A Unified Memory Model for Heterogenous Systems

ANONYMOUS AUTHOR(S)

ACM Reference Format:
Anonymous Author(s). 2022. A Unified Memory Model for Heterogenous Systems. *Proc. ACM Program. Lang.* 0, POPL, Article 0 (January 2022), 21 pages.

1 MODEL

1.1 Preliminaries

The syntax is built from

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

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

- values and registers are disjoint,
- values include at least the constants 0 and 1,
- expressions include at least registers and values,
- references do not appear in expressions: $M[N/x] = M$,
- thread ids include the *top-level* id **0**.

We model the following language (defaults underlined).

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

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

Access modes, μ, are weak (wk), relaxed (rlx), release (rel), acquire (acq), release-acquire (ra), and sequentially consistent (sc). In examples, we systematically drop the default mode rlx. 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. Let expressions ($r := M$) only affect thread-local state and thus do not have a mode.

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 thread id α, then S_1 runs with id γ and S_2 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.

Scopes, σ, are thread group (cta), processor (gpu) and system (sys). In examples, we systematically drop the default scope sys. Let $(=_{\mathsf{sys}}) = (\mathcal{T} \times \mathcal{T})$. We assume two equivalences: $(=_{\mathsf{gpu}}) \subseteq (=_{\mathsf{sys}})$ partitions threads by *processor*, and $(=_{\mathsf{cta}}) \subseteq (=_{\mathsf{gpu}})$ refines the processor partitioning into *thread*

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

groups. In examples, we mostly elide thread ids and ignore the gpu scope. We write $(S_1 \ {}_\sigma\!\Vdash\! S_2)$ to indicate that the statements run in threads related by $=_\sigma$, but not by any $=_\rho$ for $\rho \sqsubset \sigma$. In the following examples, let α be id of the rightmost thread. Then $(S_1 \ {}_{\mathsf{cta}}\!\Vdash\! S_2)$ is shorthand for $(\exists\gamma =_{\mathsf{cta}} \alpha)\ (S_1 \ {}_\gamma\!\Vdash\! S_2)$. When using this convention, $\Vdash$ associates to the left; thus, $(S_1 \ {}_{\mathsf{cta}}\!\Vdash\! S_2 \ {}_{\mathsf{sys}}\!\Vdash\! S_3)$ is read as $((S_1 \ {}_{\mathsf{cta}}\!\Vdash\! S_2) \ {}_{\mathsf{sys}}\!\Vdash\! S_3)$, which is $(\exists\gamma =_{\mathsf{cta}} \delta \neq_{\mathsf{cta}} \alpha)\ (S_1 \ {}_\gamma\!\Vdash\! S_2 \ {}_\delta\!\Vdash\! S_3)$. When there is no continuation, we further simplify $S_1 \ {}_\sigma\!\Vdash\! S_2$ to $S_1 \parallel_\sigma S_2$ and $S_1 \parallel_{\mathsf{sys}} S_2$ to $S_1 \parallel S_2$; thus, $(S_1 \parallel_{\mathsf{cta}} S_2 \parallel S_3)$ should be read as $(\exists\gamma =_{\mathsf{cta}} \delta \neq_{\mathsf{cta}} \alpha)\ (S_1 \ {}_\gamma\!\Vdash\! S_2 \ {}_\delta\!\Vdash\! S_3)$.

We use common syntax sugar, such as *extended expressions*, $\mathbb{M}$, which include memory locations. For example, if $\mathbb{M}$ includes a single occurrence of x, then $y := \mathbb{M}\,;\ S$ is shorthand for $r := x\,;\ y := \mathbb{M}[r/x]\,;\ S$. Each occurrence of x in an extended expression corresponds to an separate read. We also write `if(`M`){`S`}` as shorthand for `if(`M`){`S`}else{skip}`.

The semantics is built from the following.

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

We require that

- registers include $\mathcal{S}_\mathcal{E} = \{s_e \mid e \in \mathcal{E}\}$ which do not appear in commands: $S[N/s_e] = S$,
- formulae include tt, ff, Q_x, and the equalities $(M{=}N)$ and $(x{=}M)$,
- formulae are closed under $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and substitutions $[M/r]$, $[M/x]$, $[\phi/\mathsf{Q}_x]$,
- there is a relation $\vDash$ between formulae, capturing entailment,
- $\vDash$ has the expected semantics for $=$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and substitution.
- there is a subset of $\mathcal{A}$, distinguishing *read* actions,
- there are several relations over $\mathcal{A} \times \mathcal{A}$, detailed in the next subsection: *matches*, *blocks*, *overlaps*, *strongly-matches*, *strongly-overlaps*, *strongly-fences*, *sync-delays*, *perloc-delays*,
- matches $\subseteq$ blocks $\subseteq$ overlaps $\supseteq$ strongly-overlaps
- strongly-matches $\subseteq$ strongly-overlaps $\cup$ strongly-fences

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 $\phi \vDash \mathsf{ff}$.

1.2 Actions

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

$$a, b \ ::= \ \alpha\mathsf{W}^\mu_\sigma x v \ \mid \ \alpha\mathsf{R}^\mu_\sigma x v \ \mid \ \alpha\mathsf{F}^\mu_\sigma$$

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* *perloc-delays* *b* if $(a,b) \in \{(\mathsf{W}x,\mathsf{W}x),\ (\mathsf{R}x,\mathsf{W}x),\ (\mathsf{W}x,\mathsf{R}x)\} \cup \{(\mathsf{A}^{\mathsf{sc}},\mathsf{A}^{\mathsf{sc}})\}$.
We say *a* *sync-delays* *b* if $(a,b) \in \{(a,\mathsf{W}^{\sqsupseteq\mathsf{rel}}),\ (a,\mathsf{F}^{\sqsupseteq\mathsf{rel}}),\ (\mathsf{R},\mathsf{F}^{\sqsupseteq\mathsf{acq}}),\ (\mathsf{R}^{\sqsupseteq\mathsf{acq}},b),\ (\mathsf{F}^{\sqsupseteq\mathsf{acq}},b),\ (\mathsf{F}^{\sqsupseteq\mathsf{rel}},\mathsf{W}),\ (\mathsf{W}^{\sqsupseteq\mathsf{rel}}x,\mathsf{W}x)\}$.[1]
Actions (R) are *read* actions.

