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), 10 pages.

1 MODEL

1.1 Preliminaries

The syntax is built from

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

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

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

We model the following language.

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

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

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

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

Commands, aka *statements*, S, include memory accesses at a given mode, as well as the usual structural constructs. Following [Ferreira et al. 1996], $\rrbracket$ denotes parallel composition. If $(S_1 \rrbracket_{\gamma} S_2)$ is executed with thread ID α, then S_2 runs with ID γ and S_1 continues under ID α. Top level programs run with thread ID $\mathbf{0}$. In examples, we usually drop thread IDs. We use the symmetric $\parallel$ operator when there is no continuation after the parallel composition.

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

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

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

The semantics is built from the following.

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

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

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

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

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

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

We also require that there are subsets of actions, distinguishing *read* and *release* actions. We require several binary relations between actions, detailed in the next subsection: *overlaps*, *strongly-overlaps*, *matches*, *strongly-matches*, *strongly-fences*, *blocks*, *sync-delays* and *co-delays*. We require that strongly-overlaps implies overlaps and that strongly-matches implies matches implies blocks implies overlaps.

1.2 Actions

We combine access and fence modes into a single order: $\mathsf{wk} \longrightarrow \mathsf{rlx} \rightrightarrows \begin{smallmatrix}\mathsf{rel}\\ \mathsf{acq}\end{smallmatrix} \rightrightarrows \mathsf{ra} \longrightarrow \mathsf{sc}$. We write $\mu \sqsubseteq \nu$ for this order. Let $\mu \sqcup \nu$ denote the least upper bound of μ and ν.

Let actions be reads, writes and fences:

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

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

We say a *matches* b if $a = (\mathsf{W}xv)$ and $b = (\mathsf{R}xv)$.

We say a *blocks* b if $a = (\mathsf{W}x)$ and $b = (\mathsf{R}x)$, regardless of value.

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

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

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

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

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

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

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

(1) $\alpha_1 = \alpha_2$, or

(2a) $\mu_1, \mu_2 \neq \mathsf{wk}$,

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

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

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

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

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

1.3 Pomsets with Predicate Transformers

Definition 1.2. A *predicate transformer* is a function $\tau : \Phi \to \Phi$ such that

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

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

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

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

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

We write τ as an abbreviation of τ^E.

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

(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) $\kappa(e)$ is satisfiable,

(M4) $\tau : 2^{\mathcal{E}} \to \Phi \to \Phi$ is a *family of predicate transformers* over E,

(M5) $\checkmark : \Phi$ is a *termination condition*, such that

 (M5a) $\checkmark \vDash \tau(\mathsf{tt})$,

(M6) $\trianglelefteq : (E \times E)$ is a partial order capturing *dependency*,

(M7) $\leq : (E \times E)$ is a partial order capturing *synchronization*,

(M8) $\sqsubseteq : (E \times E)$ is a partial order capturing *per-location order*, such that

 (M8a) if $\lambda(d)$ overlaps $\lambda(e)$ then $d \leq e$ implies $d \sqsubseteq e$,

(M9) $\mathsf{rmw} : E \to E$ is a partial function capturing read-modify-write *atomicity*, such that

 (M9a) if $d \xrightarrow{\mathsf{rmw}} e$ then $\lambda(e)$ blocks $\lambda(d)$,

 (M9b) if $d \xrightarrow{\mathsf{rmw}} e$ then $d \leq e$ and $d \sqsubseteq e$,

 (M9c) if $\lambda(c)$ overlaps $\lambda(d)$ then

  (i) if $d \xrightarrow{\mathsf{rmw}} e$ then $c \trianglelefteq e$ implies $c \trianglelefteq d$, $c \leq e$ implies $c \leq d$, $c \sqsubseteq e$ implies $c \sqsubseteq d$,

  (ii) if $d \xrightarrow{\mathsf{rmw}} e$ then $d \trianglelefteq c$ implies $e \trianglelefteq c$, $d \leq c$ implies $e \leq c$, $d \sqsubseteq c$ implies $e \sqsubseteq c$.

