RustBelt Relaxed - Technical Appendix

This work is accompanied by a machine-checked formalization in Coq, which includes all definitions, theorems, lemmas and proofs in this appendix, with the exception of the correspondence proof (§2).

Contents

1 Language

1.1 Grammar

Our language is an extension of the original RustBelt's λ_{Rust} with the relaxed memory semantics of ORC11 (§1.2). λ_{Rust} is a lambda calculus with integers, locations with explicit allocation and deallocation, and a notion of poison value ☣. Instead of `sc` for atomic accesses, we use release `rel`, acquire `acq`, and relaxed `rlx` accesses together with fences.

The grammar is given in Fig. 1. Several syntactic sugars are taken as-is from the original RustBelt, given in Fig. 2. We refer the reader to the original RustBelt appendix (Jung et al. [2017]) for more explanation of the grammar and syntactic sugars.

1.2 Operational Semantics

Following iGPS (Kaiser et al. [2017]) we use an operational semantics for relaxed memory so that it can be instantiated in Iris. For this work, we extend iGPS's operational semantics for RA+NA to include relaxed accesses and fences.

The semantics, called ORC11, is defined by three sub semantics: the expressions semantics (Fig. 5), the machine semantics (Fig. 9), and the race-detecting semantics (Fig. 6 and Fig. 7). The combined thread pool semantics is given in Fig. 10 and Fig. 11. In §2, we sketch a proof of correspondence that relates ORC11 to the axiomatic semantics from Lahav et al. [2017].

$$
\begin{array}{rcl}
 & & z \in \mathbb{Z} \\
\mathit{Expr} \ni e & ::= & v \mid x \\
 & & \mid e.e \mid e + e \mid e - e \mid e \leq e \mid e == e \\
 & & \mid e(\overline{e}) \\
 & & \mid {}^{*o}e \\
 & & \mid e_1 :=_o e_2 \\
 & & \mid \texttt{CAS}(e_0, e_1, e_2, o_f, o_r, o_w) \\
 & & \mid \texttt{alloc}(e) \\
 & & \mid \texttt{free}(e_1, e_2) \\
 & & \mid \texttt{case}\, e\, \texttt{of}\, \overline{e} \\
 & & \mid \texttt{fork}\{\, e \,\} \\
 & & \mid \texttt{fence}_o \\
\mathit{Val} \ni v & ::= & \cancel{\emptyset} \mid \ell \mid z \mid \texttt{rec}\, f(\overline{x}) := e \\
\mathit{Loc} \ni \ell & ::= & (i, n) \\
\mathit{Order} \ni o & ::= & \texttt{acq} \mid \texttt{rel} \mid \texttt{rlx} \mid \texttt{na} \\
\mathit{Ctx} \ni K & ::= & \bullet \\
 & & \mid K.e \mid v.K \mid K + e \mid v + K \mid K - e \mid v - K \\
 & & \mid K \leq e \mid v \leq K \mid K == e \mid v == K \\
 & & \mid K(\overline{e}) \mid v(\overline{v} +\!\!+ [K] +\!\!+ \overline{e}) \\
 & & \mid {}^{*o}K \mid K :=_o e \mid v :=_o K \\
 & & \mid \texttt{CAS}(K, e_1, e_2, o_f, o_r, o_w) \\
 & & \mid \texttt{CAS}(v_0, K, e_2, o_f, o_r, o_w) \\
 & & \mid \texttt{CAS}(v_0, v_1, K, o_f, o_r, o_w) \\
 & & \mid \texttt{alloc}(K) \\
 & & \mid \texttt{free}(K, e_2) \\
 & & \mid \texttt{free}(e_1, K) \\
 & & \mid \texttt{case}\, K\, \texttt{of}\, \overline{e}
\end{array}
$$

Figure 1: Language syntax.

$$
\begin{aligned}
\texttt{funrec}\, f(\overline{x})\, \texttt{ret}\, k := e &:= \texttt{rec}\, f([k] +\!\!+ \overline{x}) := e \\
\texttt{let}\, x = e\, \texttt{in}\, e' &:= (\texttt{rec}\, \_([x]) := e')(e) \\
e'; e &:= \texttt{let}\, \_ = e'\, \texttt{in}\, e \\
\texttt{letcont}\, k(\overline{x}) := e\, \texttt{in}\, e' &:= \texttt{let}\, k = (\texttt{rec}\, k(\overline{x}) := e)\, \texttt{in}\, e' \\
\texttt{jump}\, k(\overline{e}) &:= k(\overline{e}) \\
\texttt{call}\, f(\overline{e})\, \texttt{ret}\, k &:= f([k] +\!\!+ \overline{e})
\end{aligned}
$$

$$
\begin{aligned}
\texttt{false} &:= 0 \\
\texttt{true} &:= 1 \\
\texttt{if}\, e_0\, \texttt{then}\, e_1\, \texttt{else}\, e_2 &:= \texttt{case}\, e_0\, \texttt{of}\, [e_1, e_2]
\end{aligned}
$$

$$
\begin{aligned}
{}^*e &:= {}^{*\texttt{na}}e \\
e_1 := e_2 &:= e_1 :=_{\texttt{na}} e_2 \\
\texttt{new} &:= \texttt{rec}\, \mathit{new}(\mathit{size}) := \\
&\quad\ \texttt{if}\, \mathit{size} == 0\, \texttt{then}\, (42, 1337)\, \texttt{else}\, \texttt{alloc}(\mathit{size}) \\
\texttt{delete} &:= \texttt{rec}\, \mathit{delete}(\mathit{size}, \mathit{ptr}) := \\
&\quad\ \texttt{if}\, \mathit{size} == 0\, \texttt{then}\, \text{☣}\, \texttt{else}\, \texttt{free}(\mathit{size}, \mathit{ptr}) \\
\texttt{memcpy} &:= \texttt{rec}\, \mathit{memcpy}(\mathit{dst}, \mathit{len}, \mathit{src}) := \\
&\quad\ \texttt{if}\, \mathit{len} \leq 0\, \texttt{then}\, \text{☣}\, \texttt{else} \\
&\quad\ \mathit{dst}.0 := \mathit{src}.0; \\
&\quad\ \mathit{memcpy}(\mathit{dst}.1, \mathit{len} - 1, \mathit{src}.1) \\
e_1 :=_n {}^*e_2 &:= \mathsf{memcpy}(e_1, n, e_2) \\
e :\overset{\mathsf{inj}\, i}{=\!=} () &:= e.0 := i \\
e_1 :\overset{\mathsf{inj}\, i}{=\!=} e_2 &:= e_1.0 := i; e_1.1 := e_2 \\
e_1 :\overset{\mathsf{inj}\, i}{=\!=}_n {}^*e_2 &:= e_1.0 := i; e_1.1 :=_n {}^*e_2
\end{aligned}
$$

$$
\begin{aligned}
\texttt{skip} &:= \texttt{let}\, x = \text{☣}\, \texttt{in}\, \text{☣} \\
\texttt{newlft} &:= \text{☣} \\
\texttt{endlft} &:= \texttt{skip}
\end{aligned}
$$

Figure 2: Syntactic sugars.

$$
\begin{aligned}
\pi \in \mathit{Thread} &::= \mathbb{N} \\
t \in \mathit{Time} &::= \mathbb{N}^+ \\
\omega \in \mathit{MsgVal} &::= \dagger \mid \Phi \mid v \in \mathit{Val} \\
\mathit{ActionIds} &::= 2^{\mathbb{N}^+} \\
V \in \mathit{View} &::= \mathit{Loc} \xrightarrow{\text{fin}} \{\mathsf{w} : \mathit{Time}, \mathsf{aw} : \mathit{ActionIds}, \mathsf{nr} : \mathit{ActionIds}, \mathsf{ar} : \mathit{ActionIds}\} \\
\mathcal{V} \in \mathit{ThreadView} &::= \left\{\mathsf{rel} : \mathit{Loc} \xrightarrow{\text{fin}} \mathit{View}, \mathsf{frel} : \mathit{View}, \mathsf{cur} : \mathit{View}, \mathsf{acq} : \mathit{View}\right\} \\
m \in \mathit{ExtMsg} &::= \left\{\mathsf{ts} : \mathit{Time}, \mathsf{val} : \mathit{MsgVal}, \mathsf{view} : \mathit{View}^?\right\} \\
\mathcal{M} \in \mathit{MsgPool} &::= \mathit{Loc} \xrightarrow{\text{fin}} \mathit{Time} \xrightarrow{\text{fin}} \left\{\mathsf{val} : \mathit{MsgVal}, \mathsf{view} : \mathit{View}^?\right\} \\
\mathcal{N} \in \mathit{NARace} &::= \mathit{View} \\
\varsigma \in \mathit{GlobalState} &::= \mathit{MsgPool} \times \mathit{NARace} \\
\mathit{MemEvent} \ni \varepsilon &::= \langle \text{Alloc}, \ell, n \in \mathbb{N}^+ \rangle \\
&\quad \mid \langle \text{Dealloc}, \ell, n \in \mathbb{N}^+ \rangle \\
&\quad \mid \langle \text{Read}, \ell, v, o \rangle \\
&\quad \mid \langle \text{Write}, \ell, v, o \rangle \\
&\quad \mid \langle \text{Update}, \ell, v_r, v_w, o_r, o_w \rangle \\
&\quad \mid \langle \text{Fence}, o \rangle
\end{aligned}
$$

Figure 3: Machine state definitions.

$$\omega \in \text{Readable}(\ell, \mathcal{M}, \mathcal{V}) := \exists t.\, \mathcal{M}(\ell)(t) = (\omega, \_) \wedge t \leq \mathcal{V}.\mathsf{cur}(\ell)$$

$\boxed{\omega \equiv v}$

$$v \equiv v \qquad \dagger \equiv \text{☣}$$

$\boxed{\ell \in \text{unalloc}(\mathcal{M})}$

$$\frac{\ell \notin \text{dom}(\mathcal{M})}{\ell \in \text{unalloc}(\mathcal{M})} \qquad \frac{\exists t.\, \mathcal{M}(\ell)(t) = (\text{†}, \_)}{\ell \in \text{unalloc}(\mathcal{M})}$$

$\boxed{\mathcal{M} \vdash v_1 = v_2}$

$$\mathcal{M} \vdash z = z \qquad \mathcal{M} \vdash \ell = \ell \qquad \frac{\ell_1 \in \text{unalloc}(\mathcal{M}) \vee \ell_2 \in \text{unalloc}(\mathcal{M})}{\mathcal{M} \vdash \ell_1 = \ell_2}$$

$\boxed{\vdash v_1 \neq v_2}$

$$\frac{z_1 \neq z_2}{\vdash z_1 \neq z_2} \qquad \frac{\ell_1 \neq \ell_2}{\vdash \ell_1 \neq \ell_2} \qquad \vdash \ell \neq 0 \qquad \vdash 0 \neq \ell$$

$\boxed{\vdash v_1 =^? v_2}$

$$\vdash z_1 =^? z_2 \qquad \vdash \ell_1 =^? \ell_2 \qquad \vdash \ell =^? 0 \qquad \vdash 0 =^? \ell$$

$\boxed{o_1 \sqsubseteq o_2}$

$$\texttt{na} \sqsubseteq \texttt{rlx} \qquad \texttt{na} \sqsubseteq \texttt{acq} \qquad \texttt{na} \sqsubseteq \texttt{rel} \qquad \texttt{rlx} \sqsubseteq \texttt{acq} \qquad \texttt{rlx} \sqsubseteq \texttt{rel}$$

Figure 4: Auxilliary relations.