Definition 1.1. We say $(\alpha_1\mathsf{A}^{\mu_1}_{\sigma_1}x)$ *strongly-overlaps* $(\alpha_2\mathsf{A}^{\mu_2}_{\sigma_2}x)$ when either
(1) $\alpha_1 = \alpha_2$, or
(2a) $\mu_1,\mu_2 \sqsupseteq \mathsf{rlx}$,
(2b) if $\sigma_1 = \mathsf{cta}$ or $\sigma_2 = \mathsf{cta}$ then $\alpha_1 =_{\mathsf{cta}} \alpha_2$,
(2c) if $\sigma_1 = \mathsf{gpu}$ or $\sigma_2 = \mathsf{gpu}$ then $\alpha_1 =_{\mathsf{gpu}} \alpha_2$.
We say $(\alpha_1\mathsf{F}^{\mu_1}_{\sigma_1})$ *strongly-fences* $(\alpha_2\mathsf{F}^{\mu_2}_{\sigma_2})$ when $\mu_1 = \mu_2 = \mathsf{sc}$ and either (1) or (2) apply, from the definition of strongly-overlaps.
We say *a* *strongly-matches* *b* when *a* is a release, *b* is an acquire, and either *a* strongly-overlaps *b* or *a* strongly-fences *b*. [Todo: This looks wrong.]

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

The semantics here includes all the features of [Jeffrey et al. 2021, §9]: Register Recycling, Register Consistency, Fences, and RMWs. We account for Address Calculation and If-Closure in §**??**. We have proposals to account for Dead Store Elimination, Store Forwarding, and Monotonicity in §11.

Definition 1.2. Let $E \subseteq \mathcal{E}$ and $\lambda : E \to \mathcal{A}$. Define $\theta_\lambda = \bigwedge_{\{(e,v)\in(E\times\mathcal{V})\mid\lambda(e)=(\mathsf{R}\,v)\}}(s_e{=}v)$.
We say that ϕ is λ-*inconsistent* if $\phi \wedge \theta_\lambda$ is unsatisfiable.

Definition 1.3. A λ-*predicate transformer* is a function $\tau : \Phi \to \Phi$ such that
(x1) $\tau(\psi_1 \wedge \psi_2) \equiv \tau(\psi_1) \wedge \tau(\psi_2)$,
(x2) $\tau(\psi_1 \vee \psi_2) \equiv \tau(\psi_1) \vee \tau(\psi_2)$,
(x3) if $\phi \vDash \psi$, then $\tau(\phi) \vDash \tau(\psi)$,
(x4) if ψ is λ-inconsistent then $\tau(\psi)$ is λ-inconsistent.

Definition 1.4. A *family of* λ-*predicate transformers* consists of a λ-predicate transformer τ^D for each $D \subseteq \mathcal{E}$, such that if $C \cap E \subseteq D$ then $\tau^C(\psi) \vDash \tau^D(\psi)$.

Definition 1.5. A *pomset with predicate transformers* is a tuple $(E, \lambda, \kappa, \tau, \checkmark, \lhd, <, \sqsubset, \mathsf{rmw})$ where
(M1) $E \subseteq \mathcal{E}$ is a set of *events*,
(M2) $\lambda : E \to \mathcal{A}$ defines a *label* for each event,
(M3) $\kappa : E \to \Phi$ defines a *precondition* for each event, such that
(M3a) $e \notin E$ implies $\kappa(e) = \mathsf{ff}$,
(M4) $\tau : 2^{\mathcal{E}} \to \Phi \to \Phi$ is a *family of* λ-*predicate transformers*,
(M5) $\checkmark : \Phi$ is a *termination condition*, such that
(M5a) $\checkmark \vDash \tau^E(\mathsf{tt})$,
(M6) $\lhd : (E \times E)$ is a strict partial order capturing *dependency*,
(M7) $< : (E \times E)$ is a strict partial order capturing *synchronization*,
(M8) $\sqsubset : (E \times E)$ is a strict partial order capturing *per-location order*, such that
(M8a) if $\lambda(d)$ overlaps $\lambda(e)$ then $d < e$ implies $d \sqsubset e$,
(M9) $\mathsf{rmw} : E \to E$ is a partial function capturing read-modify-write *atomicity*, such that
(M9a) if $d \xrightarrow{\mathsf{rmw}} e$ then $\lambda(e)$ blocks $\lambda(d)$, $d < e$, and $d \sqsubset e$,
(M9b) if if $d \xrightarrow{\mathsf{rmw}} e$ and $\lambda(c)$ overlaps $\lambda(d)$ then
(i) $c \lhd e$ implies $c \trianglelefteq d$, $c < e$ implies $c \leq d$, $c \sqsubset e$ implies $c \sqsubseteq d$,
(ii) $d \lhd c$ implies $e \trianglelefteq c$, $d < c$ implies $e \leq c$, $d \sqsubset c$ implies $e \sqsubseteq c$.

[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.

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

(c2a) if $d \xrightarrow{\mathsf{rf}} e$ then $\lambda(d)$ matches $\lambda(e)$,
(c6) if $d \xrightarrow{\mathsf{rf}} e$ then $d \lhd e$,
(c7a) if $d' \leq d \xrightarrow{\mathsf{rf}} e \leq e'$ and $\lambda(d')$ strongly-matches $\lambda(e')$ then $d' < e'$,
(c7b) if $\lambda(d)$ strongly-fences $\lambda(e)$ then either $d \leq e$ or $e \leq d$, [Todo: Is this right?]
(c8a) if $d \xrightarrow{\mathsf{rf}} e$ then $d \sqsubset e$,
(c8b) if $d \xrightarrow{\mathsf{rf}} e$ and $\lambda(c)$ blocks $\lambda(e)$ then either $c \mathrel{\tilde{\sqsubseteq}} d$ or $e \mathrel{\tilde{\sqsubseteq}} c$,
where $d' \mathrel{\tilde{\sqsubseteq}} e'$ when $e' \sqsubseteq d'$ implies $d' = e'$ and $\lambda(d')$ strongly-overlaps $\lambda(e')$ implies $d' \sqsubseteq e'$.

A candidate pomset with rf is *complete* if

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

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

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

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

We drop quantifiers when clear from context, such as $(\forall e \in E)(\forall x \in X)$. We write $d \leq e$ to mean that either $d < e$ or $d = e$, and similarly for $\trianglelefteq$ and $\sqsubseteq$. We sometimes use projection functions—for example, if $\lambda(e) = \alpha \mathsf{W}^{\mu}_{\sigma} x v$ then $\lambda_{\mathsf{thrd}}(e) = \alpha$, $\lambda_{\mathsf{mode}}(e) = \mu$, $\lambda_{\mathsf{scope}}(e) = \sigma$, $\lambda_{\mathsf{loc}}(e) = x$, $\lambda_{\mathsf{val}}(e) = v$.