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

(C2a) if $d \xrightarrow{\mathsf{rf}} e$ then $\lambda(d)$ matches $\lambda(e)$,

(C6) if $d \xrightarrow{\mathsf{rf}} e$ then $d \trianglelefteq e$,

(C7a) if $d' \leq d \xrightarrow{\mathsf{rf}} e \leq e'$ and $\lambda(d')$ strongly-matches $\lambda(e')$ then $d' \leq e'$,

(C7b) if $\lambda(d)$ strongly-fences $\lambda(e)$ then either $d \leq e$ or $e \leq d$,

(C8a) if $d \xrightarrow{\mathsf{rf}} e$ then $d \sqsubseteq e$,

(C8b) if $d \xrightarrow{\mathsf{rf}} e$ and $\lambda(c)$ blocks $\lambda(e)$ then either $c \underset{\sim}{\sqsubset} d$ or $e \underset{\sim}{\sqsubset} c$,

 where $d' \underset{\sim}{\sqsubset} e'$ when $e' \sqsubseteq d'$ implies $d' = e'$ and $\lambda(d')$ strongly-overlaps $\lambda(e')$ implies $d' \sqsubseteq e'$.

A candidate pomset with rf is *complete* if

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

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

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

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

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

1.4 Semantics

See Figure 2.

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

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

1.5 Address Calculation

Definition 1.5. If $P \in \mathit{WRITE}(L, M, \mu, \sigma)_\alpha$ then $(\exists \ell \in \mathcal{V})$ $(\exists v \in \mathcal{V})$

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

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

(R1) if $d, e \in E$ then $d = e$,
(R2) $\lambda(e) = \alpha \mathsf{R}^{\mu}_{\sigma}[\ell] v$
(R3) $\kappa(e) \wedge L{=}\ell$,
(R4a) $(\forall e \in E \cap D)$ $\tau^D(\psi) \vDash (L{=}\ell \Rightarrow v{=}s_e) \Rightarrow \psi[s_e/r]$,
(R4b) $(\forall e \in E \setminus D)$ $\tau^D(\psi) \vDash ((L{=}\ell \Rightarrow v{=}s_e) \vee (L{=}\ell \Rightarrow [\ell]{=}s_e)) \Rightarrow \psi[s_e/r]$,
(R4c) $(\forall s)$ if $E = \emptyset$ then $\tau^D(\psi) \vDash \psi[s/r]$,
(R5) if $E = \emptyset$ and $\mu \neq \mathsf{rlx}$ then $\checkmark \vDash \mathsf{ff}$.

1.6 If-closure

Definition 1.6. If $P \in \mathit{WRITE}(x, M, \mu, \sigma)_\alpha$ then $(\exists v : E \to \mathcal{V})$ $(\exists \theta : E \to \Phi)$

---

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

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

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

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

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

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

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

- (P1) $E = (E_1 \uplus E_2)$,
- (P2) $\lambda = (\lambda_1 \cup \lambda_2)$,
- (P3a) if $e \in E_1$ then $\kappa(e) \vDash \kappa_1(e)$,
- (P3b) if $e \in E_2$ then $\kappa(e) \vDash \kappa_2(e)$,
- (P4) $\tau^D(\psi) \vDash \tau_1^D(\psi)$,
- (P5) $\checkmark \vDash \checkmark_1 \wedge \checkmark_2$,
- (P6) $\trianglelefteq\ \supseteq (\trianglelefteq_1 \cup \trianglelefteq_2)$,
- (P7) $\leq\ \supseteq (\leq_1 \cup \leq_2)$,
- (P8) $\sqsubseteq\ \supseteq (\sqsubseteq_1 \cup \sqsubseteq_2)$,
- (P9) $\mathsf{rmw} = (\mathsf{rmw}_1 \cup \mathsf{rmw}_2)$.

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

- (S1) $E = (E_1 \cup E_2)$,
- (S2) (S6) (S7) (S8) (S9) as for *PAR*,
- (S3a) if $e \in E_1 \setminus E_2$ then $\kappa(e) \vDash \kappa_1(e)$,
- (S3b) if $e \in E_2 \setminus E_1$ then $\kappa(e) \vDash \kappa_2'(e)$,
- (S3c) if $e \in E_1 \cap E_2$ then $\kappa(e) \vDash \kappa_1(e) \vee \kappa_2'(e)$,
- (S3d) if $\lambda_2(e)$ is a release then $\kappa(e) \vDash \checkmark_1$,
- (S4) $\tau^D(\psi) \vDash \tau_1^D(\tau_2^D(\psi))$,
- (S5) $\checkmark \vDash \checkmark_1 \wedge \tau_1(\checkmark_2)$,
- (S7a) if $\lambda_1(d)$ sync-delays $\lambda_2(e)$ then $d \leq e$,
- (S8a) if $\lambda_1(d)$ co-delays $\lambda_2(e)$ then $d \sqsubseteq e$,