$\boxed{\mathcal{M}, \mathcal{V} \vdash e \xrightarrow{\varepsilon^?} e_1', e_2'^?}$

OE-ECTX
$$\frac{e \to e_1', e_2'^?}{\mathcal{M}, \mathcal{V} \vdash K[e] \to K[e_1'], e_2'^?}$$

OE-PROJ
$$\mathcal{M}, \mathcal{V} \vdash \ell.n \to \ell + n$$

OE-ADD
$$\frac{z_1 + z_2 = z'}{\mathcal{M}, \mathcal{V} \vdash z_1 + z_2 \to z'}$$

OE-SUB
$$\frac{z_1 - z_2 = z'}{\mathcal{M}, \mathcal{V} \vdash z_1 - z_2 \to z'}$$

OE-LE-TRUE
$$\frac{z_1 \leq z_2}{\mathcal{M}, \mathcal{V} \vdash z_1 \leq z_2 \to 1}$$

OE-LE-FALSE
$$\frac{z_1 > z_2}{\mathcal{M}, \mathcal{V} \vdash z_1 \leq z_2 \to 0}$$

OE-EQ-TRUE
$$\frac{\mathcal{M} \vdash v_1 = v_2}{\mathcal{M}, \mathcal{V} \vdash v_1 == v_2 \to 1}$$

OE-EQ-FALSE
$$\frac{\vdash v_1 \neq v_2}{\mathcal{M}, \mathcal{V} \vdash v_1 == v_2 \to 0}$$

OE-ALLOC
$$\frac{n > 0}{\mathcal{M}, \mathcal{V} \vdash \texttt{alloc}(n) \xrightarrow{\langle \text{Alloc}, \ell, n \rangle} \ell}$$

OE-FREE
$$\frac{n > 0}{\mathcal{M}, \mathcal{V} \vdash \texttt{free}(n, \ell) \xrightarrow{\langle \text{Dealloc}, \ell, n \rangle} \text{☣}}$$

OE-DEREF
$$\frac{\omega \equiv v}{\mathcal{M}, \mathcal{V} \vdash {}^{*o}\ell \xrightarrow{\langle \text{Read}, \ell, \omega, o \rangle} v}$$

OE-ASSIGN
$$\mathcal{M}, \mathcal{V} \vdash \ell :=_o v \xrightarrow{\langle \text{Write}, \ell, v, o \rangle} \text{☣}$$

OE-CAS-FAIL
$$\frac{\begin{array}{l}\texttt{rlx} \sqsubseteq o_f \\ \texttt{rlx} \sqsubseteq o_r \\ \texttt{rlx} \sqsubseteq o_w\end{array} \quad (\forall \omega \in \text{Readable}(\ell, \mathcal{M}, \mathcal{V}).\, \exists v'.\, \omega \equiv v' \wedge \vdash v_1 =^? v') \quad \vdash v_1 \neq v_r}{\mathcal{M}, \mathcal{V} \vdash \texttt{CAS}(\ell, v_1, v_2, o_f, o_r, o_w) \xrightarrow{\langle \text{Read}, \ell, v_r, o_f \rangle} 0}$$

OE-CAS-SUC
$$\frac{\begin{array}{l}\texttt{rlx} \sqsubseteq o_f \\ \texttt{rlx} \sqsubseteq o_r \\ \texttt{rlx} \sqsubseteq o_w\end{array} \quad (\forall \omega \in \text{Readable}(\ell, \mathcal{M}, \mathcal{V}).\, \exists v'.\, \omega \equiv v' \wedge \vdash v_1 =^? v') \quad \mathcal{M} \vdash v_1 = v_r}{\mathcal{M}, \mathcal{V} \vdash \texttt{CAS}(\ell, v_1, v_2, o_f, o_r, o_w) \xrightarrow{\langle \text{Update}, \ell, v_r, v_2, o_r, o_w \rangle} 1}$$

OE-FENCE
$$\mathcal{M}, \mathcal{V} \vdash \texttt{fence}_o \xrightarrow{\langle \text{Fence}, o \rangle} \text{☣}$$

OE-CASE
$$\mathcal{M}, \mathcal{V} \vdash \texttt{case}\, i\, \texttt{of}\, (\overline{e}) \to \overline{e}_i$$

OE-APP
$$\mathcal{M}, \mathcal{V} \vdash (\texttt{rec}\, f(\overline{x}) := e)(\overline{v}) \to e[\texttt{rec}\, f(\overline{x}) := e/f, \overline{v}/\overline{x}]$$

OE-FORK
$$\mathcal{M}, \mathcal{V} \vdash \texttt{fork}\, \{\, e \,\} \to \text{☣}, e$$

Figure 5: Expression semantics.

$\boxed{\mathcal{M}, \mathcal{N}, V \vdash \text{RaceFree}(\varepsilon)}$

DRF-READ-NA
$$\frac{\forall t \in \text{dom}(\mathcal{M}(\ell)).\, t \leq cur(\ell).\mathsf{w} \qquad \mathcal{N}(\ell).\mathsf{aw} \sqsubseteq cur(\ell).\mathsf{aw}}{\mathcal{M}, \mathcal{N}, (rel, frel, cur, acq) \vdash \text{RaceFree}(\langle \text{Read}, \ell, v, \mathtt{na} \rangle)}$$

DRF-WRITE-NA
$$\frac{\mathcal{N}(\ell).\mathsf{aw} \sqsubseteq cur(\ell).\mathsf{aw} \qquad \mathcal{N}(\ell).\mathsf{nr} \sqsubseteq cur(\ell).\mathsf{nr} \qquad \mathcal{N}(\ell).\mathsf{ar} \sqsubseteq cur(\ell).\mathsf{ar} \qquad \forall t \in \text{dom}(\mathcal{M}(\ell)).\, t \leq cur(\ell).\mathsf{w} < t_w}{\mathcal{M}, \mathcal{N}, (rel, frel, cur, acq) \vdash \text{RaceFree}(\langle \text{Write}, \ell, v, \mathtt{na} \rangle)}$$

DRF-READ-AT
$$\frac{\mathtt{rlx} \sqsubseteq o \qquad \mathcal{N}(\ell).\mathsf{w} \leq cur(\ell).\mathsf{w}}{\mathcal{M}, \mathcal{N}, (rel, frel, cur, acq) \vdash \text{RaceFree}(\langle \text{Read}, \ell, v, o \rangle)}$$

DRF-WRITE-AT
$$\frac{\mathtt{rlx} \sqsubseteq o \qquad \mathcal{N}(\ell).\mathsf{w} \leq cur(\ell).\mathsf{w} \qquad \mathcal{N}(\ell).\mathsf{nr} \sqsubseteq cur(\ell).\mathsf{nr}}{\mathcal{M}, \mathcal{N}, (rel, frel, cur, acq) \vdash \text{RaceFree}(\langle \text{Write}, \ell, v, o \rangle)}$$

DRF-UPDATE
$$\frac{\mathcal{M}, \mathcal{N}, V \vdash \text{RaceFree}(\langle \text{Read}, \ell, v_r, o_r \rangle) \qquad \mathcal{M}, \mathcal{N}, V \vdash \text{RaceFree}(\langle \text{Write}, \ell, v_w, o_w \rangle)}{\mathcal{M}, \mathcal{N}, V \vdash \text{RaceFree}(\langle \text{Update}, \ell, v_r, v_w, o_r, o_w \rangle)}$$

DRF-ALLOC
$$\mathcal{M}, \mathcal{N}, V \vdash \text{RaceFree}(\langle \text{Alloc}, \ell, n \rangle)$$

DRF-DEALLOC
$$\frac{\forall i \in [<n], t' \in \text{dom}(\mathcal{M}(\ell + i)).\, t' \leq cur(\ell).\mathsf{w} \qquad \forall i \in [<n].\, \mathcal{N}(\ell + i).\mathsf{aw} \sqsubseteq cur(\ell + i).\mathsf{aw} \qquad \forall i \in [<n].\, \mathcal{N}(\ell + i).\mathsf{nr} \sqsubseteq cur(\ell + i).\mathsf{nr} \qquad \forall i \in [<n].\, \mathcal{N}(\ell + i).\mathsf{ar} \sqsubseteq cur(\ell + i).\mathsf{ar}}{\mathcal{M}, \mathcal{N}, (rel, frel, cur, acq) \vdash \text{RaceFree}(\langle \text{Dealloc}, \ell, n \rangle)}$$

Figure 6: Data-race-free (DRF) pre condition, detailing the exact requirements on the local and global race detector state for any particular memory event.

$$\boxed{\mathcal{N} \xrightarrow{\varepsilon,t,m^*} \mathcal{N}'}$$

$$\text{DRF-Post-read-na}\quad \frac{r \notin \mathcal{N}(\ell).\mathsf{nr} \qquad \mathcal{N}' = \mathcal{N}\left[\ell \leftarrow \{\mathcal{N}(\ell)\ \mathtt{with}\ \mathsf{nr} := \mathcal{N}(\ell).\mathsf{nr} \cup \{r\}\}\right]}{\mathcal{N} \xrightarrow{\langle \text{Read},\ell,v,\mathtt{na}\rangle,r,[]} \mathcal{N}'}$$

$$\text{DRF-Post-write-na}\quad \frac{\mathcal{N}' = \mathcal{N}\left[\ell \leftarrow \{\mathcal{N}(\ell)\ \mathtt{with}\ \mathsf{w} := m.\mathsf{ts}\}\right]}{\mathcal{N} \xrightarrow{\langle \text{Write},\ell,v,\mathtt{na}\rangle,\bot,[m]} \mathcal{N}'}$$

$$\text{DRF-Post-read-at}\quad \frac{\mathtt{rlx} \sqsubseteq o \qquad r \notin \mathcal{N}(\ell).\mathsf{ar} \qquad \mathcal{N}' = \mathcal{N}\left[\ell \leftarrow \{\mathcal{N}(\ell)\ \mathtt{with}\ \mathsf{ar} := \mathcal{N}(\ell).\mathsf{ar} \cup \{r\}\}\right]}{\mathcal{N} \xrightarrow{\langle \text{Read},\ell,v,o\rangle,r,[]} \mathcal{N}}$$

$$\text{DRF-Post-write-at}\quad \frac{\mathtt{rlx} \sqsubseteq o \qquad \mathcal{N}' = \mathcal{N}\left[\ell \leftarrow \{\mathcal{N}(\ell)\ \mathtt{with}\ \mathsf{aw} := \mathcal{N}(\ell).\mathsf{aw} \cup \{m.\mathsf{ts}\}\}\right]}{\mathcal{N} \xrightarrow{\langle \text{Write},\ell,v,o\rangle,\bot,[m]} \mathcal{N}'}$$

$$\text{DRF-Post-update}\quad \frac{r \notin \mathcal{N}(\ell).\mathsf{ar} \qquad \mathcal{N}' = \mathcal{N}\left[\ell \leftarrow \{\mathcal{N}(\ell)\ \mathtt{with}\ \mathsf{ar} := \mathcal{N}(\ell).\mathsf{ar} \cup \{r\}\}\right] \qquad \mathcal{N}' = \mathcal{N}\left[\ell \leftarrow \{\mathcal{N}(\ell)\ \mathtt{with}\ \mathsf{aw} := \mathcal{N}(\ell).\mathsf{aw} \cup \{m.\mathsf{ts}\}\}\right]}{\mathcal{N} \xrightarrow{\langle \text{Update},\ell,v_r,v_w,o_r,o_w\rangle,r,[m]} \mathcal{N}'}$$

$$\text{DRF-Post-alloc}\quad \frac{\mathcal{N}' = \mathcal{N}\left[\ell + i \leftarrow \{\mathsf{w} := m_i.\mathsf{ts}, \mathsf{aw} := \emptyset, \mathsf{nr} := \emptyset, \mathsf{ar} := \emptyset\} \mid i \in [<n]\right]}{\mathcal{N} \xrightarrow{\langle \text{Alloc},\ell,n\rangle,\bot,[m_0 \ldots m_{n-1}]} \mathcal{N}'}$$

$$\text{DRF-Post-dealloc}\quad \frac{\mathcal{N}' = \mathcal{N}\left[\ell + i \leftarrow \{\mathcal{N}(\ell + i)\ \mathtt{with}\ \mathsf{w} := m_i.\mathsf{ts}\}) \mid i \in [<n]\right]}{\mathcal{N} \xrightarrow{\langle \text{Dealloc},\ell,n\rangle,\bot,[m_0 \ldots m_{n-1}]} \mathcal{N}'}$$

Figure 7: Data-race-free (DRF) post condition, detailing the change to the global race detector state on a per-event basis.