The semantic functions are defined in Fig. 1 and Fig. 2.

$$[\![r := M]\!] = \mathit{ASSIGN}(r, M) \qquad [\![\mathsf{F}^{\mu}_{\sigma}]\!] = \mathit{FENCE}(\mu, \sigma)_{\alpha} \qquad [\![S_1 \Vdash S_2]\!] = \mathit{PAR}([\![S_1]\!], [\![S_2]\!])$$
$$[\![x^{\mu} := M]\!] = \mathit{WRITE}(x, M, \mu, \sigma)_{\alpha} \qquad [\![\mathtt{skip}]\!] = \mathit{SKIP} \qquad [\![S_1\,;\, S_2]\!] = \mathit{SEQ}([\![S_1]\!], [\![S_2]\!])$$
$$[\![r := x^{\mu}]\!] = \mathit{READ}(r, x, \mu, \sigma)_{\alpha} \qquad [\![\mathtt{if}(M)\{S_1\}\,\mathtt{else}\,\{S_2\}]\!] = \mathit{IF}(M{\neq}0, [\![S_1]\!], [\![S_2]\!])$$

$$[\![r := \mathsf{CAS}^{\mu,\nu}_{\sigma}(x, M, N)]\!]_{\alpha} = \mathit{CAS}(r, x, M, N, \mu, \nu, \sigma)_{\alpha}$$
$$[\![r := \mathsf{FADD}^{\mu,\nu}_{\sigma}(x, M)]\!]_{\alpha} = \mathit{FADD}(r, x, M, \mu, \nu, \sigma)_{\alpha}$$
$$[\![r := \mathsf{EXCHG}^{\mu,\nu}_{\sigma}(x, M)]\!]_{\alpha} = \mathit{EXCHG}(r, x, M, \mu, \nu, \sigma)_{\alpha}$$

In diagrams, we use different shapes and colors for arrows and events. These are included only to help the reader understand why order is included. We adopt the following conventions:

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

[2] If all accesses are morally strong with each other, weak fulfillment degenerates to $\forall \lambda(c) = (\mathsf{W}x)$ either $c \sqsubseteq d$ or $e \sqsubseteq c$. If no accesses are morally strong with each other, weak fulfillment degenerates to $\nexists \lambda(c) = (\mathsf{W}x)$ both $d \sqsubset c$ and $c \sqsubset e$. Note that the difference between strong and weak fulfillment is limited to $\sqsubseteq$. We sometimes write $\sqsubseteq_{\mathsf{S}}$ for strong fulfillment and $\sqsubseteq_{\mathsf{W}}$ for weak fulfillment.

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

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

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

(P1) $E = (E_1 \uplus E_2)$,
(P2) $\lambda = (\lambda_1 \cup \lambda_2)$,
(P3) $\kappa(e) \equiv \kappa_1(e) \vee \kappa_2(e)$,
(P4) $\tau^D(\psi) \equiv \tau_2^D(\psi)$,
(P5) $\checkmark \equiv \checkmark_1 \wedge \checkmark_2$,
(P6) $\lhd \supseteq (\lhd_1 \cup \lhd_2)$,
(P7) $< \supseteq (<_1 \cup <_2)$,
(P8) $\sqsubset \supseteq (\sqsubset_1 \cup \sqsubset_2)$,
(P9) $\mathsf{rmw} = (\mathsf{rmw}_1 \cup \mathsf{rmw}_2)$.

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

(I1) $E = (E_1 \cup E_2)$,
(I2) (I6) (I7) (I8) (I9) as for *PAR*,
(I3) $\kappa(e) \equiv (\phi \wedge \kappa_1(e)) \vee (\neg\phi \wedge \kappa_2(e))$,
(I4) $\tau^D(\psi) \equiv (\phi \wedge \tau_1^D(\psi)) \vee (\neg\phi \wedge \tau_2^D(\psi))$,
(I5) $\checkmark \equiv (\phi \wedge \checkmark_1) \vee (\neg\phi \wedge \checkmark_2)$.

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

(S1) (S2) (S6) (S7) (S8) (S9) as for *IF*,
(S3) $\kappa(e) \equiv \kappa_1(e) \vee \kappa_2'(e)$,
(S4) $\tau^D(\psi) \equiv \tau_1^D(\tau_2^D(\psi))$,
(S5) $\checkmark \equiv \checkmark_1 \wedge \tau_1(\checkmark_2)$,
(S7a) if $\lambda_1(d)$ sync-delays $\lambda_2(e)$ then $d \leq e$,
(S8a) if $\lambda_1(d)$ perloc-delays $\lambda_2(e)$ then $d \sqsubseteq e$,

where

$$\kappa_2'(e) = \begin{cases} \tau_1^{E_1}(\kappa_2(e)) & \text{if } \lambda(e) \text{ is a read} \\ \tau_1^{C}(\kappa_2(e)) & \text{otherwise, where } C = \{c \mid c < e\} \end{cases}$$

Let $\mathsf{K}(D) = \bigvee_{d \in D} \kappa(d)$. Note that $\mathsf{K}(\emptyset) = \mathsf{ff}$.

Let $\Phi_{\mathcal{S}_\mathcal{E}} = \{\phi \in \Phi \mid \forall e \in \mathcal{E}.\ \forall v \in \mathcal{V}.\ \phi \equiv \phi[v/s_e]\}$.

If $P \in \mathit{FENCE}(\mu, \sigma)_\alpha$ then $(\exists \phi : E \to \Phi_{\mathcal{S}_\mathcal{E}})$

(F1) if $\phi_d \wedge \phi_e$ is satisfiable then $d = e$,
(F2) $\lambda(e) = \alpha \mathsf{F}^\mu_\sigma$,
(F3) $\kappa(e) \equiv \phi_e$,
(F4) $\tau^D(\psi) \equiv \psi$,
(F5) $\checkmark \equiv \mathsf{K}(E)$.

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

(W1) if $\phi_d \wedge \phi_e$ is satisfiable then $d = e$,
(W2) $\lambda(e) = \alpha \mathsf{W}^\mu_\sigma x v_e$,
(W3) $\kappa(e) \equiv \phi_e \wedge M{=}v_e$,
(W4) $\tau^D(\psi) \equiv \psi[M/x][\mathsf{K}(E)/\mathsf{Q}_x]$,
(W5) $\checkmark \equiv \mathsf{K}(E)$.