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

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

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

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

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

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

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

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

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

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

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

Fig. 1. Semantics of programs

- (W1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,
- (W2) $\lambda(e) = \alpha\mathsf{W}^\mu_\sigma x v_e$,
- (W3) $\kappa(e) \vDash \theta_e \wedge M{=}v_e$,
- (W4) $\tau^D(\psi) \vDash \theta_e \Rightarrow \psi[M/x]$,
- (W5) $\checkmark \vDash \theta_e \Rightarrow M{=}v_e$,

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

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

- (P1) $E = (E_1 \uplus E_2)$,
- (P2) $\lambda = (\lambda_1 \cup \lambda_2)$,
- (P3a) if $e \in E_1$ then $\kappa(e) \equiv \kappa_1(e)$,
- (P3b) if $e \in E_2$ then $\kappa(e) \equiv \kappa_2(e)$,
- (P4) $\tau^D(\psi) \equiv \tau_1^D(\psi)$,
- (P5) $\checkmark \equiv \checkmark_1 \wedge \checkmark_2$,
- (P6) $\trianglelefteq \supseteq (\trianglelefteq_1 \cup \trianglelefteq_2)$,
- (P7) $\leq \supseteq (\leq_1 \cup \leq_2)$,
- (P8) $\sqsubseteq \supseteq (\sqsubseteq_1 \cup \sqsubseteq_2)$,
- (P9) $\mathsf{rmw} = (\mathsf{rmw}_1 \cup \mathsf{rmw}_2)$.

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

- (S1) $E = (E_1 \cup E_2)$,
- (S2) (S6) (S7) (S8) (S9) as for *PAR*,
- (S3a) if $e \in E_1 \setminus E_2$ then $\kappa(e) \equiv \kappa_1(e)$,
- (S3b) if $e \in E_2 \setminus E_1$ then $\kappa(e) \equiv \kappa_2'(e) \wedge \checkmark_1(e)$,
- (S3c) if $e \in E_1 \cap E_2$ then $\kappa(e) \equiv (\kappa_1(e) \vee \kappa_2'(e)) \wedge \checkmark_1(e)$,
- (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 $\lambda_2(e)$ and $\kappa_1(d) \wedge \kappa_2(e)$ is satisfiable then $d \leq e$,
- (S8a) if $\lambda_1(d)$ co-delays $\lambda_2(e)$ and $\kappa_1(d) \wedge \kappa_2(e)$ is satisfiable then $d \sqsubseteq e$,

where $\kappa_2'(e) = \tau_1(\kappa_2(e))$ if $\lambda(e)$ is a read—otherwise $\kappa_2'(e) = \tau_1^{\downarrow e}(\kappa_2(e))$, where $\downarrow e = \{c \mid c \triangleleft e\}$;
where $\checkmark_1(e) = \checkmark_1$ if $\lambda(e)$ is a release—otherwise $\checkmark_1(e) = \mathrm{tt}$.

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

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

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

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

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

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

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

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

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

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

Fig. 2. Semantics of programs

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

(R1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,
(R2) $\lambda(e) = \alpha \mathsf{R}^\mu_\sigma x v_e$
(R3) $\kappa(e) \vDash \theta_e$,
(R4a) $(\forall e \in E \cap D)\ \tau^D(\psi) \vDash \theta_e \Rightarrow v_e{=}s_e \Rightarrow \psi[s_e/r]$,
(R4b) $(\forall e \in E \setminus D)\ \tau^D(\psi) \vDash \theta_e \Rightarrow (v_e{=}s_e \vee x{=}s_e) \Rightarrow \psi[s_e/r]$,
(R4c) $(\forall s)\ \tau^D(\psi) \vDash (\bigwedge_{e \in E} \neg\theta_e) \Rightarrow \psi[s/r]$,
(R5) if $E = \emptyset$ and $\mu \neq \mathsf{rlx}$ then $\checkmark \vDash \mathsf{ff}$.

1.7 Address Calculation and If-closure

Definition 1.7. If $P \in \mathit{WRITE}(L,\ M,\ \mu,\ \sigma)_\alpha$ then $(\exists \ell : E \to \mathcal{V})\ (\exists v : E \to \mathcal{V})\ (\exists \theta : E \to \Phi)$

(W1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,
(W2) $\lambda(e) = \alpha \mathsf{W}^\mu_\sigma [\ell] v_e$,
(W3) $\kappa(e) \vDash \theta_e \wedge L{=}\ell_e \wedge M{=}v_e$,
(W4a) $\tau^D(\psi) \vDash \theta_e \Rightarrow (L{=}\ell) \Rightarrow \psi[M/[\ell]]$,
(W4b) $(\forall k)$ $\tau^D(\psi) \vDash (\bigwedge_{e \in E} \neg\theta_e) {\Rightarrow} (L{=}k) {\Rightarrow} \psi[M/[k]]$
(W5a) $\checkmark \vDash \theta_e \Rightarrow L{=}\ell_e \wedge M{=}v_e$,
(W5b) $\checkmark \vDash \bigvee_{e \in E} \theta_e$.

