R}^{\mu}_{\sigma} x v$), and fences ($\mathsf{F}^{\nu}_{\sigma}$),
- formulae include equalities ($M{=}N$) and ($x{=}M$),
- formulae include the write symbol W, and the downgrade symbols $\downarrow^{x}$,

[1] We make this assumption when discussing any semantics of load ($r := [L]^{\mu}_{\sigma}$) that does not include the substitution $[s_e/r]$.

- formulae are closed under negation, conjunction, disjunction, and substitutions $[M/r]$, $[M/x]$, and $[\phi/\mathsf{s}]$ for each symbol s,
- there is an entailment relation $\vDash$ between formulae,
- $\vDash$ has the expected semantics for $=$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and substitution.

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 *implies* ψ if $\phi \vDash \psi$. We say ϕ is a *tautology* if $\mathsf{tt} \vDash \phi$. We say ϕ is *unsatisfiable* if $\phi \vDash \mathsf{ff}$.

1.2 Label Relations

In definitions, we leave out elements of actions that are existentially quantified. In examples, we leave out elements of actions, using defaults.

Definition 1.1. We use the following terminology:

- Actions $(\mathsf{W}^{\mu\neq\mathsf{rlx}})$ and $(\mathsf{F}^{\nu\neq\mathsf{acq}})$ are *release actions*.
- Two actions *overlap* if they access the same location.
- Action $(\mathsf{W}xv)$ *matches* $(\mathsf{R}xw)$ when $v = w$.
- Action $(\mathsf{W}xv)$ *blocks* $(\mathsf{R}xw)$, for any v, w.

Definition 1.2. When modeling IMM, no action has mode wk. The default mode is rlx. Scopes and thread ids are ignored. Let $a \overset{\mathsf{ms}}{\longleftrightarrow} b$ when the overlap and neither has mode rlx.

Definition 1.3. When modeling PTX, the default mode is wk. The default scope is cta. The definition uses two equivalences:

- The equivalence $\mathsf{cta} \subseteq (\mathcal{T} \times \mathcal{T})$ groups threads by *thread group*.
- The equivalence $\mathsf{gpu} \subseteq (\mathcal{T} \times \mathcal{T})$ groups threads by *processor*.
- We require that $\alpha \overset{\mathsf{cta}}{\longleftrightarrow} \gamma$ implies $\alpha \overset{\mathsf{gpu}}{\longleftrightarrow} \gamma$.

Let $a \overset{\mathsf{ms}}{\longleftrightarrow} b$ when $a = (\alpha^{\mu}_{\sigma})$, $b = (\gamma^{\nu}_{\rho})$ and either $\alpha = \gamma$ or

- $\mu, \nu \neq \mathsf{wk}$,
- if $\sigma = \mathsf{cta}$ or $\rho = \mathsf{cta}$ then $\alpha \overset{\mathsf{cta}}{\longleftrightarrow} \gamma$,
- if $\sigma = \mathsf{gpu}$ or $\rho = \mathsf{gpu}$ then $\alpha \overset{\mathsf{gpu}}{\longleftrightarrow} \gamma$,
- if either action is an access then they overlap.

Definition 1.4. Reorderability relations.

$$
\begin{aligned}
\ltimes_{\mathsf{co}} &= \{(\mathsf{W}x, \mathsf{W}y),\ (\mathsf{R}x, \mathsf{W}y),\ (\mathsf{W}x, \mathsf{R}y) \mid x \neq y\} \ \cup\ \{(\mathsf{R}x, \mathsf{R}y)\} \\
\ltimes_{\mathsf{sync}} &= \{(\mathsf{W}^{\mu}, \mathsf{R}^{\nu})\} \cup \{(\mathsf{W}^{\mu}, \mathsf{W}^{\mathsf{rlx}})\} \cup \{(\mathsf{F}^{\mathsf{rel}}, \mathsf{R}^{\nu})\} \\
&\quad \cup \{(\mathsf{R}^{\mathsf{rlx}}, \mathsf{W}^{\mathsf{rlx}})\} \cup \{(\mathsf{R}^{\mathsf{rlx}}, \mathsf{R}^{\nu})\} \cup \{(\mathsf{W}^{\mu}, \mathsf{F}^{\mathsf{acq}})\} \cup \{(\mathsf{F}^{\mathsf{rel}}, \mathsf{F}^{\mathsf{acq}})\} \\
\ltimes &= \ltimes_{\mathsf{sync}} \cap \ltimes_{\mathsf{co}}
\end{aligned}
$$

			2^{nd}				
1^{st}	$\mathsf{R}^{\mathsf{rlx}}$	R^{ra}	$\mathsf{W}^{\mathsf{rlx}}$	W^{ra}	$\mathsf{F}^{\mathsf{rel}}$	$\mathsf{F}^{\mathsf{acq}}$	$\mathsf{F}^{\mathsf{fsc}}$
$\mathsf{R}^{\mathsf{rlx}}$	✓	✓	✓	✗	✗	✗	✗
R^{ra}	✗	✗	✗	✗	✗	✗	✗
$\mathsf{W}^{\mathsf{rlx}}$	✓	✓	✓	✗	✗	✓	✗
W^{ra}	✓	✓	✓	✗	✗	✓	✗
$\mathsf{F}^{\mathsf{rel}}$	✓	✓	✗	✗	✗	✓	✗
$\mathsf{F}^{\mathsf{acq}}$	✗	✗	✗	✗	✗	✗	✗
$\mathsf{F}^{\mathsf{fsc}}$	✗	✗	✗	✗	✗	✗	✗

We combine access and fence modes into a single order:

$$\mathsf{wk} \longrightarrow \mathsf{rlx} \longrightarrow \mathsf{ra} \qquad\qquad \begin{matrix} & & \mathsf{acq} & \searrow & \\ & & & & \mathsf{fsc} \\ & & \mathsf{rel} & \nearrow & \end{matrix}$$