OM-READ-HELPER

$$\frac{\begin{array}{c} cur(\ell).\mathsf{w} \leq t \qquad R(\ell) \leq t \\ V = [\ell \leftarrow \{\mathsf{w} := t, \mathsf{aw} := \emptyset, \mathsf{nr} := \mathsf{if}\ o = \mathtt{na}\ \mathsf{then}\ \{r\}\ \mathsf{else}\ \emptyset, \mathsf{ar} := \mathsf{if}\ o \sqsubseteq \mathtt{rlx}\ \mathsf{then}\ \{r\}\ \mathsf{else}\ \emptyset\}] \\ cur' = \mathsf{if}\ \mathtt{acq} \sqsubseteq o\ \mathsf{then}\ cur \sqcup V \sqcup R\ \mathsf{else}\ cur \sqcup V \\ acq' = \mathsf{if}\ \mathtt{rlx} \sqsubseteq o\ \mathsf{then}\ acq \sqcup V \sqcup R\ \mathsf{else}\ acq \sqcup V \end{array}}{(rel, frel, cur, acq) \xrightarrow{\langle \mathrm{R}:o,\ell,t,R\rangle,r} (rel, frel, cur', acq')}$$

OM-WRITE-HELPER

$$\frac{\begin{array}{c} cur(\ell).\mathsf{w} < t \\ V = [\ell \leftarrow \{\mathsf{w} := t, \mathsf{aw} := \mathsf{if}\ \mathtt{rlx} \sqsubseteq o\ \mathsf{then}\ \{t\}\ \mathsf{else}\ \emptyset, \mathsf{nr} := \emptyset, \mathsf{ar} := \emptyset\}] \\ cur' = cur \sqcup V \qquad acq' = acq \sqcup V \\ V' = rel(\ell) \sqcup \mathsf{if}\ \mathtt{rel} \sqsubseteq o\ \mathsf{then}\ cur'\ \mathsf{else}\ V \qquad rel' = rel\,[\ell \leftarrow V'] \\ R_w = \mathsf{if}\ \mathtt{rlx} \sqsubseteq o\ \mathsf{then}\ V' \sqcup frel \sqcup R_r\ \mathsf{else}\ \bot \end{array}}{(rel, frel, cur, acq) \xrightarrow{\langle \mathrm{W}:o,\ell,t,R_r,R_w\rangle,\bot} (rel', frel, cur', acq')}$$

Figure 8: View-helper relations.

$$\boxed{\varsigma \mid \mathcal{V} \xrightarrow{\varepsilon} \varsigma' \mid \mathcal{V}'}$$

OM-ALLOC

$$\frac{\begin{array}{c}\ell = (i, n') \qquad \{i\} \times \mathbb{N} \# \operatorname{dom}(\mathcal{M}) \\ \mathcal{M}' = \mathcal{M}\,[\ell + m \leftarrow [t_m \leftarrow (\dagger, \bot)] \mid m \in [<n]] \\ \mathcal{V} \xrightarrow{\langle \mathrm{W}:\mathtt{na}, \ell+0, t_0, \bot, \bot\rangle} \cdots \xrightarrow{\langle \mathrm{W}:\mathtt{na}, \ell+m, t_m, \bot, \bot\rangle} \cdots \xrightarrow{\langle \mathrm{W}:\mathtt{na}, \ell+(n-1), t_{(n-1)}, \bot, \bot\rangle} \mathcal{V}' \\ ms = [(t_m, \dagger, \bot) \mid m \in [<n]]\end{array}}{\mathcal{M} \mid \mathcal{V} \xrightarrow{\langle \mathrm{Alloc}, \ell, n\rangle, \bot, ms} \mathcal{M}' \mid \mathcal{V}'}$$

OM-FREE

$$\frac{\begin{array}{c}\ell = (i, n') \qquad \operatorname{dom}(\mathcal{M}) \cap \{i\} \times \mathbb{N} = \{i\} \times ([\geq n', <n' + n]) \\ \forall m \in [<n], t \in \operatorname{dom}(\mathcal{M}(\ell + m)).\, t \leq \mathcal{V}.\mathsf{cur}(\ell + m).\mathsf{w} < t_m \wedge \mathcal{M}(\ell + m)(t).\mathsf{val} \neq \Phi \\ \forall m \in [<n].\, \operatorname{dom}(\mathcal{M}(\ell + m)) \neq \varnothing \\ \mathcal{M}' = \mathcal{M}\,[\ell + m \leftarrow [t_m \leftarrow (\Phi, \bot)] \mid m \in [<n]] \\ \mathcal{V} \xrightarrow{\langle \mathrm{W}:\mathtt{na}, \ell+0, t_0, \bot, \bot\rangle} \cdots \xrightarrow{\langle \mathrm{W}:\mathtt{na}, \ell+m, t_m, \bot, \bot\rangle} \cdots \xrightarrow{\langle \mathrm{W}:\mathtt{na}, \ell+(n-1), t_{(n-1)}, \bot, \bot\rangle} \mathcal{V}' \\ ms = [(t_m, \dagger, \bot) \mid m \in [<n]]\end{array}}{\mathcal{M} \mid \mathcal{V} \xrightarrow{\langle \mathrm{Dealloc}, \ell, n\rangle, \bot, ms} \mathcal{M}' \mid \mathcal{V}'}$$

OM-READ

$$\frac{\ell \notin \operatorname{unalloc}(\mathcal{M}) \qquad \mathcal{M}(\ell)(t) = (v, R) \qquad \mathcal{V}'' \xrightarrow{\langle \mathrm{R}:o, \ell, t, R\rangle, r} \mathcal{V}'}{\mathcal{M} \mid \mathcal{V} \xrightarrow{\langle \mathrm{Read}, \ell, v, o\rangle, r, []} \mathcal{M} \mid \mathcal{V}'}$$

OM-WRITE

$$\frac{\ell \notin \operatorname{unalloc}(\mathcal{M}) \qquad t \notin \mathcal{M}(\ell) \qquad \mathcal{M}' = \mathcal{M}\,[\ell \leftarrow \mathcal{M}(\ell)\,[t \leftarrow (v, R)]] \qquad \mathcal{V} \xrightarrow{\langle \mathrm{W}:o, \ell, t, \bot, R\rangle} \mathcal{V}'}{\mathcal{M} \mid \mathcal{V} \xrightarrow{\langle \mathrm{Write}, \ell, v, o\rangle, [(t, v, R)]} \mathcal{M}' \mid \mathcal{V}'}$$

OM-UPDATE

$$\frac{\begin{array}{c}\ell \notin \operatorname{unalloc}(\mathcal{M}) \qquad \mathcal{M}(\ell)(t_r) = (v_r, R_r) \qquad t_w = t_r + 1 \qquad t_w \notin \mathcal{M}(\ell) \\ \mathcal{M}' = \mathcal{M}\,[\ell \leftarrow \mathcal{M}(\ell)\,[t_w \leftarrow (v_w, R_w)]] \\ \mathcal{V} \xrightarrow{\langle \mathrm{R}:o_r, \ell, t_r, R_r\rangle, r} \xrightarrow{\langle \mathrm{W}:o_w, \ell, t_w, R_r, R_w\rangle} \mathcal{V}'\end{array}}{\mathcal{M} \mid \mathcal{V} \xrightarrow{\langle \mathrm{Update}, \ell, v_r, v_w, o_r, o_w\rangle, r, [(t_w, v_w, R_w)]} \mathcal{M}' \mid \mathcal{V}'}$$

OM-ACQ-FENCE

$$\mathcal{M} \mid \mathcal{V} \xrightarrow{\langle \mathrm{Fence}, \mathtt{acq}\rangle} \mathcal{M} \mid (\mathcal{V}.\mathsf{rel}, \mathcal{V}.\mathsf{frel}, \mathcal{V}.\mathsf{acq}, \mathcal{V}.\mathsf{acq})$$

OM-REL-FENCE

$$\mathcal{M} \mid \mathcal{V} \xrightarrow{\langle \mathrm{Fence}, \mathtt{rel}\rangle} \mathcal{M} \mid ([\ell \leftarrow \mathcal{V}.\mathsf{cur} \mid \ell \in \operatorname{dom}(\mathcal{V}.\mathsf{rel})], \mathcal{V}.\mathsf{cur}, \mathcal{V}.\mathsf{cur}, \mathcal{V}.\mathsf{acq})$$

Figure 9: Machine semantics.

$\boxed{\varsigma \mid \mathcal{V} \mid e \xrightarrow{\varepsilon}{}^{\pi} \varsigma' \mid \mathcal{V}' \mid e'}$

CombRed-pure

$$\frac{\mathcal{M}, \mathcal{V} \vdash e \to e', es}{(\mathcal{M}, \mathcal{N}, V) \mid e \xRightarrow{\bot, es} (\mathcal{M}, \mathcal{N}, V) \mid e'}$$

CombRed-event

$$\frac{\begin{array}{c}\forall \varepsilon', \mathcal{M}'', \mathcal{V}'', e'', r', ms'. \mathcal{M}, \mathcal{V} \vdash e \to^{\varepsilon'} e', [] \wedge \mathcal{M} \mid \mathcal{V} \xrightarrow{\varepsilon, r', ms'} \mathcal{M}'' \mid \mathcal{V}'' \implies \mathcal{M}, \mathcal{N}, \mathcal{V} \vdash \mathrm{RaceFree}(\varepsilon') \\ \mathcal{M}, \mathcal{V} \vdash e^{\varepsilon} \to e', [] \qquad \mathcal{M} \mid V \xrightarrow{\varepsilon, r, ms} \mathcal{M}' \mid V' \qquad \mathcal{N} \xrightarrow{\varepsilon, r, ms} \mathcal{N}'\end{array}}{(\mathcal{M}, \mathcal{N}, V) \mid e \xRightarrow{\varepsilon, []} (\mathcal{M}, \mathcal{N}, V) \mid e'}$$

Figure 10: Combined machine and expression semantics

$\boxed{\varsigma \mid \mathcal{TS} \to \varsigma' \mid \mathcal{TS}'}$

$$\mathrm{ForkView}(\mathcal{V}) ::= (\emptyset, \emptyset, \mathcal{V}.\mathsf{cur}, \mathcal{V}.\mathsf{cur})$$

OT-step

$$\frac{\mathcal{TS}(\pi) = (e, V) \qquad (\mathcal{M}, \mathcal{N}, V) \mid e \xRightarrow{\varepsilon, [e_{f,0}, \ldots, e_{fn}]} (\mathcal{M}', \mathcal{N}', V') \mid e' \qquad \{\rho_0 \ldots \rho_n\} \cap \mathrm{dom}\mathcal{TS} = \emptyset}{(\mathcal{M}, \mathcal{N}) \mid \mathcal{TS} \to (\mathcal{M}', \mathcal{N}') \mid \mathcal{TS}\,[\pi \leftarrow (e', V')]\,[\rho_i \leftarrow (e_{f,i}, \mathrm{ForkView}(V')) \mid i \in [<n]]}$$

Figure 11: Threadpool semantics.

2 Correspondence of ORC11 to RC11

The memory model of ORC11 is modeled after Lahav et al. [2017] (referred to as "RC11" from now on) without SC accesses and SC fences. It is worth noting that the memory model of ORC11 is more conservative and declares more programs racy than RC11. To prove this, we show that any program that is racy under RC11 is also considered racy by ORC11. We make this claim more precise below.

The race detector in ORC11 (and the one in the intermediate OGS machine) is stronger, *i.e.*, detects more races, than RC11. In particular, ORC11 does not permit reducing a **CAS** expression with order **acq** in the presence of an unsynchronized non-atomic read *even* when the **CAS** itself synchronizes with the non-atomic read. In contrast, the self-synchronizing nature of **CAS** leads to RC11 accepting this particular behavior as non-racy.

To simplify the proof, we allow RC11 to take expression reduction steps that are disallowed in ORC11. In particular, the declarative semantics in RC11 may compare arbitrary values with each other, whereas ORC11 will get stuck in some of these cases (see Fig. 5). A potential theorem to prove would then be that ORC11 detects any RC11 race or gets stuck for other reasons. Fortunately, the race detector in ORC11 already models races as being stuck and so the theorem statement simply becomes: *Any program that is racy under RC11 will get stuck under ORC11* (see Theorem 1).

Definition 1 (Extended Order) The set of *extended orders ExtOrder* is defined by

$$o \in \mathit{ExtOrder} := \mathit{Order} \uplus \{\mathtt{relacq}\}.$$

Note that $\mathtt{relacq} \sqsupseteq o$ for any (extended) order o. We define $o.w$ and $o.r$ s.t.

$$o.w, o.r := \begin{cases} \mathtt{rel}, \mathtt{acq} & \text{if } o = \mathtt{relacq} \\ \mathtt{rel}, \mathtt{rlx} & \text{if } o = \mathtt{rel} \\ \mathtt{rlx}, \mathtt{acq} & \text{if } o = \mathtt{acq} \\ \mathtt{rlx}, \mathtt{rlx} & \text{if } o = \mathtt{rlx} \\ \mathtt{na}, \mathtt{na} & \text{if } o = \mathtt{na} \end{cases}$$

Definition 2 (Labels) The set of *labels*, *Label*, is defined by the following (tagged) union of events:

$$\begin{aligned} \gamma \in \mathit{Label} := \quad & \{\mathtt{R}^o(\ell, v) \mid o \in \mathit{Order}, \ell \in \mathit{Loc}, v \in \mathit{Val}\} \\ & \cup \{\mathtt{W}^o(\ell, v) \mid o \in \mathit{Order}, \ell \in \mathit{Loc}, v \in \mathit{Val}\} \\ & \cup \left\{\mathtt{U}^o(\ell, v_r, v_w) \mid o \in \mathit{ExtOrder}, \ell \in \mathit{Loc}, v_r \in \mathrm{codom}(\vdash \cdot =^? \cdot), v_w \in \mathit{Val}\right\} \\ & \cup \{\mathtt{F}^o \mid o \in \{\mathtt{rel}, \mathtt{acq}\}\} \\ & \cup \{\mathtt{Fork}^\rho \mid \rho \in \mathit{Thread}\} \end{aligned}$$

We write $\gamma \sim \varepsilon$ when γ corresponds a memory event ε (mapping all labels except `Fork` to their corresponding counterparts in *MemEvent*).