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

(R1) if $\theta_d \wedge \theta_e$ is satisfiable then $d = e$,
(R2) $\lambda(e) = \alpha \mathsf{R}^\mu_\sigma [\ell] v_e$
(R3) $\kappa(e) \vDash \theta_e \wedge L{=}\ell_e$,
(R4a) $(\forall e \in E \cap D)\ \tau^D(\psi) \vDash \theta_e \Rightarrow (L{=}\ell_e \Rightarrow v_e{=}s_e) \Rightarrow \psi[s_e/r]$,
(R4b) $(\forall e \in E \setminus D)\ \tau^D(\psi) \vDash \theta_e \Rightarrow ((L{=}\ell_e \Rightarrow v_e{=}s_e) \vee (L{=}\ell_e \Rightarrow [\ell]{=}s_e)) \Rightarrow \psi[s_e/r]$,
(R4c) $(\forall s)\ \tau^D(\psi) \vDash (\bigwedge_{e \in E} \neg\theta_e) \Rightarrow \psi[s/r]$,
(R5) if $E = \emptyset$ and $\mu \neq \mathsf{rlx}$ then $\checkmark \vDash \mathsf{ff}$.

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

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

If $P \in \mathit{FADD}(r, L, M, \mu, \nu)$ then $(\exists P_1 \in \mathit{SEQ}(\mathit{READ}'(r,\ L,\ \mu),\ \mathit{WRITE}(L,\ r{+}M,\ \nu)))$
(U1) if $\lambda_1(e)$ is a write then there is a read $\lambda_1(d)$ such that $\kappa(e) \vDash \kappa(d)$ and $d \xrightarrow{\mathsf{rmw}} e$.
If $P \in \mathit{EXCHG}(r, L, M, \mu, \nu)$ then $(\exists P_1 \in \mathit{SEQ}(\mathit{READ}'(r,\ L,\ \mu),\ \mathit{WRITE}(L,\ M,\ \nu)))$
(U1) if $\lambda_1(e)$ is a write then there is a read $\lambda_1(d)$ such that $\kappa(e) \vDash \kappa(d)$ and $d \xrightarrow{\mathsf{rmw}} e$.
If $P \in \mathit{CAS}(r, L, M, N, \mu, \nu)$ then $(\exists P_1 \in \mathit{SEQ}(\mathit{READ}'(r,\ L,\ \mu),\ \mathit{IF}(r{=}M,\ \mathit{WRITE}(L,\ N,\ \nu),\ \mathit{SKIP})))$
(U1) if $\lambda_1(e)$ is a write then there is a read $\lambda_1(d)$ such that $\kappa(e) \vDash \kappa(d)$ and $d \xrightarrow{\mathsf{rmw}} e$.

2 PROPERTIES

Lemma 2.1. *(a)* $\mathcal{P} = (\mathcal{P} \mathbin{\|} \mathit{SKIP}) = (\mathcal{P}\,;\, \mathit{SKIP}) = (\mathit{SKIP}\,;\, \mathcal{P})$.
(b) $(\mathcal{P}_1 \mathbin{\|} \mathcal{P}_2) \mathbin{\|} \mathcal{P}_3 = \mathcal{P}_1 \mathbin{\|} (\mathcal{P}_2 \mathbin{\|} \mathcal{P}_3)$.
(c) $(\mathcal{P}_1\,;\, \mathcal{P}_2)\,;\, \mathcal{P}_3 = \mathcal{P}_1\,;\, (\mathcal{P}_2\,;\, \mathcal{P}_3)$.
(d) $\mathtt{if}(\phi)\{\mathcal{P}_1\}\,\mathtt{else}\,\{\mathcal{P}_2\} = \mathtt{if}(\phi)\{\mathcal{P}_1\};\, \mathtt{if}(\neg\phi)\{\mathcal{P}_2\} = \mathtt{if}(\neg\phi)\{\mathcal{P}_2\};\, \mathtt{if}(\phi)\{\mathcal{P}_1\}$.
(e) $\mathtt{if}(\phi)\{\mathcal{P}_1\}\,\mathtt{else}\,\{\mathcal{P}_2\} = \mathcal{P}_1$ *if* ϕ *is a tautology.*
(f) $\mathtt{if}(\phi)\{\mathtt{if}(\psi)\{\mathcal{P}\}\} = \mathtt{if}(\phi \wedge \psi)\{\mathcal{P}\}$.
(g) $\mathtt{if}(\phi)\{\mathcal{P}_1\,;\, \mathcal{P}_3\}\,\mathtt{else}\,\{\mathcal{P}_2\,;\, \mathcal{P}_3\} \supseteq \mathtt{if}(\phi)\{\mathcal{P}_1\}\,\mathtt{else}\,\{\mathcal{P}_2\};\, \mathcal{P}_3$.
(h) $\mathtt{if}(\phi)\{\mathcal{P}_1\,;\, \mathcal{P}_2\}\,\mathtt{else}\,\{\mathcal{P}_1\,;\, \mathcal{P}_3\} \supseteq \mathcal{P}_1;\, \mathtt{if}(\phi)\{\mathcal{P}_2\}\,\mathtt{else}\,\{\mathcal{P}_3\}$.
(i) $\mathtt{if}(\phi)\{\mathcal{P}\}\,\mathtt{else}\,\{\mathcal{P}\} \supseteq \mathcal{P}$.