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

Definition 1.5. Define $\prec : \mathcal{A} \times \mathcal{A} \to 2^{\mathcal{A}}$ as follows. If $a_0 \in a_1 \prec a_2$, then a_1 and a_2 can coalesce, resulting in a_0. Allows optimizations $(x := 1\,;\, x := 2)$ to $(x := 2)$ and $(x := 1\,;\, r := x)$ to $(x := 1\,;\, r := 1)$

$$\begin{aligned} \mathsf{R}^{\mu} x v \prec \mathsf{R}^{\nu} x v &= \{\mathsf{R}^{\mu \sqcup \nu} x v\} \\ \mathsf{W}^{\mu} x v \prec \mathsf{W}^{\nu} x w &= \{\mathsf{W}^{\mu \sqcup \nu} x w\} \\ \mathsf{W}^{\nu} x v \prec \mathsf{R}^{\mathsf{rlx}} x v &= \{\mathsf{W}^{\nu} x v\} \\ \mathsf{F}^{\mu} \prec \mathsf{F}^{\nu} &= \{\mathsf{F}^{\mu \sqcup \nu}\} \\ a \prec b &= \emptyset, \text{ otherwise} \end{aligned}$$

1.3 Pomsets with Predicate Transformers

Definition 1.6. A *predicate transformer* is a function $\tau : \Phi \to \Phi$ such that

(1) $\tau(\mathsf{ff})$ is ff,
(2) $\tau(\psi_1 \wedge \psi_2)$ is $\tau(\psi_1) \wedge \tau(\psi_2)$,
(3) $\tau(\psi_1 \vee \psi_2)$ is $\tau(\psi_1) \vee \tau(\psi_2)$,
(4) if ϕ implies ψ, then $\tau(\phi)$ implies $\tau(\psi)$.

Definition 1.7. 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 $\tau^C(\psi)$ implies $\tau^D(\psi)$.

Definition 1.8. A *pomset with predicate transformers* is a tuple $(E, \lambda, \kappa, \tau, \checkmark, \trianglelefteq, \leq, \sqsubseteq, \mathsf{rmw})$ where

(1) $E \subset \mathcal{E}$ is a set of *events*,
(2) $\lambda : E \to \mathcal{A}$ defines a *label* for each event,
(3) $\kappa : E \to \Phi$ defines a *precondition* for each event,
(4) $\tau : 2^{\mathcal{E}} \to \Phi \to \Phi$ defines a *predicate transformer* for each set of events,
(5) $\checkmark : \Phi$ defines a *termination condition*,
(6) $\trianglelefteq \subseteq (E \times E)$ is a partial order capturing *dependency*,
(7) $\leq \subseteq (E \times E)$ is a partial order capturing *synchronization*,
(8) $\sqsubseteq \subseteq (E \times E)$ is a partial order capturing *per-location order*, such that
 (8a) if $\lambda(d)$ and $\lambda(e)$ overlap then $d \leq e$ implies $d \sqsubseteq e$,
(9) $\mathsf{rmw} : E \to E$ is a partial function capturing read-modify-write *atomicity*, such that
 (9a) if $d \xrightarrow{\mathsf{rmw}} e$ then $d \leq e$ and $d \sqsubseteq e$,
 (9b) if $\lambda(c)$ and $\lambda(d)$ overlap then

- 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 $c \sqsubseteq d$,
- 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 $e \sqsubseteq c$,

(9c) if $d \xrightarrow{\mathsf{rmw}} e$ then $\lambda(e)$ blocks $\lambda(d)$.

A pomset is an IMM *candidate* if there is a partial function $\mathsf{rf} : E \to E$ such that:

(10a) if $d \xrightarrow{\mathsf{rf}} e$ then $d \trianglelefteq e$ and $d \sqsubseteq e$,
(10b) if $d \xrightarrow{\mathsf{rf}} e$ and $\lambda(d) \overset{\mathsf{ms}}{\longleftrightarrow} \lambda(e)$ then $d \leq e$,
(10c) if $d \xrightarrow{\mathsf{rf}} e$ then $\lambda(d)$ matches $\lambda(e)$,
(10d) if $d \xrightarrow{\mathsf{rf}} e$ and $\lambda(c)$ blocks $\lambda(e)$ then either $c \sqsubseteq d$ or $e \sqsubseteq c$.

A pomset is a PTX *candidate* if there is a partial function $\mathsf{rf} : E \to E$, satisfying conditions (10a)–(10c) above, such that:

(10d) if $d \xrightarrow{\mathsf{rf}} e$ and $\lambda(c)$ blocks $\lambda(e)$ then either $c \sqsubseteq\!\!\!\sim\, d$ or $e \sqsubseteq\!\!\!\sim\, c$,
where $d' \sqsubseteq\!\!\!\sim\, e'$ is $d' \sqsubseteq e'$ if $\lambda(d') \overset{\mathsf{ms}}{\longleftrightarrow} \lambda(e')$; otherwise $d' \sqsubseteq\!\!\!\sim\, e'$ is $e' \not\sqsubseteq d'$.

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

(11a) $\kappa(e)$ is a tautology,
(11b) if $\lambda(e)$ is a read then there is some $d \xrightarrow{\mathsf{rf}} e$.

Let P range over pomsets, and $\mathcal{P}$ over sets of pomsets. Let Pom be the set of all pomsets.

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

Definition 1.9. $\mathcal{P}_1$ *refines* $\mathcal{P}_2$ if $\mathcal{P}_1 \subseteq \mathcal{P}_2$.