2.1 Executions

An execution G is defined by:

1. a finite set of events $\mathrm{E} \subseteq \mathbb{N}$. with events $\mathrm{E} \supseteq \mathrm{E}_0 := \{a_0^\ell \mid \ell \in \mathcal{L}\}$.
2. a labelling function $\mathsf{lab} \in \mathrm{E} \to \mathit{Label}$, with projections $\mathsf{typ}, \mathsf{mod}, \mathsf{loc}, \mathrm{val}_r, \mathrm{val}_w$ where defined.
3. a function tid assigning a thread identifier to every event in E. We write E^π to denote the events in E with $\mathrm{tid}(a) = \pi$.
4. a strict partial order $\mathtt{sb} \subseteq \mathrm{E} \times \mathrm{E}$ which is total on E^π for every thread π. and which puts all events in E_0 before all other events.
5. a binary relation $\mathtt{rf} \subseteq [\mathtt{WU}]; =_{\mathrm{loc}}; [\mathtt{RU}]$ such that
   (a) $\forall \langle a, b\rangle \in \mathtt{rf}.\, \mathrm{val}_w(a) = \mathrm{val}_r(b)$
   (b) $\forall b, \langle a_1, b\rangle \in \mathtt{rf}, \langle a_2, b\rangle \in \mathtt{rf}.\, a_1 = a_2$.
6. a family of strict total orders $\{\mathtt{mo}_\ell\}_{\ell\in\mathcal{L}}$ and $\mathtt{mo} := \uplus_{\ell\in\mathcal{L}} \mathtt{mo}_\ell$.

2.1.1 Consistent Executions

Definition 3 (Completeness) An execution G is called *complete* if and only if for every $a \in \mathtt{R}$ we have $\mathrm{val}_r(a) = \text{☣} \vee \exists b \in \mathtt{W}_{\mathrm{loc}(a)}.\, \langle b, a\rangle \in \mathtt{rf}$. Note that this condition is weaker than in RC11 as it allows reads from uninitialized locations (signified by the value ☣).

Definition 4 (Auxiliary relations)

$$
\begin{aligned}
\mathtt{rb} &:= \mathtt{rf}^{-1}; \mathtt{mo} \\
\mathtt{eco} &:= (\mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb})^+ \\
\mathtt{rs} &:= [\mathtt{WU}];\, \mathtt{sb}|^?_{=\mathrm{loc}};\, [(\mathtt{WU})^{\sqsupseteq \mathtt{rlx}}];\, (\mathtt{rf}; [U])^* \\
\mathtt{asw} &:= [\mathtt{Fork}_\rho]; (\mathtt{sb}|^?_{\mathrm{tid}=\rho}); [\mathrm{E}^\rho] \\
\mathtt{sw} &:= \mathtt{asw} \cup \left([\mathrm{E}^{\sqsupseteq \mathtt{rel}}]; ([\mathtt{F}]; \mathtt{sb})^?; \mathtt{rs}; \mathtt{rf}; [(\mathtt{RU})^{\sqsupseteq \mathtt{rlx}}]; (\mathtt{sb}; [\mathtt{F}])^?; [\mathrm{E}^{\sqsupseteq \mathtt{acq}}]\right) \\
\mathtt{hb} &:= (\mathtt{sb} \cup \mathtt{sw})^+
\end{aligned}
$$

Definition 5 (Consistency) An execution is called RC11-*consistent* (simply "consistent" from now on) if it is complete **and**

- $\mathtt{hb}; \mathtt{eco}^?$ is irreflexive (COHERENCE)
- $\mathtt{sb} \cup \mathtt{rf}$ is acyclic (NO-THIN-AIR)

This definition does not include RC11's SC axiom.

2.2 Declarative Semantics

The following definitions are taken from Kaiser et al. [2017] ("iGPS") and, if necessary, adapted to our setting. Below we define threadpool reduction that generates *traces*. Note that we circumvent checks (such as those for legal comparisons) in the expression reduction by providing existentially quantified memory $\mathcal{M}$ and local view V.

$$\textsc{Trace-Red-Silent}\ \frac{\mathcal{M}, \mathcal{V} \vdash \mathcal{TS}(\pi) \to e, []}{\mathcal{TS} \overset{\epsilon}{\Rightarrow}^{\pi} \mathcal{TS}[\pi \mapsto e]} \qquad \textsc{Trace-Red-Mem}\ \frac{\gamma \sim \varepsilon \qquad \mathcal{M}, \mathcal{V} \vdash \mathcal{TS}(\pi) \xrightarrow{\varepsilon} e, []}{\mathcal{TS} \overset{\gamma}{\Rightarrow}^{\pi} \mathcal{TS}[\pi \mapsto e]}$$

$$\textsc{Trace-Red-Fork}\ \frac{\mathcal{V}(\pi) = (e, V) \qquad \mathcal{M}, \mathcal{V} \vdash \mathcal{TS}(\pi) \to e', e_f \qquad \rho \notin \mathrm{dom}(\mathcal{TS})}{\mathcal{TS} \overset{\mathtt{Fork}_\rho}{\Longrightarrow}^{\pi} \mathcal{TS}[\pi \mapsto e'] \uplus [\rho \mapsto e_f]}$$

We write $\mathcal{TS} \Rightarrow^{\pi} \mathcal{TS}'$ if $\mathcal{TS} \overset{x}{\Rightarrow}^{\pi} \mathcal{TS}'$ for some transition label x; $\mathcal{TS} \overset{x}{\Rightarrow} \mathcal{TS}'$ if $\mathcal{TS} \overset{x}{\Rightarrow}^{\pi} \mathcal{TS}'$ for some thread identifier π; and $\mathcal{TS} \Rightarrow \mathcal{TS}'$ if $\mathcal{TS} \overset{x}{\Rightarrow}^{\pi} \mathcal{TS}'$ for some transition label x and thread identifier π. A threadpool is called *final* if $\mathcal{TS}(\pi) \in \mathit{Val}$ for every $\pi \in \mathrm{dom}(\mathcal{TS})$.

Definition 6 (Traces) A *trace* is a sequence of pairs $\langle \gamma_1, \pi_1 \rangle, \ldots, \langle \gamma_n, \pi_n \rangle$. We say that $tr = \langle \gamma_1, \pi_1 \rangle, \ldots, \langle \gamma_n, \pi_n \rangle$ is a trace of an expression e if

$$[0 \mapsto e] \overset{\epsilon}{\Rightarrow}^{*} \overset{\gamma_1}{\Longrightarrow}^{\pi_1} \overset{\epsilon}{\Rightarrow}^{*} \ldots \overset{\epsilon}{\Rightarrow}^{*} \overset{\gamma_n}{\Longrightarrow}^{\pi_n} \overset{\epsilon}{\Rightarrow}^{*} \mathcal{TS}$$

for some thread π and threadpool $\mathcal{TS}$. When $\mathcal{TS}$ is final, we call tr a *full* trace.

Definition 7 A trace $tr = \langle \gamma_1, \pi_1 \rangle, \ldots, \langle \gamma_n, \pi_n \rangle$ induces partial order on indices $\mathtt{sb}(tr)$, called *sequenced-before*, and a relation on indices $\mathtt{asw}(tr)$, called *additional-synchronized-with*. They are defined by:

$$\frac{i < j \qquad \pi_i = \pi_j}{\langle i, j \rangle \in \mathtt{sb}(tr)} \qquad \frac{\langle i, j \rangle \in \mathtt{sb}(tr) \qquad \langle j, k \rangle \in \mathtt{sb}(tr)}{\langle i, k \rangle \in \mathtt{sb}(tr)}$$

$$\frac{i < j \qquad \gamma_i = \mathtt{Fork}_{\pi_j}}{\langle i, j \rangle \in \mathtt{asw}(tr)}$$

Lemma 1 Let tr be a trace of an expression e. Then

- Any prefix of tr is also a trace of e.
- Any permutation tr' of tr with $\mathtt{sb}(tr') = \mathtt{sb}(tr)$ and $\mathtt{asw}(tr') = \mathtt{asw}(tr)$ is a trace of e.

Definition 8 An execution G *follows* a trace $tr = \langle \gamma_1, \pi_1 \rangle, \ldots, \langle \gamma_n, \pi_n \rangle$ if:

- $\mathrm{E} = \{a_1, \ldots, a_n\}$ such that $\mathrm{lab}(a_k) = \gamma_k$ and $\mathrm{tid}(a_k) = \pi_k$ for every $1 \leq k < n$
- $\mathtt{sb} = \{\langle a_i, a_j \rangle \mid \langle i, j \rangle \in \mathtt{sb}(tr)\}$.

We call G an execution of expression e if G follows some trace of e.

Definition 9 (Conflict) Two events a and b are called *conflicting* in an execution G if $a, b \in \mathrm{E}$, $\{\mathrm{typ}(a), \mathrm{typ}(b)\} \cap \{\mathtt{W}, \mathtt{U}\} \neq \emptyset$, $a \neq b$, and $\mathrm{loc}(a) = \mathrm{loc}(b)$.

Definition 10 (Races) A pair $\langle a, b \rangle$ is called a *race* in G if a and b are conflicting events in G, and $\langle a, b \rangle \notin \mathtt{hb} \cup \mathtt{hb}^{-1}$. An execution G is called *racy* if there is some race $\langle a, b \rangle$ in G with $\mathtt{na} \in \{\mathrm{mod}(a), \mathrm{mod}(b)\}$.

Definition 11 (Bugginess) An execution G is buggy if it is racy. An expression e is buggy if some consistent execution of e is buggy.

2.3 Operational Graph Semantics (OGS)

We now introduce an operationalized account of RC11 (*OGS*, short for Operational Graph Semantics), in which we build up executions step by step. This serves as an important stepping stone towards a our correspondence proof with ORC11.

Definition 12 (Execution Extension: Memory Accesses) We write $G' \in \mathrm{Add}(G, \pi, \rho, \gamma)$ if there exists an event a s.t.

- $G'.\mathrm{E} = G.\mathrm{E} \uplus \{a\}$, $G'.\mathrm{tid} = G.\mathrm{tid} \cup \{a \mapsto \rho\}$, $G'.\mathrm{lab} = G.\mathrm{lab} \cup \{a \mapsto \gamma\}$
- if $\rho \neq \pi$ then $\rho \notin \mathrm{codom}(G.\mathrm{tid})$
- $G'.\mathtt{sb} = (G.\mathtt{sb} \uplus (G.\mathrm{E}^{\rho} \times \{a\}))^{+}$
- $G'.\mathtt{rf} \supseteq G.\mathtt{rf}$
- $G'.\mathtt{mo} \supseteq G.\mathtt{mo}$ and if $\gamma = \mathtt{W}^{\mathtt{na}}(\_, \_)$ then a is $\mathtt{mo}$-maximal in G'

Definition 13 (Race Predicate) We define a predicate $\mathrm{Race}(G, \pi)$ which holds for all memory events that would cause a data race in execution G. Note that this race detector models exactly the rules implement in ORC11. Thus, it detects more races than RC11 *but* only in (potentially non-buggy) executions following buggy expressions.

$$\textsc{Race-I}\quad \frac{o \sqsupseteq \mathtt{rlx} \qquad \gamma \in (\mathtt{RU})^{o}_{\ell} \qquad \exists a \in \mathtt{W}^{\mathtt{na}}_{\ell}.\, \forall b \in \mathrm{E}^{\pi}.\, \langle a, b\rangle \notin \mathtt{hb}^{*}}{\gamma \in \mathrm{Race}(G, \pi)}$$

$$\textsc{Race-II}\quad \frac{\gamma = \mathtt{R}^{\mathtt{na}}(\ell, \_) \qquad \exists a \in (\mathtt{WU})_{\ell}.\, \forall b \in \mathrm{E}^{\pi}.\, \langle a, b\rangle \notin \mathtt{hb}^{*}}{\gamma \in \mathrm{Race}(G, \pi)}$$

$$\textsc{Race-III}\quad \frac{\gamma = \mathtt{W}^{\mathtt{na}}(\ell, \_) \qquad \exists a \in (\mathtt{RWU})_{\ell}.\, \forall b \in \mathrm{E}^{\pi}.\, \langle a, b\rangle \notin \mathtt{hb}^{*}}{\gamma \in \mathrm{Race}(G, \pi)}$$

$$\textsc{Race-IV}\quad \frac{o \sqsupseteq \mathtt{rlx} \qquad \gamma = \mathtt{W}^{o}_{\ell} \qquad \exists a \in (\mathtt{RW})^{\mathtt{na}}_{\ell}.\, \forall b \in \mathrm{E}^{\pi}.\, \langle a, b\rangle \notin \mathtt{hb}^{*}}{\gamma \in \mathrm{Race}(G, \pi)}$$

$$\textsc{Race-V}\quad \frac{\gamma = \mathtt{U}^{o}_{\ell} \qquad \exists a \in (\mathtt{RW})^{\mathtt{na}}_{\ell}.\, \forall b \in \mathrm{E}^{\pi}.\, \langle a, b\rangle \notin \mathtt{hb}^{*}}{\gamma \in \mathrm{Race}(G, \pi)}$$