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

(R1) if $\phi_d \wedge \phi_e$ is satisfiable then $d = e$,
(R2) $\lambda(e) = \alpha \mathsf{R}^\mu_\sigma x v_e$
(R3) $\kappa(e) \equiv \phi_e \wedge \mathsf{Q}_x$,
(R4) $\tau^D(\psi) \equiv \bigwedge_{e \in E \cap D} \phi_e \Rightarrow (\kappa(e) \Rightarrow v_e{=}s_e) \Rightarrow \psi[s_e/r]$
$\wedge \bigwedge_{e \in E \setminus D} \phi_e \Rightarrow (\kappa(e) \Rightarrow (v_e{=}s_e \vee x{=}s_e)) \Rightarrow \psi[s_e/r]$
$\wedge (\bigwedge_{e \in E} \neg\phi_e) \Rightarrow (\forall s)\ \psi[s/r]$
(R5a) if $\mu \sqsubseteq \mathsf{rlx}$ then $\checkmark \equiv \mathsf{tt}$,
(R5b) if $\mu \sqsupseteq \mathsf{acq}$ then $\checkmark \equiv \mathsf{K}(E)$.

If $P \in \mathit{READ}'(r, x, \mu, \sigma)_\alpha$ then $(\exists v : E \to \mathcal{V})\ (\exists \phi : E \to \Phi_{\mathcal{S}_\mathcal{E}})$ as for *READ* except

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

Fig. 1. Semantic Functions

If $P \in CAS(r, x, M, N, \mu, \nu, \sigma)_\alpha$ then $(\exists P \in SEQ(READ'(r, x, \mu, \sigma)_\alpha, IF(r{=}M, WRITE(x, N, \nu, \sigma)_\alpha, SKIP)))$
(υ9) if $\lambda(e)$ is a write then there is a read $\lambda(d)$ such that $\kappa(e) \vDash \kappa(d)$ and $d \xrightarrow{\text{rmw}} e$.

If $P \in FADD(r, x, M, \mu, \nu, \sigma)_\alpha$ then $(\exists P \in SEQ(READ'(r, x, \mu, \sigma)_\alpha, WRITE(x, r{+}M, \nu, \sigma)_\alpha))$
(υ9) if $\lambda(e)$ is a write then there is a read $\lambda(d)$ such that $\kappa(e) \vDash \kappa(d)$ and $d \xrightarrow{\text{rmw}} e$.

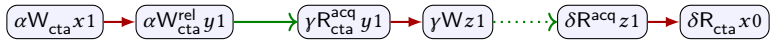
If $P \in EXCHG(r, x, M, \mu, \nu, \sigma)_\alpha$ then $(\exists P \in SEQ(READ'(r, x, \mu, \sigma)_\alpha, WRITE(x, M, \nu, \sigma)_\alpha))$
(υ9) if $\lambda(e)$ is a write then there is a read $\lambda(d)$ such that $\kappa(e) \vDash \kappa(d)$ and $d \xrightarrow{\text{rmw}} e$.

Fig. 2. RMW Semantic Functions

2 SYNC EXAMPLES

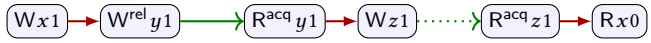
The first of these is seen in hardware. All are allowed by PTX. Showing rf that is not included in the order using a dotted arrow. $\alpha =_{\text{cta}} \gamma \neq_{\text{cta}} \delta$

$$x_{\text{cta}} := 1;\ y^{\text{rel}}_{\text{cta}} := 1 \parallel_{\text{cta}} r := y^{\text{acq}}_{\text{cta}};\ z := r \parallel r := z^{\text{acq}};\ s := x_{\text{cta}}$$



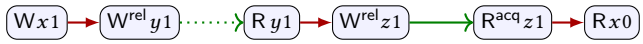
(<)

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



(<)

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



(<)

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

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

(<)

Previous def of candidate requires:
(c7a) if $d \xrightarrow{\text{rf}} e$ and $\lambda(d)$ strongly-matches $\lambda(e)$ then $d < e$.
This is not good enough for fences. A possible fix is the following closure condition:
(c7a′) if $d' \leq d \xrightarrow{\text{rf}} e \leq e'$ and $\lambda(d')$ strongly-matches $\lambda(e')$ then $d' < e'$.
With that we have the following, using $\twoheadrightarrow$ for edges induced by closure when $d' \neq d$ or $e' \neq e$:

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

(<)

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

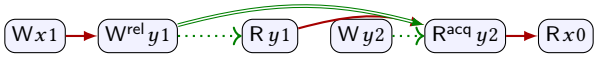
- One possibility is to restrict to preceding and following things in the same thread:
(c7a″) if $d' \leq_{\text{po}} d \xrightarrow{\text{rf}} e \leq_{\text{po}} e'$ and $\lambda(d')$ strongly-matches $\lambda(e')$ then $d' < e'$.
where $\leq_{\text{po}}$ is the obvious restriction of $\leq$ to actions on the same thread.
- With either (c7a′) or (c7a″) is it too strong to require that $<$ be transitive? In particular:
  - if we restrict to $\leq_{\text{po}}$, 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:

(M9a′) if $d \xrightarrow{\text{rmw}} e$ then $d \sqsubset e$,
(C7a‴) if $d' \le d$ $(\xrightarrow{\text{rf}}; (\xrightarrow{\text{rmw}}; \xrightarrow{\text{rf}})^*)$ $e \le e'$ and $\lambda(d')$ strongly-matches $\lambda(e')$ then $d' < e'$.

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).

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



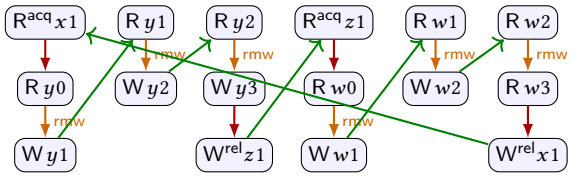
(<)

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

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

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

$$r := x^{\text{acq}};\ \texttt{INC}(y) \ \|\ \texttt{INC}(y) \ \|\ \texttt{INC}(y);\ z^{\text{rel}} := 1 \ \|\ s := z^{\text{acq}};\ \texttt{INC}(w) \ \|\ \texttt{INC}(w) \ \|\ \texttt{INC}(w);\ x^{\text{rel}} := 1$$



About release sequences, fences, and RMWs:

- Jamboard 10 allowed by IMM
- Examples like

$$\mathsf{W}x \xrightarrow{\text{po}} \mathsf{F}^{\text{rel}} \xrightarrow{\text{po}} \xrightarrow{\text{rmw}} \xrightarrow{\text{rf}} \xrightarrow{\text{rmw}} \xrightarrow{\text{rf}} \xrightarrow{\text{rmw}} \xrightarrow{\text{po}} \mathsf{F}^{\text{acq}}$$

About SC fences:

- total order on SC fences must obey

$$\mathsf{F}^{\text{sc}} \xrightarrow{\text{po}} \mathsf{W}x \xrightarrow{\text{eco}} \mathsf{W}x \xrightarrow{\text{po}} \mathsf{F}^{\text{sc}}$$

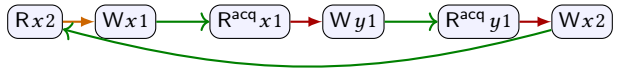
  but it's not in happens before.