1.4 Semantics

Definition 1.10. If $P \in \mathcal{P}_1 \between \mathcal{P}_2$ then $(\exists P_1 \in \mathcal{P}_1)\ (\exists P_2 \in \mathcal{P}_2)$
(1) $E = (E_1 \cup E_2)$, $\trianglelefteq \supseteq (\trianglelefteq_1 \cup \trianglelefteq_2)$, $\leq \supseteq (\leq_1 \cup \leq_2)$, $\sqsubseteq \supseteq (\sqsubseteq_1 \cup \sqsubseteq_2)$, $\mathsf{rmw} = (\mathsf{rmw}_1 \cup \mathsf{rmw}_2)$,
(2) $\lambda = (\lambda_1 \cup \lambda_2)$,
(3) if $e \in E_1$ then $\kappa(e)$ implies $\kappa_1(e)$,
(4) if $e \in E_2$ then $\kappa(e)$ implies $\kappa_2(e)$,
(5) $\tau^D(\psi)$ implies $\tau_2^D(\psi)$,
(6) E_1 and E_2 are disjoint,
(7) $\checkmark$ implies $\checkmark_1 \wedge \checkmark_2$.

If $P \in \mathcal{P}_1 ; \mathcal{P}_2$ then $(\exists P_1 \in \mathcal{P}_1)\ (\exists P_2 \in \mathcal{P}_2)$
(1) as for $\between$,
(2) if $e \in E_1 \setminus E_2$ then $\lambda(e) = \lambda_1(e)$,
(3) if $e \in E_2 \setminus E_1$ then $\lambda(e) = \lambda_2(e)$,
(4) if $e \in E_1 \cap E_2$ then $\lambda(e) \in \lambda_1(e) \prec \lambda_2(e)$,
(5) if $e \in E_1 \setminus E_2$ then $\kappa(e)$ implies $\kappa_1(e)$,
(6) if $e \in E_2 \setminus E_1$ then $\kappa(e)$ implies $\kappa_2'(e)$,
(7) if $e \in E_1 \cap E_2$ then $\kappa(e)$ implies $\kappa_1(e) \vee \kappa_2'(e)$, where $\kappa_2'(e) = \tau_1^{\downarrow e}(\kappa_2(e))$,
where $\downarrow e = \{c \mid c \lhd e\}$ if $\lambda(e)$ is a write, and $\downarrow e = E_1$, otherwise,
(8) if $d \in E_1$ and $e \in E_2$ then either $d \leq e$ or $\lambda_1(d) \ltimes_{\mathsf{sync}} \lambda_2(e)$,
(9) if $d \in E_1$ and $e \in E_2$ then either $d \sqsubseteq e$ or $\lambda_1(d) \ltimes_{\mathsf{co}} \lambda_2(e)$,
(10) if $e \in E_2$ and $\lambda(e)$ is a release then $\kappa(e)$ implies $\checkmark_1$,
(11) $\tau^D(\psi)$ implies $\tau_1^D(\tau_2^D(\psi))$,
(12) $\checkmark$ implies $\checkmark_1 \wedge \tau_1^{E_1}(\checkmark_2)$.

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

(1–2) as for $\between\!\!\!\rightarrow$,
(3) if $e \in E_1 \setminus E_2$ then $\kappa(e)$ implies $\phi \wedge \kappa_1(e)$,
(4) if $e \in E_2 \setminus E_1$ then $\kappa(e)$ implies $\neg\phi \wedge \kappa_2(e)$,
(5) if $e \in E_1 \cap E_2$ then $\kappa(e)$ implies $(\phi \Rightarrow \kappa_1(e)) \wedge (\neg\phi \Rightarrow \kappa_2(e))$,
(6) $\tau^D(\psi)$ implies $(\phi \Rightarrow \tau_1^D(\psi)) \wedge (\neg\phi \Rightarrow \tau_2^D(\psi))$,
(7) $\checkmark$ implies $(\phi \Rightarrow \checkmark_1) \wedge (\neg\phi \Rightarrow \checkmark_2)$.

If $P \in \mathit{LET}(r, M)$ then $E = \emptyset$ and $\tau^D(\psi)$ implies $\psi[M/r]$.
If $P \in \mathit{SKIP}$ then $E = \emptyset$ and $\tau^D(\psi)$ implies ψ.
If $P \in \mathit{FENCE}(\mu, \sigma)_\alpha$ then
(1) if $d, e \in E$ then $d = e$,
(2) $\lambda(e) = \mathsf{F}^\mu_\sigma$,
(3) $\tau^D(\psi)$ implies ψ,
(4) if $E = \emptyset$ then $\checkmark$ implies ff.

If $P \in \mathit{STORE}(x, M, \mu, \sigma)_\alpha$ then $(\exists v \in \mathcal{V})$
(1) if $d, e \in E$ then $d = e$,
(2) $\lambda(e) = \alpha\mathsf{W}^\mu_\sigma x v$,
(3) $\kappa(e)$ implies $M{=}v$,
(4) $\tau^D(\psi)$ implies ψ,
(5) if $E = \emptyset$ then $\checkmark$ implies ff,
(6) if $E \neq \emptyset$ then $\checkmark$ implies $M{=}v$.