Definition 14 (OGS Reductions)

$$\textsc{OGS-Memory-Step}\quad \frac{\begin{array}{c}\gamma \in \{\mathtt{R}^{o}(\ell, v), \mathtt{W}^{o}(\ell, v), \mathtt{F}^{o}\} \\ \gamma \notin \mathrm{Race}(G, \pi) \\ G' \in \mathrm{Add}(G, \pi, \pi, \gamma) \\ G' \text{ is consistent}\end{array}}{G \xrightarrow{\gamma}_{\pi} G'}$$

$$\textsc{OGS-Fork}\quad \frac{\begin{array}{c}G' \in \mathrm{Add}(G, \pi, \rho, \mathtt{Fork}_{\rho}) \\ G' \text{ is consistent}\end{array}}{G \xrightarrow{\mathtt{Fork}_{\rho}}_{\pi} G'}$$

$$\textsc{OGS-Race}\quad \frac{\gamma \in \mathrm{Race}(G, \pi)}{G \xrightarrow{\gamma}_{\pi} \bot_{\mathrm{race}}}$$

We define combined machine and expression semantics for OGS. We once again allow expression reductions to proceed independent of the current state, thus capturing more behaviors than those allowed by ORC11.

$$\textsc{OGS-CombRed-pure}\quad \frac{\mathcal{M}, \mathcal{V} \vdash e \to e', es}{G \mid e \xRightarrow{\bot, es} G \mid e'}$$

$$\textsc{OGS-CombRed-event}\quad \frac{\forall \varepsilon', \mathcal{M}'', \mathcal{V}'', e''.\, \mathcal{M}, \mathcal{V} \vdash e \to^{\varepsilon'} e', [] \implies \neg(G \xrightarrow{\varepsilon}{}^{\pi} \bot_{\text{race}}) \qquad \mathcal{M}, \mathcal{V} \vdash e \overset{\varepsilon}{\to} e', [] \qquad G \xrightarrow{\varepsilon}{}^{\pi} G'}{G \mid e \xRightarrow{\varepsilon, []}{}^{\pi} G' \mid e'}$$

$$\textsc{OGS-OT-step}\quad \frac{\mathcal{TS}(\pi) = e \qquad G \mid e \xRightarrow{\varepsilon, [e_{f,0}, \ldots, e_{f_n}]}{}^{\pi} G' \mid e' \qquad \{\rho_0 \ldots \rho_n\} \cap \mathrm{dom}\mathcal{TS} = \emptyset}{G \mid \mathcal{TS} \to G' \mid \mathcal{TS}\,[\pi \leftarrow (e', V')]\,[\rho_i \leftarrow e_{f,i} \mid i \in [<n]]}$$

We define G_0 to be an execution in which all locations are allocated with an initial value of 0.

Lemma 2 (Inclusion of Behaviors (I)) Let G be a non-buggy, consistent execution of expression e. Then there exists a trace $tr = \langle \varepsilon_1, \pi_1 \rangle \ldots \langle \varepsilon_n, \pi_n \rangle$ of e such that $G_0 \xrightarrow{\varepsilon_1}{}^{\pi_1} \ldots \xrightarrow{\varepsilon_n}{}^{\pi_n} G \vee \exists j \leq n.\, G_0 \xrightarrow{\varepsilon_1}{}^{\pi_1} \ldots \xrightarrow{\varepsilon_j}{}^{\pi_j} \bot_{\text{race}}$.

Proof. As G is consistent, we have that $\mathtt{sb} \cup \mathtt{rf}$ is acyclic. Let $a_1, \ldots, a_n$ be an enumeration of E that respects $(\mathtt{sb} \cup \mathtt{rf})^+$. For every $1 \leq i \leq n$, let $\pi_i := \mathrm{tid}(a_i)$, $\varepsilon_i = \mathrm{lab}(a_i)$, and $tr = \langle \varepsilon_1, \pi_1 \rangle \ldots \langle \varepsilon_n, \pi_n \rangle$. Adding events $a_1, \ldots, a_n$ one-by-one we can thus establish either $G_0 \xrightarrow{\varepsilon_1}{}^{\pi_1} \ldots \xrightarrow{\varepsilon_n}{}^{\pi_n} G$, or—if in any step $j \leq n$ the race predicate detects a spurious race—$G_0 \xrightarrow{\varepsilon_1}{}^{\pi_1} \ldots \xrightarrow{\varepsilon_j}{}^{\pi_j} \bot_{\text{race}}$. □

Lemma 3 (Inclusion of Behaviors (II)) Let e be a buggy expression. Then $G_0; [0 \mapsto e] \to^* \bot_{\text{race}}$.

Proof. We have that e is buggy and, thus, a consistent execution G which is buggy. Let $a_1, \ldots, a_n$ be an enumeration of E that respects $\mathtt{sb} \cup \mathtt{rf}$. Let k be the minimal index such that $G \cap \{a_1, \ldots, a_k\}$ is buggy, *i.e.*, racy.

We thus have that $G \cap \{a_1, \ldots, a_k\}$ is racy. Let $j < k$ be the minimal index such that $\mathrm{loc}(a_k) = \mathrm{loc}(a_j)$, $\langle a_k, a_j \rangle \notin \mathtt{hb} \cup \mathtt{hb}^{-1}$, and one of the following holds:

- $a_k \in (\mathtt{WU})^{\sqsupseteq \mathtt{rlx}} \wedge a_j \in \mathtt{R}^{\mathtt{na}} \vee a_k \in \mathtt{W}^{\mathtt{na}}$
- $a_j \in (\mathtt{WU})^{\sqsupseteq \mathtt{rlx}} \wedge a_k \in \mathtt{R}^{\mathtt{na}} \vee a_j \in \mathtt{W}^{\mathtt{na}}$

Note that we have $\mathrm{tid}(a_k) \neq \mathrm{tid}(a_j)$, as otherwise these events would be related by $G.\mathtt{sb}$, and, thus $G.\mathtt{hb}$.

1. $a_k \in \mathtt{W}^{\mathtt{na}}$. We define $B := \{a \in \mathrm{E} | \langle a, a_j \rangle \in G.\mathtt{hb} \vee \langle a, a_k \rangle \in G.\mathtt{hb}^*\}$ and $G' := G \cap B$. Note that G' is non-empty, consistent, and not buggy. By Lemma 2, we have that $G_0 \xrightarrow{\varepsilon_1}{}^{\pi_1} \ldots \xrightarrow{\varepsilon_n}{}^{\pi_n} G' \vee \exists j \leq n.\, G_0 \xrightarrow{\varepsilon_1}{}^{\pi_1} \ldots \xrightarrow{\varepsilon_j}{}^{\pi_j} \bot_{\text{race}}$ for some trace $\langle \varepsilon_1, \pi_1 \rangle, \ldots, \langle \varepsilon_n, \pi_n \rangle$ of e. In the latter case our proof is done. Otherwise we have $\langle a_k, a_j \rangle \notin G'.\mathtt{hb}$ and we show that $G' \xrightarrow{\mathrm{lab}(a_j)}{}^{\mathrm{tid}(a_j)} \bot_{\text{race}}$.

   By Definition 13 (using whichever case corresponds to $\mathrm{lab}(a_j)$), it suffices to show that $\langle a_k, b \rangle \notin G'.\mathtt{hb}^*$ for all $b \in \mathrm{E}^{\mathrm{tid}(a_j)}$. By way of contradiction, assume $b \in \mathrm{E}^{\mathrm{tid}(a_j)}$ and $\langle a_k, b \rangle \in G'.\mathtt{hb}^*$. By definition of G', we have $\langle b, a_j \rangle \in G.\mathtt{hb} \vee \langle b, a_k \rangle \in G.\mathtt{hb}^*$.

(a) $\langle b, a_j\rangle \in G.\mathtt{hb}$. By transitivity, we have $\langle a_k, a_j\rangle \in G.\mathtt{hb}$, which contradicts our assumption.

(b) $\langle b, a_k\rangle \in G.\mathtt{hb}^*$. From $\mathrm{tid}(a_k) \neq \mathrm{tid}(a_j)$ we have that $b \neq a_k$. Thus, $\langle b, a_k\rangle \in G.\mathtt{hb}$. By transitivity, we have $\langle b, b\rangle \in G.\mathtt{hb}$, which contradicts $\mathtt{hb}$'s irreflexivity.

As $\langle \varepsilon_1, \pi_1\rangle, \ldots, \langle \varepsilon_n, \pi_n\rangle, \langle \mathrm{lab}(a_j), \mathrm{tid}(a_j)\rangle$ is a valid trace for e, we have that $G_0; [0 \mapsto e] \to^* \bot_{\mathrm{race}}$.

2. $a_j \in \mathtt{W}^{\mathtt{na}}$ is symmetric to the case above.

3. $a_k \in (\mathtt{WU})^{\sqsupseteq \mathtt{rlx}} \wedge a_j \in \mathtt{R}^{\mathtt{na}}$. We define $B := \{a \in \mathrm{E} | \langle a, a_j\rangle \in G.\mathtt{hb} \vee \langle a, a_k\rangle \in G.\mathtt{hb}^*\}$ and $G' := G \cap B$. Note that G' is consistent and not buggy. By Lemma 2, we have that $G_0 \xrightarrow{\varepsilon_1}{}^{\pi_1} \ldots \xrightarrow{\varepsilon_n}{}^{\pi_n} G' \vee \exists j \leq n. G_0 \xrightarrow{\varepsilon_1}{}^{\pi_1} \ldots \xrightarrow{\varepsilon_j}{}^{\pi_j} \bot_{\mathrm{race}}$ for some trace $\langle \varepsilon_1, \pi_1\rangle, \ldots, \langle \varepsilon_n, \pi_n\rangle$ of e. In the latter case our proof is done. Otherwise we have $\langle a_k, a_j\rangle \notin G'.\mathtt{hb}$ and we show that $G' \xrightarrow{\mathrm{lab}(a_j)}{}^{\mathrm{tid}(a_j)} \bot_{\mathrm{race}}$.

By Definition 13, it suffices to show that $\langle a_k, b\rangle \notin G'.\mathtt{hb}^*$ for all $b \in \mathrm{E}^{\mathrm{tid}(a_j)}$. By way of contradiction, assume $b \in \mathrm{E}^{\mathrm{tid}(a_j)}$ and $\langle a_k, b\rangle \in G'.\mathtt{hb}^*$. By definition of G', we have $\langle b, a_j\rangle \in G.\mathtt{hb} \vee \langle b, a_k\rangle \in G.\mathtt{hb}^*$.

(a) $\langle b, a_j\rangle \in G.\mathtt{hb}$. By transitivity, we have $\langle a_k, a_j\rangle \in G.\mathtt{hb}$, which contradicts our assumption.

(b) $\langle b, a_k\rangle \in G.\mathtt{hb}^*$. From $\mathrm{tid}(a_k) \neq \mathrm{tid}(a_j)$ we have that $b \neq a_k$. Thus, $\langle b, a_k\rangle \in G.\mathtt{hb}$. By transitivity, we have $\langle b, b\rangle \in G'.\mathtt{hb}$, which contradicts $\mathtt{hb}$'s irreflexivity.

As $\langle \varepsilon_1, \pi_1\rangle, \ldots, \langle \varepsilon_n, \pi_n\rangle, \langle \mathrm{lab}(a_j), \mathrm{tid}(a_j)\rangle$ is a valid trace for e, we have that $G_0; [0 \mapsto e] \to^* \bot_{\mathrm{race}}$.

4. $a_j \in \mathtt{W}^{\sqsupseteq \mathtt{rlx}} \wedge a_k \in \mathtt{R}^{\mathtt{na}}$. This case is symmetric to the one above.