3 EXAMPLE FROM JAM PAPER

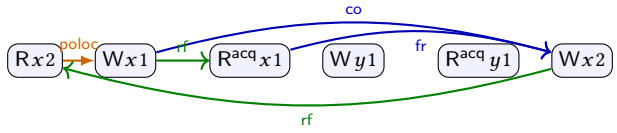
From [Bender and Palsberg 2019, §3.3]. With partial coherence/weak fulfillment you need to be careful that RMWs are totally ordered (if that's a property you want). May not come for free.

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

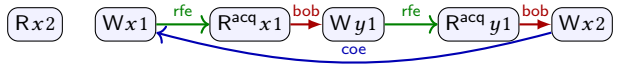
$r := x \,;\, x := 1 \parallel r := x^{\mathsf{acq}} \,;\, y := 1 \parallel r := y^{\mathsf{acq}} \,;\, x := 2$ (TOTAL-CO)



(✗Arm8)



(✗Arm8)



(✗Arm8)

4 IRIW

Check the examples in [Bender and Palsberg 2019, Fig 17]. They should all be allowed. That semantics is based on two partial orders, I think: co and vo (although vo is perhaps not transitive).

Also check example from https://bugs.openjdk.java.net/browse/JDK-8262877, which should be forbidden.

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

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

(<)

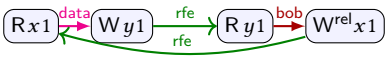
5 RELATING IMM AND PTX

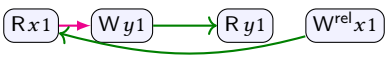
Release and acquire are symmetric in PTX, but nor IMM.

IMM includes fences, dependencies, rf in a single relation

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:

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





(◁)

(<)

(⊏)

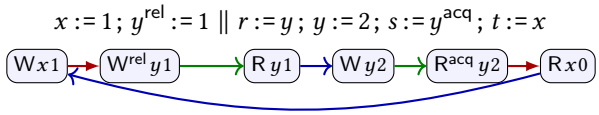
IMM forbids it, but PTX allows it. The point is that IMM mixes dependencies and release/acquire-induced po-order in its NoOOTA axiom, whereas PTX doesn't — release/acquire are only used to have coherence.

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.

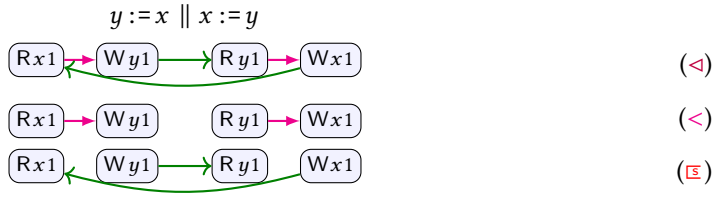
The previous example in the section shows that IMM's acquires are stronger than PTX for this pattern. The next example shows that acquiring reads in PTX are stronger than in IMM for a different pattern. Thus the acquires in PTX and IMM are incomparable.

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).



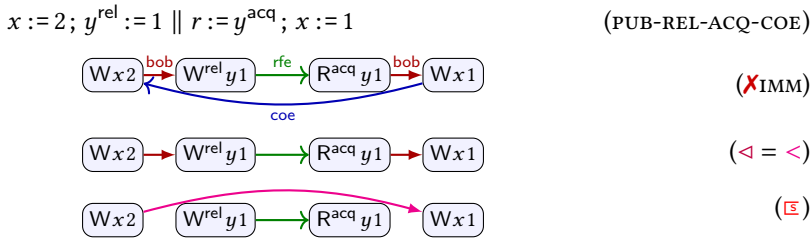
6 THIN AIR

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

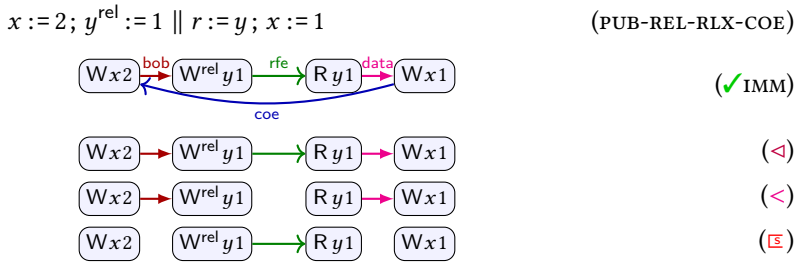


7 IMM EXAMPLES

Disallowed by IMM:

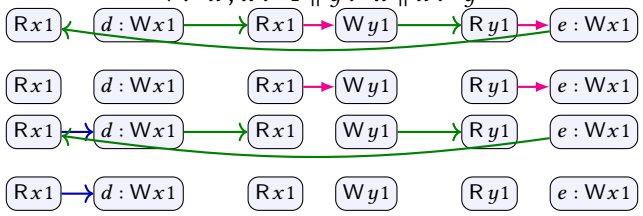


Allowed by IMM, but not by Power/ARMv7/ARMv8/TSO:



Example from talk:

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



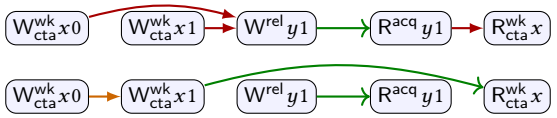
8 PTX EXAMPLES

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

In examples, all threads in different ctas.

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

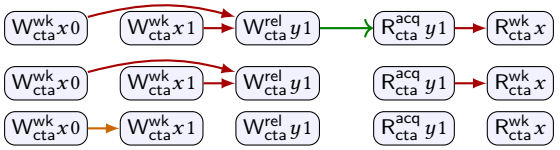
$x^{\mathsf{wk}}_{\mathsf{cta}} := 0 \,;\, x^{\mathsf{wk}}_{\mathsf{cta}} := 1 \,;\, y^{\mathsf{rel}} := 1 \parallel r := y^{\mathsf{acq}} \,;\, s := x^{\mathsf{wk}}_{\mathsf{cta}}$ (PUB1$_{\mathrm{SYS}}$)



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

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

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



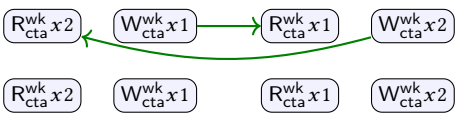
We do not have $(\mathsf{W}^{\mathsf{rel}}y1) < (\mathsf{R}^{\mathsf{acq}}y1)$ since c7a only requires order for things that are morally strong.