Proof. Straightforward calculation. (a) requires M5a for the termination condition in $(\mathcal{P}\,;\, \mathit{SKIP})$. (c) requires both conjunction closure (X2, for the termination condition) and disjunction closure (X3, for the predicate transformers themselves).

(d) requires s7a and s8a not impose order when $\kappa_1(d) \wedge \kappa_2(e)$ is unsatisfiable, which in turn requires that κ calculates *weakest* preconditions, rather than simple preconditions (see [Jeffrey and Riely 2021]).

(e) requires M3a.

In §1.6, we refine the semantics to validate the reverse inclusions for (g), (h), and (i). □

Definition 2.2. P_2 is an *augment* of P_1 if

(1) $E_2 = E_1$, (2) $\lambda_2(e) = \lambda_1(e)$, (3) $\kappa_2(e) \equiv \kappa_1(e)$, (4) $\tau_2^D(\psi) \equiv \tau_1^D(\psi)$, (5) $\checkmark_2 \equiv \checkmark_1$, (6) $\mathsf{rf}_2 \supseteq \mathsf{rf}_1$, (7) $\leq_2 \supseteq \leq_1$.

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

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

3 ACCESS ELIMINATION AND "MONOTONICITY"

As noted in §??, the semantics of Figure 2 validates elimination of irrelevant relaxed reads. In §??, we discussed redundant read elimination. Figure 2 also validates elimination of writes of the same value. However, Figure 2 does not validate general write elimination, where, for example, $(x := 1;\ x := 2)$ is refined to $x := 2$. Nor does it validate store forwarding, where, for example, $(x := 1;\ r := x)$ is refined to $(x := 1;\ r := 1)$.

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

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

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

Let $\mathsf{merge} : \mathcal{A} \times \mathcal{A} \to 2^{\mathcal{A}}$ be defined as follows.

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

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

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

Then we can replace s2-s3 in Figure 2 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 $\lambda(e) \in \mathsf{merge}(\lambda_1(e),\ \lambda_2(e))$,
(s3a) if $e \in E_1 \setminus E_2$ then $\kappa(e) \vDash \kappa_1(e)$,
(s3b) if $e \in E_2 \setminus E_1$ then $\kappa(e) \vDash \kappa'_2(e)$,
(s3c) if $e \in E_1 \cap E_2$ then either
- $\lambda(e) = \lambda_1(e) = \lambda_2(e)$ and $\kappa(e) \equiv \kappa_1(e) \vee \kappa'_2(e)$,
- $\lambda(e) = \lambda_1(e) \neq \lambda_2(e)$ and $\kappa(e) \equiv \kappa_1(e)$ and $\kappa'_2(e) \vDash \kappa_1(e)$ (write-read),
- $\lambda(e) = \lambda_2(e) \neq \lambda_1(e)$ and $\kappa(e) \equiv \kappa_1(e) \equiv \kappa'_2(e)$ (write-write).

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

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

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

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

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

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

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 $r := 1;\ x := 2;\ \texttt{if}(M)\{r := 2\}$.

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

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

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

Idea for multi-merge. Use $E_1' \subseteq E_1$, with a function $\mathsf{coalesce} : E_1 \to E_1'$ that shows how events merge. Require that the events that coalesce have disjoint preconditions. Then each of them has to merge into the same event from E_2 using the merge function.

REFERENCES

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

Alan Jeffrey and James Riely. 2021. Sequential Composition for Relaxed Memory: Pomsets with Predicate Transformers. 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/