If $P \in \mathit{LOAD}(r, x, \mu, \sigma)_\alpha$ then $(\exists v \in \mathcal{V})$
(1) if $d, e \in E$ then $d = e$,
(2) $\lambda(e) = \alpha\mathsf{R}^\mu_\sigma x v$,
(3) $\tau^D(\psi)$ implies $v{=}r \Rightarrow \psi$, if $(E \cap D) \neq \emptyset$,
(4) $\tau^D(\psi)$ implies ψ, if $(E \cap D) = \emptyset$.

$$\begin{aligned}
[\![\texttt{if(}M\texttt{)\{}S_1\texttt{\}else\{}S_2\texttt{\}}]\!]_\alpha &= \mathit{IF}(M{\neq}0, [\![S_1]\!]_\alpha, [\![S_2]\!]_\alpha) \\
[\![x^\mu := M]\!]_\alpha &= \mathit{STORE}(x, M, \mu, \sigma)_\alpha & [\![\texttt{skip}]\!]_\alpha &= \mathit{SKIP} \\
[\![r := x^\mu]\!]_\alpha &= \mathit{LOAD}(r, x, \mu, \sigma)_\alpha & [\![S_1 \ {}_\gamma\!\between\!\!\!\rightarrow\ S_2]\!]_\alpha &= [\![S_1]\!]_\gamma \between\!\!\!\rightarrow [\![S_2]\!]_\alpha \\
[\![r := M]\!]_\alpha &= \mathit{LET}(r, M) & [\![S_1\,;\, S_2]\!]_\alpha &= [\![S_1]\!]_\alpha\,;\, [\![S_2]\!]_\alpha \\
[\![\mathsf{F}^\nu_\sigma]\!]_\alpha &= \mathit{FENCE}(\nu, \sigma)_\alpha
\end{aligned}$$

Full versions (everything but address calculation):

If $P \in \mathit{STORE}(x, M, \mu, \sigma)_\alpha$ then $(\exists v : E \to \mathcal{V})$ $(\exists \theta : E \to \Phi)$
(1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,
(2) $\lambda(e) = \alpha\mathsf{W}^\mu_\sigma x v_e$,
(3) $\kappa(e)$ implies $\theta_e \wedge M{=}v_e$,
(4) $\tau^D(\psi)$ implies $\theta_e \Rightarrow \psi[M/x]$,
(5) $\checkmark$ implies $\bigvee_{e\in E} \theta_e$.

If $P \in \mathit{LOAD}(r, x, \mu, \sigma)_\alpha$ then $(\exists v : E \to \mathcal{V})$ $(\exists \theta : E \to \Phi)$
(1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,
(2) $\lambda(e) = \alpha\mathsf{R}^\mu_\sigma x v_e$
(3) $\kappa(e)$ implies θ_e,
(4) $(\forall e \in E \cap D)$ $\tau^D(\psi)$ implies $\theta_e \Rightarrow v_e{=}s_e \Rightarrow \psi[s_e/r]$,
(5) $(\forall e \in E \setminus D)$ $\tau^D(\psi)$ implies $\theta_e \Rightarrow (v_e{=}s_e \vee x{=}s_e) \Rightarrow \psi[s_e/r]$,
(6) $(\forall s)$ $\tau^D(\psi)$ implies $(\bigwedge_{e\in E} \neg\theta_e) \Rightarrow \psi[s/r]$.

1.5 Fulfillment

Definition 1.11. Define $\sqsubseteq\!\!\!\sim$ as follows.

$$d \sqsubseteq\!\!\!\sim e \text{ when } \begin{cases} d \sqsubseteq e & \text{if } d \text{ is morally strong with } e \\ e \not\sqsubset d & \text{otherwise} \end{cases}$$

A read event e is *strongly fulfilled* if there is a $d \xrightarrow{\text{rf}} e$ and

for any c that can block e, either $c \sqsubseteq d$ or $e \sqsubseteq c$.

A read event e is *weakly fulfilled* if there is a $d \xrightarrow{\text{rf}} e$ and

for any c that can block e, either $c \sqsubseteq\!\!\!\sim d$ or $e \sqsubseteq\!\!\!\sim c$.

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 $\boxed{\text{s}}$ for strong fulfillment and $\boxed{\text{w}}$ for weak fulfillment.

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:

- $e \longrightarrow d$ arises from *reads-from* (rf),
- $e \dashrightarrow d$ arises from *fulfillment*,
- $e \longrightarrow d$ arises from control/data/address *dependency*,
- $e \Longrightarrow d$ arises from *synchronized access*.

2 NOTES

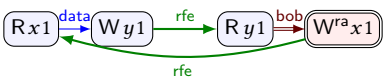
GPU stuff:

- Vulcan/Alloy
- OpenCL
- AMD PTX
- Matthew Sinclair/Sarita Adve stuff "Chasing Away RAts- Semantics and Evaluation for Relaxed Atomics on Heterogeneous Systems" and his thesis

3 ANTON'S RECENT EXAMPLES 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:

$$r := x \,;\, y := r \;\|\; s := y \,;\, x^{\text{ra}} := 1$$



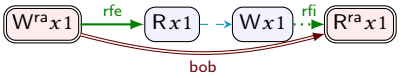
IMM forbids it, but PTX allows it. The point is that IMM mixes dependencies and release/acquire-induced po-order in its NoOOTA axiom, whereas PTX doesn't — release/acquire are only used to have coherence.

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.

Another oddity: PTX includes the bob edge below; IMM does not.

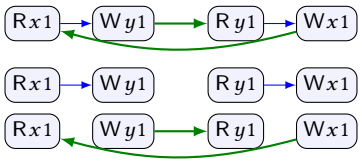
$$x^{\mathsf{ra}} := 1 \parallel r := x\,;\, x := 1\,;\, 1 := x^{\mathsf{ra}}$$



4 THIN AIR

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

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



($\trianglelefteq$)

($\leq$)

($\boxed{\mathsf{s}}$)

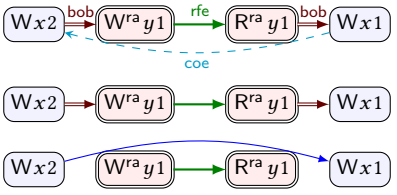
5 IMM EXAMPLES

Interpreting this definition for the IMM:

- No wk, default is rlx
- All threads in same cta (only one scope)
- Actions are morally strong when both are ra/sc, mimicking happens-before
- Strong fulfillment may do the right thing

Disallowed by IMM:

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



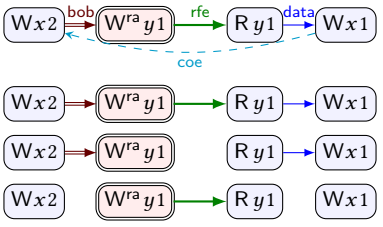
(✗IMM)

($\trianglelefteq = \leq$)

($\boxed{\mathsf{s}}$)

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

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



(✓IMM)

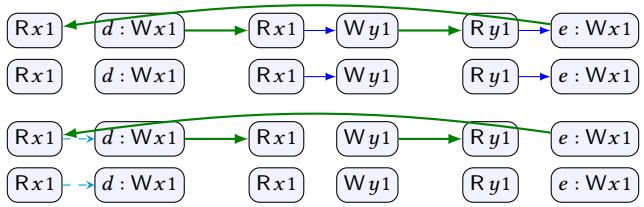
($\trianglelefteq$)

($\leq$)

($\boxed{\mathsf{s}}$)

Example from talk:

$r := x;\ x := 1 \parallel y := x \parallel x := y$ (ARM7-WEAK)



(⊴) (≤) (⊑ˢ) (⊑ʷ)

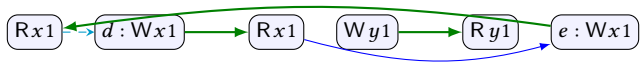
6 TWO ORDER IDEA

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 ⊑ᵖ for this notion, with strong fulfillment.

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



(⊑ᵖ)

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.

7 PTX EXAMPLES

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

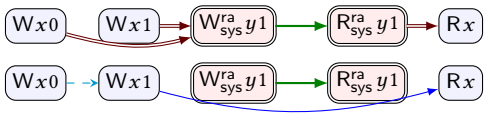
PTX requires weak fulfillment.

Default scope is cta. In examples, all threads in different ctas.

Default mode is wk.

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

$x := 0;\ x := 1;\ y^{\mathsf{ra}}_{\mathsf{sys}} := 1 \parallel r := y^{\mathsf{ra}}_{\mathsf{sys}};\ s := x$ ($\mathrm{PUB1}_{\mathrm{SYS}}$)

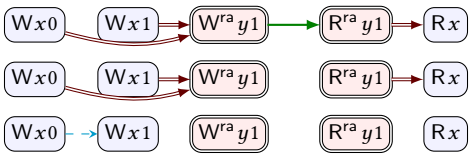


(⊴ = ≤) (⊑)

$(\mathsf{W}x1) \sqsubseteq_{\sim} (\mathsf{R}x)$ is required by M7, enforcing publication.

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

$x := 0;\ x := 1;\ y^{\mathsf{ra}} := 1 \parallel r := y^{\mathsf{ra}};\ s := x$ ($\mathrm{PUB1}_{\mathrm{CTA}}$)



(⊴) (≤) (⊑)

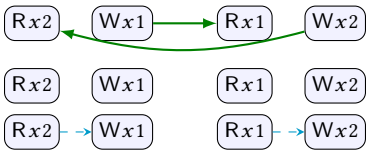
We do not have $(\mathsf{W}^{\mathsf{ra}}y1) \leq (\mathsf{R}^{\mathsf{ra}}y1)$ since F3 only requires order for things that are morally strong.

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

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

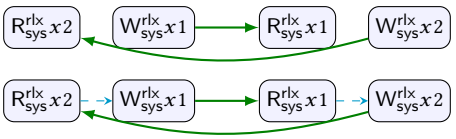
PTX allows TC16 for events that are not mutually strong (TC16_{WK}), but disallows it when events are mutually strong (TC16_{SYS}). Note that $\leq$ imposes no requirements here. Fulfillment imposes no order. This example shows that F3C cannot be strengthened to require that $d \sqsubseteq e$.

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



$(\trianglelefteq)$ $(\leq)$ $(\sqsubseteq)$

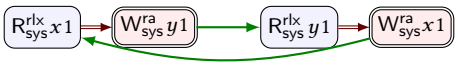
$$r := x^{\mathsf{rlx}}_{\mathsf{sys}}\,;\, x^{\mathsf{rlx}}_{\mathsf{sys}} := 1 \parallel s := x^{\mathsf{rlx}}_{\mathsf{sys}}\,;\, x^{\mathsf{rlx}}_{\mathsf{sys}} := 2 \qquad (\text{TC16}_{\text{SYS}})$$



$(\trianglelefteq = \leq)$ $(\sqsubseteq)$

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.

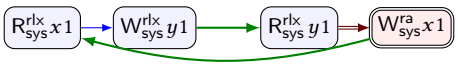
$$r := x^{\mathsf{rlx}}_{\mathsf{sys}}\,;\, y^{\mathsf{ra}}_{\mathsf{sys}} := 1 \parallel s := y^{\mathsf{rlx}}_{\mathsf{sys}}\,;\, x^{\mathsf{ra}}_{\mathsf{sys}} := 1 \qquad (\text{LB-REL})$$



$(\trianglelefteq = \leq)$

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.

$$r := x^{\mathsf{rlx}}_{\mathsf{sys}}\,;\, y^{\mathsf{rlx}}_{\mathsf{sys}} := r \parallel s := y^{\mathsf{rlx}}_{\mathsf{sys}}\,;\, x^{\mathsf{ra}}_{\mathsf{sys}} := 1 \qquad (\text{LB-DATA-REL})$$

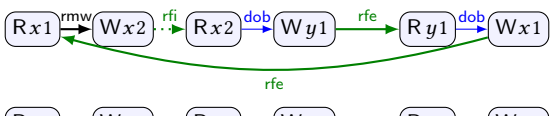


$(\trianglelefteq = \leq)$

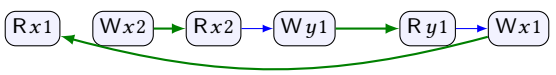
8 RFI EXAMPLES

Bad example:

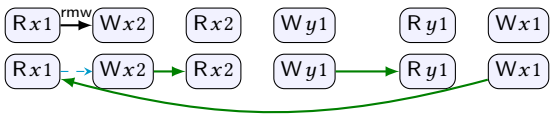
$$r := \mathsf{EXCHG}(x, 2)\,;\, s := x\,;\, y := s-1 \parallel r := y\,;\, x := r$$



$(\checkmark\text{ARM8})$

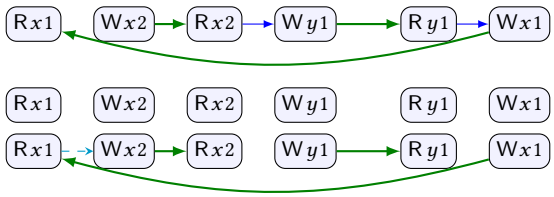


$(\trianglelefteq)$



$(\leq)$ $(\sqsubseteq)$

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



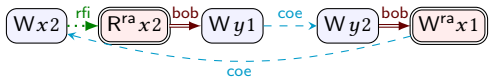
(⊴)

(≤)

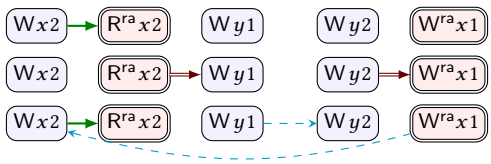
(⊑)

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

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



(✓ARM8)



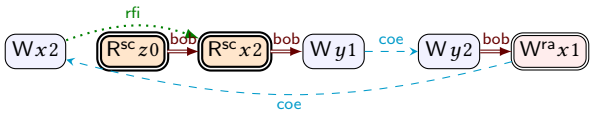
(⊴)

(≤)

(⊑)

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

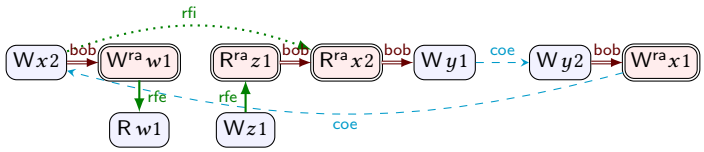
$x := 2\,;\ s := z^{\mathsf{sc}}\,;\ r := x^{\mathsf{sc}}\,;\ y := 1 \parallel y := 2\,;\ x^{\mathsf{ra}} := 1$ (RFI-ACQ-COE-COE)



(✓ARM8)

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

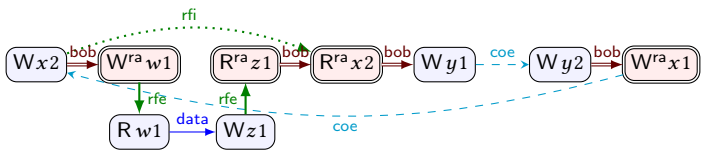
$x := 2\,;\ w^{\mathsf{ra}} := 1\,;\ s := z^{\mathsf{ra}}\,;\ r := x^{\mathsf{ra}}\,;\ y := 1$
$\parallel y := 2\,;\ x^{\mathsf{ra}} := 1 \parallel w := r\,;\ r := 1\,;$ (RFI-RA-COE-COE2)



(✓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:

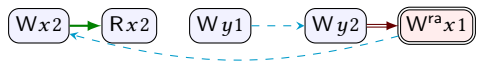
$x := 2\,;\ w^{\mathsf{ra}} := 1\,;\ s := z^{\mathsf{ra}}\,;\ r := x^{\mathsf{ra}}\,;\ y := 1$
$\parallel y := 2\,;\ x^{\mathsf{ra}} := 1 \parallel w := r\,;\ r := z\,;$ (RFI-RA-DATA-COE-COE)



(✗ARM8)

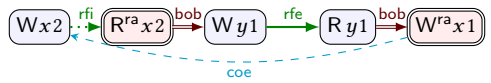
To allow this, weaken ra to rlx when read fulfilled by relaxed write of same thread (don't need to allow this when the write is part of an RMW).

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



RF variant `[rfi-rfe-coe]`:

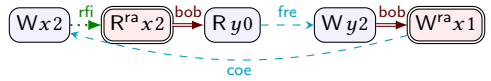
$$x := 2; r := x^{\mathsf{ra}}; y := 1 \parallel s := y; x^{\mathsf{ra}} := 1 \qquad \text{(RFI-RFE-COE)}$$



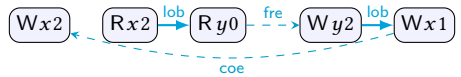
(✓ARM8)

TSO variant `[rfi-fre-coe]`:

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



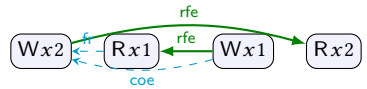
(✓ARM8)



(✓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]
- $d \overset{\text{po}}{\cdots\!\blacktriangleright} e$ when $c \overset{\text{rfe}}{\longrightarrow} d \overset{\text{po}}{\cdots\!\blacktriangleright} e$

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

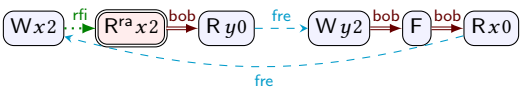
- [W]; po; [W]
- $d \overset{\text{po}}{\cdots\!\blacktriangleright} e$ when $c \overset{\text{rfe}}{\longrightarrow} d \overset{\text{po}}{\cdots\!\blacktriangleright} e$, from `(range(rfe) * _)`
- [R]; po; [W], from `(rfi^-1; lob)`

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]`:

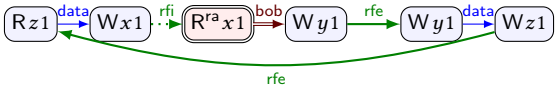
$$x := 2; r := x^{\mathsf{ra}}; s := y \parallel y := 2; \mathsf{F}; r := x \qquad \text{(RFI-FRE-FRE)}$$