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

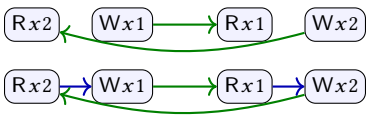
$x := 0 \,;\, x := 1 \parallel y := x \parallel \texttt{if}(y)\{r := x\}$

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

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



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



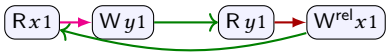
About Release-Acquire semantics. Anton confirms that the following example is allowed in C11, but disallowed in the IMM. It is apparently allowed in C11 with the intention to allow releasing writes to be downgraded to relaxed in the case that only fulfill relaxed reads.

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

$(\lhd = <)$

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

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

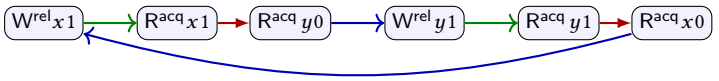


$(\lhd = <)$

9 SC EXAMPLES

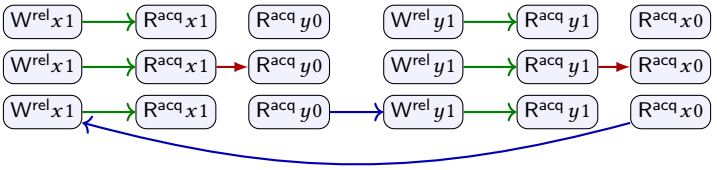
Example 9.1. Consider IRIW with all ra access:

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



(✓POWER,C11)

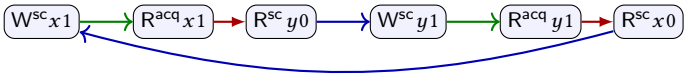
We allow this execution:



$(\lhd)$ $(<)$ $(\sqsubset)$

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

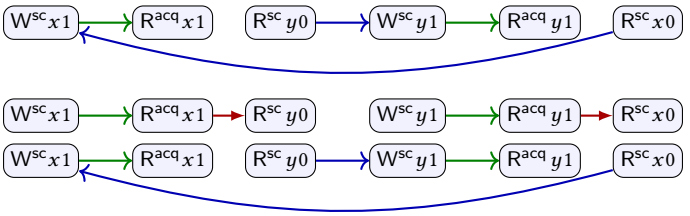
$$x^{\mathsf{sc}} := 1 \parallel r := x^{\mathsf{acq}} \,;\, s := y^{\mathsf{sc}} \parallel y^{\mathsf{sc}} := 1 \parallel r := y^{\mathsf{acq}} \,;\, s := x^{\mathsf{sc}} \qquad \text{(IRIW-ACQ-SC1)}$$



(✓POWER,✗C11)

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

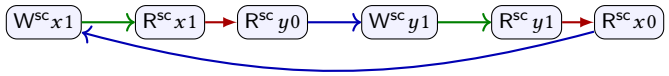
- add sc bullet to def of $\sqsubset$ in c8b,
- add SC access to *sync-delays*.



$(\lhd)$ $(<)$ $(\sqsubset)$

This correctly forbids the all sc version:

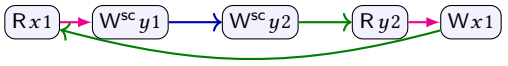
$$x^{sc} := 1 \parallel r := x^{sc}; s := y^{sc} \parallel y^{sc} := 1 \parallel r := y^{sc}; s := x^{sc} \quad \text{(IRIW-sc-sc)}$$



(◁)

Example 9.2. Thin air with an SC antidependency:

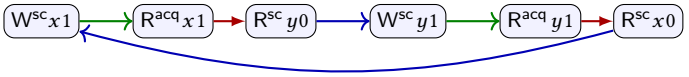
$$y^{sc} := x \parallel y^{sc} := 2 \parallel x := y - 1$$



(◁)

IRIW-acq-sc2 is allowed by trailing-sync compilation to power [Lahav et al. 2017, §1].

$$x^{sc} := 1 \parallel r := x^{acq}; s := y^{sc} \parallel y^{sc} := 1 \parallel r := y^{acq}; s := x^{sc} \quad \text{(IRIW-acq-sc2)}$$

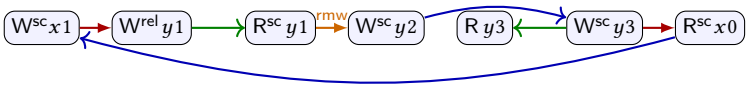


(✓POWER,RC11)

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].

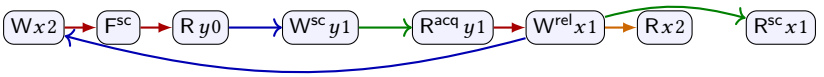
$$x^{sc} := 1; y^{rel} := 1 \parallel \mathsf{FADD}^{sc,sc}(y, 1); s := y \parallel y^{sc} := 3; s := x^{sc} \quad \text{(Z6.U)}$$



(✓POWER,RC11)

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

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

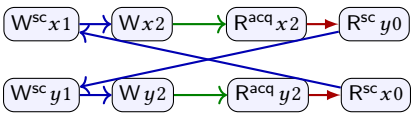


(✓RC11)

Fulfillment of $(\mathsf{R}x2)$ requires that either $(\mathsf{W}^{rel}x1) \longrightarrow (\mathsf{W}x2)$ or $(\mathsf{R}x2) \longrightarrow (\mathsf{W}^{rel}x1)$. It's interesting that in the pomset, $(\mathsf{R}^{sc}x1)$ 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:

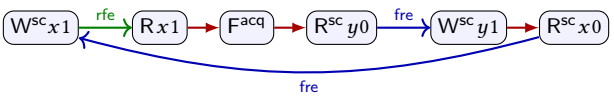
$$x^{sc} := 1; x := 2 \parallel y^{sc} := 1; y := 2 \parallel r := x^{acq}; s := y^{sc} \parallel r := y^{acq}; s := x^{sc} \quad \text{(IRIW-sc-rlx-acq)}$$



(✓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].