□

2.4 OGS to ORC11

Definition 15 We define auxiliary relations $\{\mathtt{auxrel}\}_\ell$, $\mathtt{auxfrel}$, and $\mathtt{auxacq}$.

$$\begin{aligned}\mathtt{auxrel}_\ell &:= \mathtt{hb}; [\mathtt{W}_\ell^{\mathtt{rel}}]\\ \mathtt{auxfrel} &:= \mathtt{hb}; [\mathtt{F}^{\mathtt{rel}}]\\ \mathtt{auxacq} &:= \mathtt{hb}; ([\mathrm{E}^{\mathtt{rel}}]; ([\mathrm{F}]; \mathtt{sb})^?; \mathtt{rs}; \mathtt{rf}; [\mathrm{R}^{\mathtt{rlx}}])^?\end{aligned}$$

Definition 16 (Timestamp Assignment) A *timestamp assignment* for an execution graph G is a function $ts : \mathtt{W} \to \mathit{Time}$, that satisfies $ts(a) < ts(b)$ whenever $\langle a, b\rangle \in \mathtt{mo}$. Given a timestamp assignment ts for G and an event $a \in \mathtt{W}$, the view of an event a in G according to ts, denoted $\mathit{view}(a, G, ts)$, and the message induced by a in G according to ts, denoted $\mathit{msg}(a, G, ts)$, are given by:

$$\mathit{view}(a, G, ts) = \begin{cases} \lambda\ell.\ \max\{ts(b) \mid b \in \mathtt{W}_\ell, \langle b, a\rangle \in \mathtt{hb}^*\} & \text{if } \mathrm{mod}(a) = \mathtt{rel}\\ \lambda\ell.\ \max\{ts(b) \mid b \in \mathtt{W}_\ell, \langle b, a\rangle \in (\mathtt{auxfrel} \cup \mathtt{auxrel}_\ell)^?; \mathtt{hb}^*\} & \text{if } \mathrm{mod}(a) = \mathtt{rlx}\\ \bot & \text{otherwise}\end{cases}$$

$$\mathit{msg}(a, G, ts) = (\mathrm{val}_w(a), \mathit{view}(a, G, ts))$$

In these definitions, we take $\bot$ to be the maximum of an empty set.

Definition 17 (Event Injection) Let G be an execution and $a \in \mathrm{E}$. We define an injection into natural numbers, written $\mathrm{Inj}(G, a)$, as follows.

$$\mathrm{Inj}(G, a) := \mathrm{prime}(\mathrm{tid}(a))^{|\{b \mid \langle b,a\rangle \in \mathtt{sb}\}|}$$

where $\mathrm{prime}(n)$ is the n^{th} prime number. Note that Inj is injective and that performing a machine step $G \to G'$ implies $\mathrm{Inj}(G', a) = \mathrm{Inj}(G, a)$ for any $a \in G$.

We write $\mathrm{Inj}(G, X)$ for $\{\mathrm{Inj}(G, a) \mid a \in X\}$. We also write $a \in Y$ for $\mathrm{Inj}(G, a) \in Y$ if Y is defined as $\mathrm{Inj}(G, X)$ for some X. (Note that this implies $a \in X$.)

Definition 18 Let G be an execution and ts be a timestamp assignment for G. We define the physical state $(\mathcal{M}^{ts}_G, \mathcal{V}^{ts}_G, \mathcal{N}^{ts}_G)$ as follows.

- The memory is defined by $\mathcal{M}^{ts}_G := \lambda\ell.\, \lambda t.\, \begin{cases} msg(a, G, ts) & \text{if } \exists a.\, t = ts(a) \\ \bot & \text{otherwise} \end{cases}$
- The thread view $\mathcal{V}^{ts}_G$ is defined by

$$\begin{aligned}
\mathsf{ThEvs}(X, S, R) :=\ & \{a \in S \mid \exists b \in X.\, \langle a, b\rangle \in R^*\} \\
t_{\max}(X, S, R) :=\ & \max\{ts(a) \mid a \in \mathsf{ThEvs}(X, S, R)\} \\
V(X, R) :=\ & \lambda\ell.\, \{\ \mathsf{w} := t_{\max}(X, \mathtt{W}_\ell, R), \\
& \qquad \mathsf{aw} := \left\{ts(a) \mid a \in \mathsf{ThEvs}(X, \mathtt{W}^{\sqsupseteq \mathtt{rlx}}_\ell, R)\right\}, \\
& \qquad \mathsf{nr} := \mathrm{Inj}(G, \mathsf{ThEvs}(X, \mathtt{R}^{\mathtt{na}}_\ell, R)), \\
& \qquad \mathsf{ar} := \mathrm{Inj}(G, \mathsf{ThEvs}(X, \mathtt{R}^{\sqsupseteq \mathtt{rlx}}_\ell, R)) \\
& \ \} \\
\mathcal{V}^{ts}_G(\pi) :=\ & \{\mathsf{rel} := \lambda\ell'.\, V(\mathrm{E}^\pi, \mathtt{auxrel}_{\ell'}), \\
& \ \mathsf{frel} := V(\mathrm{E}^\pi, \mathtt{auxfrel}), \\
& \ \mathsf{cur} := V(\mathrm{E}^\pi, \mathtt{hb}), \\
& \ \mathsf{acq} := V(\mathrm{E}^\pi, \mathtt{auxacq}) \\
& \}
\end{aligned}$$

- The global race detector state $\mathcal{N}^{ts}_G$ is defined by

$$\begin{aligned}
\mathcal{N}^{ts}_G := \lambda\ell.\, \{ & \\
& \mathsf{w} := t_{\max}(\mathrm{E}, \mathtt{W}^{\mathtt{na}}_\ell, (=)), \\
& \mathsf{aw} := \left\{ts(a) \mid a \in \mathtt{W}^{\sqsupseteq \mathtt{rlx}}_\ell\right\}, \\
& \mathsf{nr} := \mathrm{Inj}(G, \mathtt{R}^{\mathtt{na}}_\ell), \\
& \mathsf{ar} := \mathrm{Inj}(G, \mathtt{R}^{\sqsupseteq \mathtt{rlx}}_\ell) \\
& \}
\end{aligned}$$

In these definitions, we take $\bot$ to be the maximum of an empty set.

We say that G *relates* to a physical state $(\mathcal{M},\mathcal{V},\mathcal{N})$, denoted $G \sim (\mathcal{M},\mathcal{V},\mathcal{N})$, if and only if $(\mathcal{M}^{ts}_G,\mathcal{V}^{ts}_G,\mathcal{N}^{ts}) = (\mathcal{M},\mathcal{V},\mathcal{N})$.

Definition 19 In the following, we lift ORC11's machine semantics to thread views such that

$$(\mathcal{M},\mathcal{N},\mathcal{V}) \xrightarrow{\varepsilon}{}^{\pi} (\mathcal{M},\mathcal{N}',\mathcal{V}') := (\mathcal{M},\mathcal{N}) \mid \mathcal{V}(\pi) \xrightarrow{\varepsilon} (\mathcal{M}',\mathcal{N}') \mid V' \wedge \mathcal{V}' = \mathcal{V}\,[\pi \leftarrow V']$$

Lemma 4 Suppose $G \xrightarrow{\gamma}{}^{\pi} G'$, $\gamma \sim \varepsilon$ and let ts' be a timestamp assignment for G'. Then $ts\,|_{G.\mathtt{W}}$ is a timestamp assignment for G and $(\mathcal{M}^{ts}_G,\mathcal{V}^{ts}_G,\mathcal{N}^{ts}_G) \xrightarrow{\varepsilon}{}^{\pi} (\mathcal{M}^{ts'}_{G'},\mathcal{V}^{ts'}_{G'},\mathcal{N}^{ts'}_{G'})$.

Lemma 5 (Inclusion of Behaviors (I)) Suppose $G \xrightarrow{\gamma_1}{}^{\pi_1} \ldots \xrightarrow{\gamma_n}{}^{\pi_n} G_n$ and $G \sim (\mathcal{M}_1,\mathcal{V}_1,\ \mathcal{N}_1)$. Then either

- there exist $\varepsilon_1 \ldots \varepsilon_n$, $(\mathcal{M}_2,\mathcal{N}_2,\mathcal{V}_2)\ G_2 \ldots (\mathcal{M}_n,\mathcal{N}_n,\mathcal{V}_n)\ G_n$ such that $\xrightarrow{\pi_1}\ [\varepsilon_1] \ldots\ldots \xrightarrow{\varepsilon_n}{}^{\pi_n} (\mathcal{M}_n,\mathcal{N}_n,\mathcal{V}_n)$.
- or there exist $j < n$, $\varepsilon_1 \ldots \varepsilon_{j+1}$, $(\mathcal{M}_2,\mathcal{N}_2,\mathcal{V}_2)\ G_2 \ldots (\mathcal{M}_j,\mathcal{N}_j,\mathcal{V}_j)\ G_j$ such that $\xrightarrow{\pi_1}\ [\varepsilon_1] \ldots\ldots \xrightarrow{\varepsilon_j}{}^{\pi_j} (\mathcal{M}_{j-1},\mathcal{N}_j,\mathcal{V}_j) \wedge \neg \left((\mathcal{M}_j,\mathcal{N}_j,\mathcal{V}_j) \xrightarrow{\varepsilon_{j+1}}{}^{\pi_{j+1}} \_\right)$.

Lemma 6 (Inclusion of Behaviors (II)) Let G be a consistent execution that is not buggy, $G \xrightarrow{\gamma}{}^{\pi} \bot_{\text{race}}$, $G \sim (\mathcal{M},\mathcal{V},\ \mathcal{N})$, and $\gamma \sim \varepsilon$. Then $\neg \left((\mathcal{M},\mathcal{N}) \mid \mathcal{V}(\pi) \xrightarrow{\varepsilon} \_\right)$

Proof. Let ts be the timestamp assignment implied by $G \sim (\mathcal{M},\mathcal{V},\mathcal{N})$. We consider the following cases.

1. $\gamma \in \{\mathtt{R}^o(\ell,\_), \mathtt{U}^o(\ell,\_,\_)\} \wedge o \sqsupseteq \mathtt{rlx} \wedge \exists a \in \mathtt{W}^{\mathtt{na}}_\ell.\, \forall b \in \mathrm{E}^\pi.\, \langle a,b\rangle \notin \mathtt{hb}^*$.

   We show $\neg\,(\mathcal{M},\mathcal{N},\mathcal{V}(\pi) \vdash \text{RaceFree}(\langle \text{Read},\ell,\_,o\rangle))$. It suffices to show that $\mathcal{V}(\pi).\mathsf{cur}(\ell) < \mathcal{N}(\ell)$. Let $a_m \in \mathtt{W}^{\mathtt{na}}_\ell$ be the $\mathtt{mo}$-maximal non-atomic write event on ℓ (which implies $ts(a_m) \geq ts(a)$). Then $\mathcal{N}(\ell) = ts(a_m)$. It thus suffices to show that $\mathcal{V}(\pi).\mathsf{cur}(\ell) < ts(a_m)$. By way of contradiction, assume that $\mathcal{V}(\pi).\mathsf{cur}(\ell) \geq ts(a_m)$. Then, there exists $c \in \mathtt{W}_\ell$ and $b \in \mathrm{E}^\pi$ s.t. $\langle c,b\rangle \in \mathtt{hb}^* \wedge ts(c) \geq ts(a_m)$. From $\langle a,a_m\rangle \in \mathtt{mo}^*$, COHERENCE, and G being non-racy we have that $\langle a,a_m\rangle \in \mathtt{hb}^*$. As G is non-racy, we also have $c = a_m \vee \langle a_m,c\rangle \in \mathtt{hb} \vee \langle c,a_m\rangle \in \mathtt{hb}$.

   (a) $c = a_m$. We have $\langle a_m,b\rangle \in \mathtt{hb}^*$. Then, by transitivity, we have $\langle a,b\rangle \in \mathtt{hb}^*$ which contradicts our initial assumption.

   (b) $\langle a_m,c\rangle \in \mathtt{hb}$. By transitivity, we have $\langle a_m,b\rangle \in \mathtt{hb}^*$, and, thus, $\langle a,b\rangle \in \mathtt{hb}^*$. This contradicts our initial assumption.

   (c) $\langle c,a_m\rangle \in \mathtt{hb} \wedge c \neq a$. By COHERENCE, we have $\langle a_m,c\rangle \notin \mathtt{mo}$ and, thus, $\langle c,a_m\rangle \in \mathtt{mo}$. This contradicts $ts(c) \geq ts(a_m)$.

2. $\gamma = \mathtt{R}^{\mathtt{na}}(\ell,\_) \wedge \exists a \in (\mathtt{WU})_\ell.\, \forall b \in \mathrm{E}^\pi.\, \langle a,b\rangle \notin \mathtt{hb}^*$.