(✓ARM8)

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

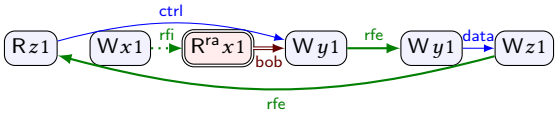
$x := z \,;\, r := x^{\mathsf{ra}} \,;\, y := 1 \parallel z := y$ (DATA-RFI-RFE-RFE)



(✗ARM8)

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

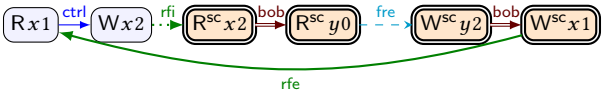
$\mathtt{if}(z)\{\}\,;\, x := 1 \,;\, r := x^{\mathsf{ra}} \,;\, y := 1 \parallel z := y$ (CTRL-RFI-RFE-RFE)



(✗ARM8)

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

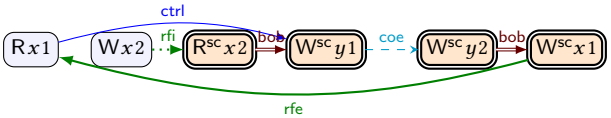
$\mathtt{if}(x)\{\}\,;\, x := 2 \,;\, r := x^{\mathsf{sc}} \,;\, s := y^{\mathsf{sc}} \parallel y^{\mathsf{sc}} := 2 \,;\, x^{\mathsf{sc}} := 1$ (CTRL-RFI-FRE-RFE)



(✓ARM8)

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

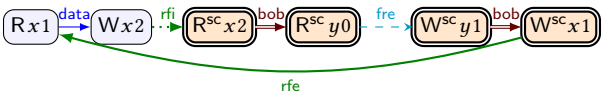
$\mathtt{if}(x)\{\}\,;\, x := 2 \,;\, r := x^{\mathsf{sc}} \,;\, y^{\mathsf{sc}} := 1 \parallel y^{\mathsf{sc}} := 2 \,;\, x^{\mathsf{sc}} := 1$ (CTRL-RFI-COE-RFE)



(✗ARM8)

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

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

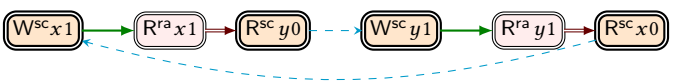


(✗ARM8)

9 SC EXAMPLES

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

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

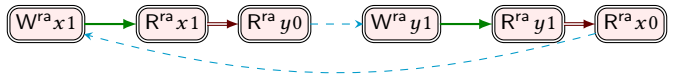


(✓POWER,RC11)

With the obvious semantics, we disallow this. 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.

Consider the variant with all ra access:

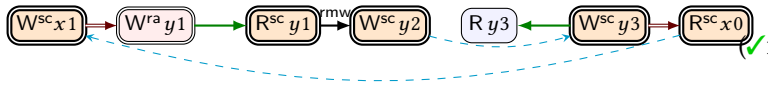
$$x^{\mathsf{ra}} := 1 \parallel y^{\mathsf{ra}} := 1 \parallel r := x^{\mathsf{ra}};\ s := y^{\mathsf{ra}} \parallel r := y^{\mathsf{ra}};\ s := x^{\mathsf{ra}} \qquad \text{(IRIW-ACQ-ACQ)}$$



(✓POWER,RC11)

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

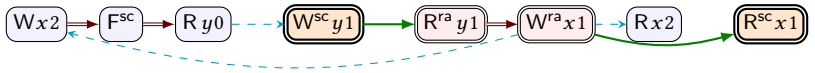
$$x^{\mathsf{sc}} := 1;\ y^{\mathsf{ra}} := 1 \parallel \mathsf{FADD}^{\mathsf{sc},\mathsf{sc}}(y, 1);\ s := y \parallel y^{\mathsf{sc}} := 3;\ s := x^{\mathsf{sc}} \qquad \text{(Z6.U)}$$



(✓POWER,RC11)

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

$$x := 2;\ \mathsf{F}^{\mathsf{sc}};\ r := y \parallel y^{\mathsf{sc}} := 1 \parallel r := y^{\mathsf{ra}};\ x^{\mathsf{ra}} := 1;\ s := x \parallel r := x^{\mathsf{sc}} \qquad \text{(RSYNC+RSC)}$$

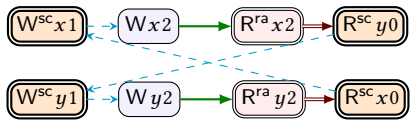


(✓RC11)

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

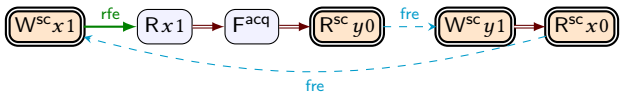
$$x^{\mathsf{sc}} := 1;\ x := 2 \parallel y^{\mathsf{sc}} := 1;\ y := 2 \parallel r := x^{\mathsf{ra}};\ s := y^{\mathsf{sc}} \parallel r := y^{\mathsf{ra}};\ s := x^{\mathsf{sc}} \qquad \text{(IRIW-SC-RLX-ACQ)}$$



(✓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].

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



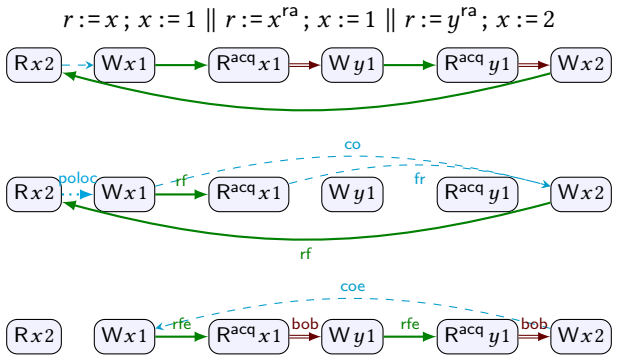
(✗ARM8)

10 RMWS

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.