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

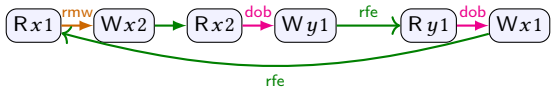


(✗Arm8)

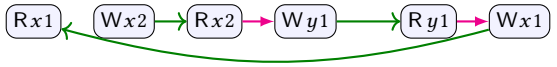
10 RFI EXAMPLES

Bad example:

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

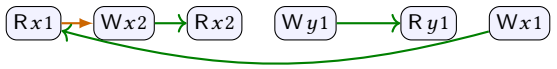


(✓Arm8)



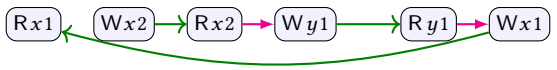
(◁)

(<)

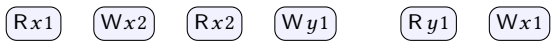


(⊏)

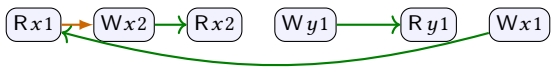
$r := x\,;\ x := 2\,;\ s := x\,;\ y := s-1 \ \|\ r := y\,;\ x := r$



(◁)



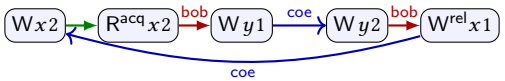
(<)



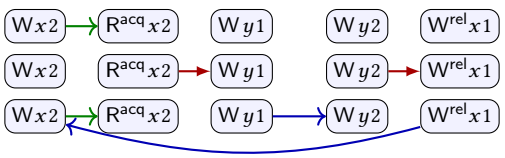
(⊏)

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

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



(✓Arm8)



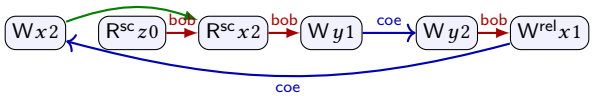
(◁)

(<)

(⊏)

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

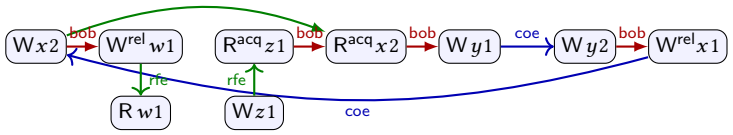
$x := 2;\ s := z^{\mathsf{sc}}\,;\ r := x^{\mathsf{sc}}\,;\ y := 1 \ \|\ y := 2;\ x^{\mathsf{rel}} := 1$ (RFI-ACQ-COE-COE)



(✓Arm8)

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

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

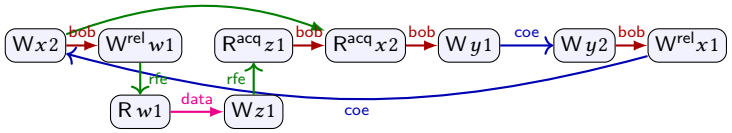


(✓Arm8)

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:

$$x := 2;\ w^{\mathsf{rel}} := 1;\ s := z^{\mathsf{acq}};\ r := x^{\mathsf{acq}};\ y := 1 \quad \| \quad y := 2;\ x^{\mathsf{rel}} := 1 \quad \| \quad r := w;\ z := r;$$

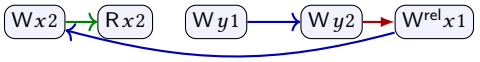
(RFI-RA-DATA-COE-COE)



(✗Arm8)

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

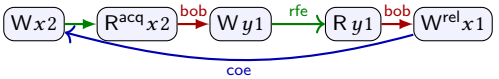
$$x := 2;\ r := x^{\mathsf{acq}};\ y := 1 \quad \| \quad y := 2;\ x^{\mathsf{rel}} := 1$$



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

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

(RFI-RFE-COE)

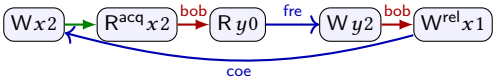


(✓Arm8)

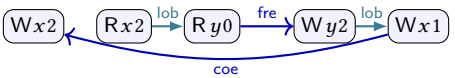
TSO variant `[rfi-fre-coe2]`:

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

(RFI-COE-COE2)



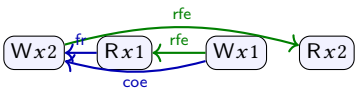
(✓Arm8)



(✓TSO)

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

$$x := 2;\ r := x \quad \| \quad x := 1;\ s := x$$



(✗TSO)

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

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

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

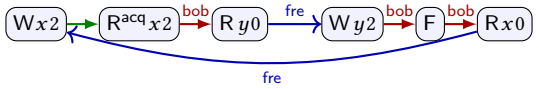
- [W]; po; [W]
- $d \overset{\mathsf{po}}{\dashrightarrow} e$ when $c \xrightarrow{\mathsf{rfe}} d \overset{\mathsf{po}}{\dashrightarrow} e$, from `(range(rfe) * _)`
- [R]; po; [W], from `(rfi^-1; lob)`

Ignoring fences and RMWs:

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

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

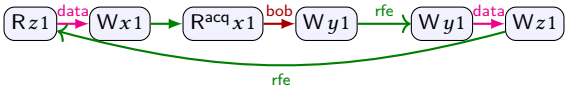
$$x := 2; r := x^{\mathsf{acq}}; s := y \parallel y := 2; \mathsf{F}; r := x \qquad \text{(RFI-FRE-FRE)}$$



(✓Arm8)

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

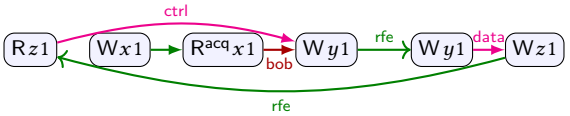
$$x := z; r := x^{\mathsf{acq}}; y := 1 \parallel z := y \qquad \text{(DATA-RFI-RFE-RFE)}$$



(✗Arm8)

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

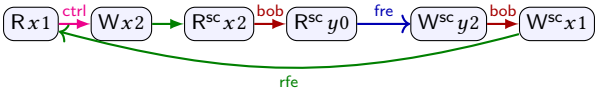
$$\texttt{if}(z)\{\}; x := 1; r := x^{\mathsf{acq}}; y := 1 \parallel z := y \qquad \text{(CTRL-RFI-RFE-RFE)}$$



(✗Arm8)