   We show $\neg\,(\mathcal{M},\mathcal{N},\mathcal{V}(\pi) \vdash \text{RaceFree}(\langle \text{Read},\ell,\_,\mathtt{na}\rangle))$. It suffices to show that there exists t', $(v',V') = \mathcal{M}(\ell)(t')$ s.t. $\mathcal{V}(\pi).\mathsf{cur}(\ell) < t' \vee \mathcal{N}(\ell).\mathsf{aw} \not\sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{aw}$.

   We choose $t' = ts(a)$ and $(v',V') := msg(a,G,ts)$. It suffices to show $\mathcal{V}(\pi).\mathsf{cur}(\ell) < ts(a) \vee \mathcal{N}(\ell).\mathsf{aw} \not\sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{aw}$. There exists $c \in \mathtt{W}_\ell$ and $b \in \mathrm{E}^\pi$ s.t. $\langle c,b\rangle \in \mathtt{hb}^*$ and $\mathcal{V}(\pi).\mathsf{cur}(\ell) = ts(c)$.

   We consider two cases:

(a) $\mathrm{mod}(a) = \mathtt{na}$. We show $ts(c) < ts(a)$. By way of contradiction, assume $ts(c) \geq ts(a)$. We have $c \neq a$ as otherwise $\langle a, b\rangle \in \mathtt{hb}^*$, contradicting our assumption. Thus we have $ts(c) > ts(a)$ and $\langle a, c\rangle \in \mathtt{mo}$. From G being non-racy, COHERENCE, and $\langle a, c\rangle \in \mathtt{mo}$ we have that $\langle a, c\rangle \in \mathtt{hb}$. By transitivity, $\langle a, b\rangle \in \mathtt{hb}^*$, which contradicts our assumption.

(b) $\mathrm{mod}(a) = \mathtt{rlx}$. We show $\mathcal{N}(\ell).\mathsf{aw} \not\sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{aw}$. By way of contradiction, assume that $\mathcal{N}(\ell).\mathsf{aw} \sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{aw}$. We have $a \in \mathcal{N}(\ell).\mathsf{aw}$ and, thus, $a \in \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{aw}$. Hence, there exists $b' \in \mathrm{E}^\pi$ s.t. $\langle a, b'\rangle \in \mathtt{hb}^*$, which contradicts our assumption.

3. $\gamma = \mathtt{W}^{\mathtt{na}}(\ell, v) \wedge \exists a \in (\mathtt{RWU})_\ell. \forall b \in \mathrm{E}^\pi. \langle a, b\rangle \notin \mathtt{hb}^*$.

We show that $\neg(\mathcal{M}, \mathcal{N}, \mathcal{V}(\pi) \vdash \mathrm{RaceFree}(\langle \mathrm{Write}, \ell, \_, \mathtt{na}\rangle))$. There exists $c \in \mathtt{W}_\ell$ and $b \in \mathrm{E}^\pi$ s.t. $\langle c, b\rangle \in \mathtt{hb}^* \wedge \mathcal{V}(\pi).\mathsf{cur}(\ell) = ts(c)$. We also have $a \neq c$ as that would imply $\langle a, b\rangle \in \mathtt{hb}^*$, contradicting our assumption.

We consider the following cases.

(a) $a \in \mathtt{W}^{\mathtt{na}}_\ell$. We show that there exists t', $(v', V') = \mathcal{M}(\ell)(t')$ s.t. $ts(c) < t'$. We choose $t' := ts(a)$ and $(v', V') := msg(a, G, ts)$. It suffices to show $ts(c) < ts(a)$. As G is non-racy, we have $\langle a, c\rangle \in \mathtt{hb} \vee \langle c, a\rangle \in \mathtt{hb}$. The former implies, by transitivity, that $\langle a, b\rangle \in \mathtt{hb}^*$, which would contradict our assumption. Thus, $\langle c, a\rangle \in \mathtt{hb}$. As $a \neq b$ we derive $\langle c, a\rangle \in \mathtt{mo}$ from COHERENCE and, thus, $ts(c) < ts(a)$.

(b) $a \in (\mathtt{WU})^{\sqsupseteq \mathtt{rlx}}_\ell$. We show that $\mathcal{N}(\ell).\mathsf{aw} \not\sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{aw}$. By way of contradiction, assume that $\mathcal{N}(\ell).\mathsf{aw} \sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{aw}$. We have $a \in \mathcal{N}(\ell).\mathsf{aw}$ and, thus, $a \in \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{aw}$. Hence, there exists $b' \in \mathrm{E}^\pi$ s.t. $\langle a, b'\rangle \in \mathtt{hb}^*$, which contradicts our assumption.

(c) $a \in \mathtt{R}_\ell$. We show that $\mathcal{N}(\ell).\mathsf{nr} \not\sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{nr} \vee \mathcal{N}(\ell).\mathsf{ar} \not\sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{ar}$. We have $a \in \mathcal{N}(\ell).\mathsf{nr} \vee a \in \mathcal{N}(\ell).\mathsf{ar}$. By way of contradiction, assume that $\mathcal{N}(\ell).\mathsf{nr} \sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{nr} \wedge \mathcal{N}(\ell).\mathsf{ar} \sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{ar}$. Then $a \in \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{nr} \cup \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{ar}$. Hence, there exists $b' \in \mathrm{E}^\pi$ s.t. $\langle a, b'\rangle \in \mathtt{hb}$. This contradicts our assumption.

4. $\gamma = \mathtt{W}^o(\ell, \_) \wedge o \sqsupseteq \mathtt{rlx} \wedge \exists a \in (\mathtt{RW})^{\mathtt{na}}_\ell. \forall b \in \mathrm{E}^\pi. \langle a, b\rangle \notin \mathtt{hb}^*$.

We show $\neg(\mathcal{M}, \mathcal{N}, \mathcal{V}(\pi) \vdash \mathrm{RaceFree}(\langle \mathrm{Write}, \ell, \_, o\rangle))$. We consider $a \in \mathtt{R}^{\mathtt{na}}$ and $a \in \mathtt{W}^{\mathtt{na}}$ separately.

(a) $a \in \mathtt{R}^{\mathtt{na}}$. It suffices to show that $\mathcal{N}(\ell).\mathsf{nr} \not\sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{nr}$.
By way of contradiction, assume that $\mathcal{N}(\ell).\mathsf{nr} \sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{nr}$. We have $a \in \mathcal{N}(\ell).\mathsf{nr}$ and, thus, $a \in \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{nr}$. This implies that there exists $b' \in \mathrm{E}^\pi$ s.t. $\langle a, b'\rangle \in \mathtt{hb}^*$, which contradicts our assumption.

(b) $a \in \mathtt{W}^{\mathtt{na}}$. It suffices to show that $V(\pi).\mathsf{cur}(\ell) < \mathcal{N}(\ell)$.
Let $a_m \in \mathtt{W}^{\mathtt{na}}_\ell$ be the $\mathtt{mo}$-maximal non-atomic write event on ℓ (which implies $ts(a_m) \geq ts(a)$). Then $\mathcal{N}(\ell) = ts(a_m)$. It thus suffices to show that $\mathcal{V}(\pi).\mathsf{cur}(\ell) < ts(a_m)$. By way of contradiction, assume that $\mathcal{V}(\pi).\mathsf{cur}(\ell) \geq ts(a_m)$. Then, there exists $c \in \mathtt{W}_\ell$ and $b \in \mathrm{E}^\pi$ s.t. $\langle c, b\rangle \in \mathtt{hb}^* \wedge ts(c) \geq ts(a_m)$. From $\langle a, a_m\rangle \in \mathtt{mo}^*$, COHERENCE, and G being non-racy we have that $\langle a, a_m\rangle \in \mathtt{hb}^*$. As G is non-racy, we also have $c = a_m \vee \langle a_m, c\rangle \in \mathtt{hb} \vee \langle c, a_m\rangle \in \mathtt{hb}$.

i. $c = a_m$. We have $\langle a_m, b\rangle \in \mathtt{hb}^*$. Then, by transitivity, we have $\langle a, b\rangle \in \mathtt{hb}^*$ which contradicts our initial assumption.

ii. $\langle a_m, c\rangle \in \mathtt{hb}$. By transitivity, we have $\langle a_m, b\rangle \in \mathtt{hb}^*$, and, thus, $\langle a, b\rangle \in \mathtt{hb}^*$. This contradicts our initial assumption.

iii. $\langle c, a_m\rangle \in \mathtt{hb} \wedge c \neq a$. By COHERENCE, we have $\langle a_m, c\rangle \notin \mathtt{mo}$ and, thus, $\langle c, a_m\rangle \in \mathtt{mo}$. This contradicts $ts(c) \geq ts(a_m)$.

5. $\gamma = \mathtt{U}^o(\ell, \_, \_) \wedge \exists a \in (\mathtt{RW})^{\mathtt{na}}_\ell. \forall b \in \mathrm{E}^\pi. \langle a, b\rangle \notin \mathtt{hb}^*$.

We show that performing the "write" part of the update event leads to a race in ORC11, *i.e.*, $\neg(\mathcal{M}, \mathcal{N}, \mathcal{V}(\pi) \vdash \mathrm{RaceFree}(\langle \mathrm{Write}, \ell, \_, o.w\rangle))$. We consider $a \in \mathtt{R}^{\mathtt{na}}$ and $a \in \mathtt{W}^{\mathtt{na}}$ separately.

(a) $a \in \mathtt{R}^{\mathtt{na}}$. We show that $\mathcal{N}(\ell).\mathsf{nr} \not\sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{nr}$.

By way of contradiction, assume that $\mathcal{N}(\ell).\mathsf{nr} \sqsubseteq \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{nr}$. We have $a \in \mathcal{N}(\ell).\mathsf{nr}$ and, thus, $a \in \mathcal{V}(\pi).\mathsf{cur}(\ell).\mathsf{nr}$. This implies that there exists $b' \in \mathrm{E}^\pi$ s.t. $\langle a, b'\rangle \in \mathtt{hb}^*$, which contradicts our assumption.

(b) $a \in \mathtt{W}^{\mathtt{na}}$. It suffices to show that $V(\pi).\mathsf{cur}(\ell) < \mathcal{N}(\ell)$.

Let $a_m \in \mathtt{W}^{\mathtt{na}}_\ell$ be the $\mathtt{mo}$-maximal non-atomic write event on ℓ (which implies $ts(a_m) \geq ts(a)$). Then $\mathcal{N}(\ell) = ts(a_m)$. It thus suffices to show that $\mathcal{V}(\pi).\mathsf{cur}(\ell) < ts(a_m)$. By way of contradiction, assume that $\mathcal{V}(\pi).\mathsf{cur}(\ell) \geq ts(a_m)$. Then, there exists $c \in \mathtt{W}_\ell$ and $b \in \mathrm{E}^\pi$ s.t. $\langle c, b\rangle \in \mathtt{hb}^* \wedge ts(c) \geq ts(a_m)$. From $\langle a, a_m\rangle \in \mathtt{mo}^*$, COHERENCE, and G being non-racy we have that $\langle a, a_m\rangle \in \mathtt{hb}^*$. As G is non-racy, we also have $c = a_m \vee \langle a_m, c\rangle \in \mathtt{hb} \vee \langle c, a_m\rangle \in \mathtt{hb}$.

i. $c = a_m$. We have $\langle a_m, b\rangle \in \mathtt{hb}^*$. Then, by transitivity, we have $\langle a, b\rangle \in \mathtt{hb}^*$ which contradicts our initial assumption.

ii. $\langle a_m, c\rangle \in \mathtt{hb}$. By transitivity, we have $\langle a_m, b\rangle \in \mathtt{hb}^*$, and, thus, $\langle a, b\rangle \in \mathtt{hb}^*$. This contradicts our initial assumption.

iii. $\langle c, a_m\rangle \in \mathtt{hb} \wedge c \neq a$. By COHERENCE, we have $\langle a_m, c\rangle \notin \mathtt{mo}$ and, thus, $\langle c, a_m\rangle \in \mathtt{mo}$. This contradicts $ts(c) \geq ts(a_m)$.

□

Lemma 7 Suppose $G \xrightarrow{\gamma_1}{}^{\pi_1} \ldots \xrightarrow{\gamma_n}{}^{\pi_n} \bot_{\mathrm{race}}$ and $G \sim (\mathcal{M}_1, \mathcal{V}_1, \mathcal{N}_1)$. Then there exist $0 \leq j < n$, $\varepsilon_1 \ldots \varepsilon_{j+1}$, $(\mathcal{M}_2, \mathcal{N}_2, \mathcal{V}_2)\ G_2 \ldots (\mathcal{M}_j, \mathcal{N}_j, \mathcal{V}_j)\ G_j$ such that $\xrightarrow{\pi_1}\ [\varepsilon_1] \ldots\ldots \xrightarrow{\varepsilon_j}{}^{\pi_j}\ (\mathcal{M}_{j-1}, \mathcal{N}_j, \mathcal{V}_j) \wedge \neg\left((\mathcal{M}_j, \mathcal{N}_j, \mathcal{V}_j) \xrightarrow{\varepsilon_{j+1}}{}^{\pi_{j+1}} \_\right)$.