11 EXAMPLE FROM JAM PAPER

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"



12 OLD MODEL

μ	::= wk	(Weak)	σ	::= cta	(Thread group)
	| rlx	(Relaxed)		| gpu	(Processor)
	| ra	(Release/Acquire)		| sys	(System)
	| sc	(Sequentially Consistent)			

Orders/Relations in model

- $\trianglelefteq$ is the old $\leq$ (without coherence stuff from F4 and P5B).
  This provides the NO-TAR axiom.
- $\leq$ is a the *happens-before* suborder, which only includes rf when they are morally strong.
  This serves as a cross-location transitive kernel for the per-location order.
- $\sqsubseteq$ is a per-location order that relates morally strong and poloc accesses
  This includes $\leq$ for morally strong accesses.
  This provides the SC-PER-LOC axiom.

Write $d \vartriangle e$ if they conflict (ie, read/write or write/write, same location).
Write $d \blacktriangle e$ if they conflict and are morally strong

Definition 12.1. A *pomset with preconditions* is a tuple $(E, \lambda, \leq, \trianglelefteq, \sqsubseteq)$ where

(M1) E is a set of *events*
(M2) $\lambda : E \to (\Phi \times \mathcal{A})$ is a *labeling* from which we derive functions
- $\kappa : E \to \Phi$ *(formulae)*
- $\lambda : E \to \mathcal{A}$ *(actions)*

(M3) $\leq \subseteq (E \times E)$, $\trianglelefteq \subseteq (E \times E)$, and $\sqsubseteq \subseteq (E \times E)$ are partial orders
(M4) $\bigwedge_e \kappa(e)$ is satisfiable *(consistency)*
(M5) if $d \trianglelefteq e$ then $\kappa(e)$ implies $\kappa(d)$ *(causal strengthening)*
(M6) if $d \leq e$ then $d \trianglelefteq e$
(M7) if $d \leq e$ and d conflicts with e then $d \sqsubseteq e$

Definition 12.2 (Strong fulfillment). We say $\lambda(d) = (\mathsf{W}xv)$ *fulfills* $\lambda(e) = (\mathsf{R}xv)$ if

(F3A) $d \lhd e$
(F3B) $d < e$ if d is morally strong with e
(F3C) $d \sqsubseteq e$ (if d is not morally strong with e)
(F4) $\forall \lambda(c) = (\mathsf{W}x..)$ either $c \sqsubseteq d$ or $e \sqsubseteq c$,

Definition 12.3 (Weak fulfillment). We say $\lambda(d) = (\mathsf{W}xv)$ *fulfills* $\lambda(e) = (\mathsf{R}xv)$ if

(F3A) $d \lhd e$

(F3B) $d < e$ if d is morally strong with e
(F3C) $e \not\sqsubseteq d$ (if d is not morally strong with e)
(F4) $\forall \lambda(c) = (\mathsf{W}x..)$ either $c \sqsubseteq\!\!\!\!\sim\, d$ or $e \sqsubseteq\!\!\!\!\sim\, c$, where

$$d \sqsubseteq\!\!\!\!\sim\, e \text{ when } \begin{cases} d \sqsubseteq e & \text{if } d \text{ is morally strong with } e \\ e \not\sqsubset d & \text{otherwise} \end{cases}$$

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

(F3) $d < e$
(F4) $\forall \lambda(c) = (\mathsf{W}x..)$ either $c \sqsubseteq d$ or $e \sqsubseteq c$

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

(F3) $e \not\sqsubseteq d$
(F4) $\nexists \lambda(c) = (\mathsf{W}x..)$ both $d \sqsubset c$ and $c \sqsubset e$

Note that the difference between strong and weak fulfillment is limited to $\sqsubseteq$. We sometimes write $\boxed{\mathsf{S}}$ for strong fulfillment and $\boxed{\mathsf{W}}$ for weak fulfillment.

Prefixing is as in OOPSLA, using $\leq$ for order everywhere except P5B, which has $\sqsubseteq$.

Definition 12.4. Let $P' \in (\phi \mid a) \Rightarrow \mathcal{P}$ when $(\exists P \in \mathcal{P})\ (\forall e \in E)$

(P1) $E' = E \cup \{d\}$
(P2) $\leq' \supseteq \leq$, $\trianglelefteq' \supseteq \trianglelefteq$, and $\sqsubseteq' \supseteq \sqsubseteq$
(P3A) $\lambda'(e) = \lambda(e)$
(P3B) $\lambda'(d) = a$
(P4A) $\kappa'(d)$ implies $\phi \wedge (d \notin E \vee \kappa(d))$
(P4B) if $d \neq (\mathsf{R}..)$ then $e = d$ or $\kappa'(e)$ implies $\kappa(e)$
(P4C) if $d = (\mathsf{R}vx)$ then $e = d$ or $\kappa'(e)$ implies $\kappa(e)[v/x]$
(P5A) if $d = (\mathsf{R}..)$, $e = (\mathsf{W}..)$ then $e = d$ or $\kappa'(e)$ implies $\kappa(e)$ or $d \leq' e$
(P5B) if d conflicts with e then $d \sqsubseteq' e$
(P5C) if d is an acquire or e is a release then $d \leq' e$
(P5D) if d is an SC write and e is an SC read then $d \leq' e$
(P5E) if d reads, and e is an acquiring fence, then $d \leq' e$
(P5F) if d is a releasing fence, and e writes, then $d \leq' e$

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

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

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.

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