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

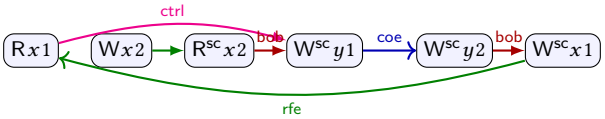
$$\texttt{if}(x)\{\}; x := 2; r := x^{\mathsf{sc}}; s := y^{\mathsf{sc}} \parallel y^{\mathsf{sc}} := 2; x^{\mathsf{sc}} := 1 \qquad \text{(CTRL-RFI-FRE-RFE)}$$



(✓Arm8)

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

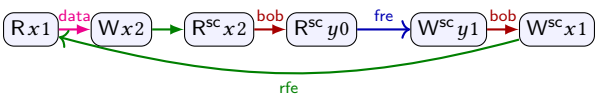
$$\texttt{if}(x)\{\}; x := 2; r := x^{\mathsf{sc}}; y^{\mathsf{sc}} := 1 \parallel y^{\mathsf{sc}} := 2; x^{\mathsf{sc}} := 1 \qquad \text{(CTRL-RFI-COE-RFE)}$$



(✗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)

11 DEAD STORE ELIMINATION, STORE FORWARDING, AND MONOTONICITY

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

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

One could do the same for scopes.

[Todo: The rest of this is very sketchy. It is difficult to get merging with alternate worlds not messing things up. Any kind of disjointness requirement imperils associativity.]

The semantics already validates:

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

It does not validate:

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

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

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

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

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

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

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

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

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

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

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

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

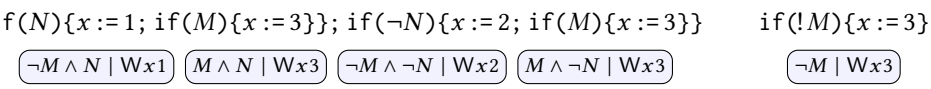
Full merge: $\mathtt{if}(M)\{x := 1\};\ x := 2$ can become $x := 2$.

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

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

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

We could have a lot merging:

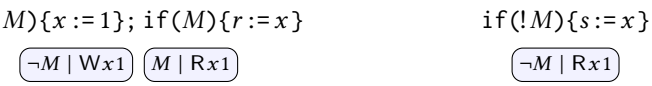


Full merge: $x := 1;\ \texttt{if}(M)\{r := x\}$ can become $x := 1;\ \texttt{if}(M)\{r := 1\}$.

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

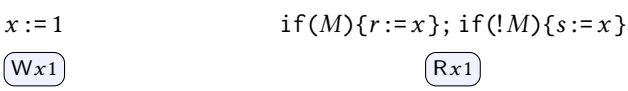
I don't think we need multi-merge for write-read. Reads only affect the world via the predicate transformer. Any conditional surrounding a read is baked into the predicate transformer, and so does not to persist in the preconditions of the actions themselves after the merge. Consider $r := 1;\ x := 2;\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

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



Another example:

Another example:



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

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

Require that the writes that coalesce have disjoint preconditions.

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

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

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

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

12 ALEX AIKEN PLDI KEYNOTE

- PAD Machines: Parallel, Accelerated (GPUs..), Distributed (memory)
- "Registered" memory: shared with network
- "Zero-copy" memory: shared with GPUs
  - http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#um-unified-memory-programming-hd Unified Memory offers a "single-pointer-to-data" model that is conceptually similar to CUDA's zero-copy memory. One key difference between the two is that with zero-copy allocations the physical location of memory is pinned in CPU system memory such that a program may have fast or slow access to it depending on where it is being accessed from. Unified Memory, on the other hand, decouples memory and execution spaces so that all data accesses are fast.
- Software manages "NUMA domains"
- In PAD machines, most computation happening on the accelerators.
- Bad Programming models:
  - Data Analytics, Machine Learning: Single program in Spark Hadoop, Etc. Accessible to many people, Low performance
  - HPC: MPI, OpenMP, Vector Intrinsics, Cuda, etc... Experts only, High performance
- Task based models are good.
  - Asynchronous functions that get mapped to real resources. May have sub-tasks or not.
  - Tasks capture locality:
    - Task is co-located with its arguments.
    - Tasks are a "local address space" model.
  - Tasks capture asynchrony
- Semantics
  - ✗Explicit Parallelism: X10, Chapel
  - ✓Implicit Parallelism (dataflow) — much easier to reason about (sequential semantics) — automated data movement
    - Layout Explicitly programmed: StarPU
    - Layout computed statically: PaRSEC/PTG, DPJ, Sequoia
    - Layout computed dynamically: Legion, PaRSEC/DTD, TensorFlow
- Partitioning:
  - Flat disjoint partitions: TensorFlow, PyTorch.
    - Not compositional: Problem is that different parts of the application may want different views of the data.
- Hierarchical: StarPU
  - Partitions created/destroyed
  - Partitions are disjoint
  - Only one partition of collection at a time
  - Only leaves can be used by application
  - Eager Repartitioning move data unnecessarily
- Legion allows multiple hierarchical partitions
  - Compositional: data movement is lazy
  - Task dependence analysis requires alias analysis

13 DITCH ARM7?

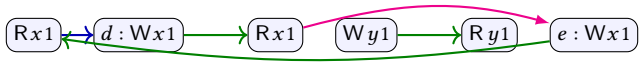
13.1 Two order idea

The two order idea from OOPSLA talk is:

- Require: $d \sqsubset e$ when $d \lhd e$ 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 $\boxed{\sqsubset}$ for this notion, with strong fulfillment.

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



($\boxed{\sqsubset}$)

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

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

13.2 Ditch Arm7

If we ditch ARM-v7, we may be able to do something nicer, but we cannot use IMM out of the box.

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

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

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

Additional stuff:

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

RMW:

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

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

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

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

- d writes v to x,
- e reads v from x,
- $d \leq e$, and

[3]Note we can *not* require

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

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

- if c writes to x then either $c \sqsubseteq d$ or $e \sqsubseteq c$.

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

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

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

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

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

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

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

13.3 Power versus ARM7

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

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

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

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

REFERENCES

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

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

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

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

William Ferreira, Matthew Hennessy, and Alan Jeffrey. 1996. A Theory of Weak Bisimulation for Core CML. In *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming, ICFP 1996, Philadelphia, Pennsylvania, USA, May 24-26, 1996*, Robert Harper and Richard L. Wexelblat (Eds.). ACM, 201–212. https://doi.org/10.1145/232627.232649

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

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

Alan Jeffrey, James Riely, Mark Batty, Simon Cooksey, Ilya Kaysin, and Anton Podkopaev. 2021. The Leaky Semicolon: Compositional Semantic Dependencies for Relaxed-Memory Concurrency. https://github.com/chicago-relaxed-memory/seqcomp.

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

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

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

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

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

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