Proof. Follows from Lemma 5 and Lemma 6. □

Definition 20 (Initial State) We define the initial physical state $\mathcal{M}_0$, global race detector state $\mathcal{N}_0$ as well as an initial thread view V_0 as follows.

$$\mathcal{M}_0 := \lambda\ell.\, \lambda t.\, \begin{cases} (0, \bot) & \text{if } t = 0 \\ \bot & \text{otherwise} \end{cases}$$

$$\mathcal{N}_0 := \lambda\ell.\, 0$$

$$\mathrm{auxview} := \lambda\ell.\, \{\ \mathsf{w} := 0, \mathsf{aw} := \emptyset, \mathsf{nr} := \emptyset, \mathsf{ar} := \emptyset,\ \}$$

$$V_0 := \{\mathsf{rel} := \lambda\ell.\, \bot, \mathsf{frel} := \bot, \mathsf{cur} := \mathrm{auxview}, \mathsf{acq} := \mathrm{auxview},\}$$

Theorem 1 (ORC11: Racy Programs Get Stuck) Suppose e is buggy. Then $(\mathcal{M}_0, \mathcal{N}_0) \mid [0 \mapsto (e, V_0)] \to^* (\mathcal{M}', \mathcal{N}') \mid \mathcal{TS}'$ such that $\neg((\mathcal{M}', \mathcal{N}') \mid \_)$.

Proof. Follows from Lemma 3 and Lemma 7. □

3 Lifetime Logic for Views

The lifetime logic in RustBelt needs to be adapted to a logic like GPFSL where assertions depend on views.

3.1 Proof Rules

Splitting ownership in time. The lifetime logic adds a built-in notion of *lifetimes*, and the notion of "owning P borrowed for lifetime κ", written $\&^{\kappa}_{\mathbf{full}} P$.

The rule LftL-begin is used to create a new lifetime. At this point, we obtain the token $[\kappa]_1$ which asserts that *we own the lifetime* κ: We know that the lifetime is still running, and we can end it any time by applying the view shift we got. Now, it turns out that we may want multiple parties to be able to witness that κ is ongoing, so we need to be able to split this assertion: $[\kappa]_q$ denotes ownership of the fraction q of κ. Lifetimes can be *intersected* using the $\sqcap$ operator.

We also obtain an update to end the new lifetime again. This makes use of the "update that takes a step", defined as follows:

$$P \Rrightarrow\!\!\!\!\ast^{\mathcal{E}_2}_{\mathcal{E}_1} Q := P \mathrel{-\!\!*} {}^{\mathcal{E}_1}\!\Rrightarrow^{\mathcal{E}_2} \triangleright\, {}^{\mathcal{E}_2}\!\Rrightarrow^{\mathcal{E}_1} Q$$

The core operation of the lifetime logic is *borrowing* an assertion P at a given lifetime. Using LftL-borrow, P is split into ownership of P during the lifetime κ (the full borrow), and ownership when κ died (a view shift that lets us "inherit" P from κ). In some sense, we are *splitting ownership along the time axis*: The justification for the separating conjunction is the fact that a lifetime is never both ongoing and has already ended at the same time. Thus, the two parts that we split P into can be treated as disjoint resources: They govern the same part of the (logical and physical) state, but they do so at different points in time.

When a lifetime ends, full borrows at that lifetime are not worth anything any more, a fact that is witnessed by LftL-bor-fake.

Borrowed assertions can still be split and merged, as shown by LftL-bor-sep. To get access to a borrowed assertion, we use LftL-bor-acc-strong. The rule is quite a mouthful, so it is worth looking at the following simpler (derived) version:

$$\langle \&^{\kappa}_{\mathbf{full}} P * [\kappa]_q \Longleftrightarrow \triangleright P \rangle_{\mathcal{N}_{\mathrm{lft}}} \tag{1}$$

This lets us *open* full borrows ($\&^{\kappa}_{\mathbf{full}} P$) if we can prove that the lifetime is still ongoing, which we do by presenting any fraction of the lifetime token. We obtain $\triangleright P$, but lose access to that token for as long as the full borrow is open, which ensures that we do not end the lifetime while the full borrow is open. Once we re-established $\triangleright P$, we can *close* the full borrow again the get our token back.

The full rule LftL-bor-acc-strong actually lets us close not just with $\triangleright P$, but with any $\triangleright Q$ if we can show that Q entails P through a view shift. Furthermore, that view shift is only actually tun when the lifetime ends, which is witnessed by providing the appropriate token ($[\dagger\kappa]$).

Figure 12: Lifetime logic assertions and proof rules

Notation	Meaning	Timeless	Persistent
$[\kappa]_q$	Fraction q of lifetime token for κ: Witnessing that the lifetime is still ongoing	Yes	No
$[\dagger\kappa]$	Witness confirming that the lifetime κ is dead (*i.e.*, it has ended)	Yes	Yes
$\&^{\kappa}_{\mathbf{full}}\, P$	Ownership of the *full borrow* of P for κ	No	No
$\&^{\kappa}_{i}\, P$	There is an *indexed borrow* named i of P for κ	No	Yes
$[\mathrm{Bor} : i]$	Ownership of the indexed borrow i	Yes	No
$\&^{\kappa/0}_{\mathbf{at}}\, P$	Internal atomic persistent borrow of P for κ	No	Yes

Lifetimes. Lifetimes κ form a cancellable PCM with intersection as the operation ($\sqcap$) and unit ε.

$$\kappa \sqsubseteq \kappa' := \Box \forall q.\, \langle [\kappa]_q \Longleftrightarrow q'.\, [\kappa']_{q'} \rangle_{\mathcal{N}_{\mathrm{lft}}}$$

Lifetime creation and end.

LftL-begin
$\mathsf{True} \Rrightarrow_{\mathcal{N}_{\mathrm{lft}}} \exists \kappa.\, [\kappa]_1 * \Box\big([\kappa]_1 \mathrel{\Rrightarrow\!\!\!\!\!\!\ast}^{\mathcal{N}_{\mathrm{lft}}}_{\emptyset} [\dagger\kappa]\big)$

LftL-tok-fract
$[\kappa]_{q+q'} \Leftrightarrow [\kappa]_q * [\kappa]_{q'}$

LftL-tok-fract-obj
$[\kappa]_{q+q'} \Rrightarrow [\kappa]_q * \langle obj \rangle\, [\kappa]_{q'}$

LftL-tok-comp
$[\kappa \sqcap \kappa']_q \Leftrightarrow [\kappa]_q * [\kappa']_q$

LftL-tok-unit
$\mathsf{True} \Rrightarrow [\varepsilon]_q$

LftL-not-own-end
$[\kappa]_q * [\dagger\kappa] \Rrightarrow \mathsf{False}$

LftL-end-comp
$[\dagger\kappa \sqcap \kappa'] \Leftrightarrow [\dagger\kappa] \vee [\dagger\kappa']$

LftL-end-unit
$[\dagger\varepsilon] \Rrightarrow \mathsf{False}$

Creating full borrows and using them.

LftL-borrow
$\triangleright P \Rrightarrow_{\mathcal{N}_{\mathrm{lft}}} \&^{\kappa}_{\mathbf{full}}\, P * \big([\dagger\kappa] \mathrel{\Rrightarrow\!\!\!\!\!\!\ast}_{\mathcal{N}_{\mathrm{lft}}} \triangleright P\big)$

LftL-bor-sep
$\&^{\kappa}_{\mathbf{full}}(P * Q) \Longleftrightarrow_{\mathcal{N}_{\mathrm{lft}}} \&^{\kappa}_{\mathbf{full}}\, P * \&^{\kappa}_{\mathbf{full}}\, Q$

LftL-bor-fake
$\langle subj \rangle\, [\dagger\kappa] \Rrightarrow_{\mathcal{N}_{\mathrm{lft}}} \&^{\kappa}_{\mathbf{full}}\, P$

LftL-bor-acc-strong
$\&^{\kappa}_{\mathbf{full}}\, P * [\kappa]_q \Rrightarrow_{\mathcal{N}_{\mathrm{lft}}} \exists \kappa'.\, \kappa \sqsubseteq \kappa' * \triangleright P * \Big(\forall Q.\, \triangleright\big(\triangleright Q * \langle subj \rangle\, [\dagger\kappa'] \mathrel{\Rrightarrow\!\!\!\!\!\!\ast}_{\emptyset} \triangleright P\big) * \triangleright Q \mathrel{\Rrightarrow\!\!\!\!\!\!\ast}_{\mathcal{N}_{\mathrm{lft}}} \&^{\kappa'}_{\mathbf{full}}\, Q * [\kappa]_q\Big)$

LftL-bor-acc-atomic-strong
$\&^{\kappa}_{\mathbf{full}}\, P \;\; {}^{\mathcal{N}_{\mathrm{lft}}}\!\Rrightarrow^{\emptyset} \Big(\exists P' \kappa'.\, \kappa \sqsubseteq \kappa' * \triangleright(\lceil P' \rceil \wedge P) * \Big(\forall Q.\, \triangleright\big(\triangleright Q * \langle subj \rangle\, [\dagger\kappa'] \mathrel{\Rrightarrow\!\!\!\!\!\!\ast}_{\emptyset} \triangleright P\big) * \triangleright(\lceil P' \rceil \wedge Q) \;\; {}^{\emptyset}\!\Rrightarrow\!\!\!\!\!\!\ast^{\mathcal{N}_{\mathrm{lft}}} \&^{\kappa'}_{\mathbf{full}}\, Q\Big)\Big) \vee$
$\Big(\exists \kappa'.\, \kappa \sqsubseteq \kappa' * \langle subj \rangle\, [\dagger\kappa'] * {}^{\emptyset}\!\Rrightarrow^{\mathcal{N}_{\mathrm{lft}}} \mathsf{True}\Big)$

Figure 13: Lifetime logic assertions and proof rules, continued

Indexed borrows.

LftL-bor-idx
$$\&^{\kappa}_{\mathbf{full}}\, P \Leftrightarrow \exists i.\ \&^{\kappa}_{i}\, P * [\mathrm{Bor} : i]$$

LftL-idx-shorten
$$\frac{\kappa' \sqsubseteq \kappa}{\&^{\kappa}_{i}\, P \Rightarrow \&^{\kappa'}_{i}\, P}$$

LftL-idx-acc
$$\&^{\kappa}_{i}\, P(V_{\mathrm{tok}}) * [\mathrm{Bor} : i](V_{\mathrm{bor}}) * [\kappa]_q(V_{\mathrm{tok}}) \Rrightarrow_{\mathcal{N}_{\mathrm{lft}}} \exists V.\, V \sqsubseteq V_{\mathrm{tok}} \sqcup V_{\mathrm{bor}} * \triangleright P(V) * \left(\forall V'_{\mathrm{tok}}.\, V_{\mathrm{tok}} \sqsubseteq V'_{\mathrm{tok}} * \triangleright P(V'_{\mathrm{tok}} \sqcup V) \mathrel{-\!\!*\!\!\!\Rrightarrow}_{\mathcal{N}_{\mathrm{lft}}} [\mathrm{Bor} : i](V'_{\mathrm{tok}} \sqcup V) * [\kappa]_q(V'_{\mathrm{tok}})\right)$$

LftL-idx-bor-unnest
$$\&^{\kappa}_{i}\, P * \&^{\kappa'}_{\mathbf{full}}([\mathrm{Bor} : i]) \Rrightarrow_{\mathcal{N}_{\mathrm{lft}}} \&^{\kappa \sqcap \kappa'}_{\mathbf{full}}\, P$$

LftL-idx-bor-iff
$$\frac{\triangleright \Box(P \Leftrightarrow Q)}{\&^{\kappa}_{i}\, P \Rightarrow \&^{\kappa}_{i}\, Q}$$

Internal persistent atomic borrows.

LftL-bor-in-at
$$\&^{\kappa}_{\mathbf{full}}\, P \Rrightarrow_{\mathcal{N}_{\mathrm{lft}}} \&^{\kappa/0}_{\mathbf{at}}\, P$$

LftL-in-at-acc
$$\&^{\kappa/0}_{\mathbf{at}}\, P \vdash \langle [\kappa]_q \Longleftrightarrow V_b.\, \triangleright \lfloor P \rfloor_{\sqcup V_b} \rangle^{\emptyset}_{\mathcal{N}_{\mathrm{lft}}}$$

LftL-in-at-iff
$$\frac{\triangleright \Box(P \Leftrightarrow Q)}{\&^{\kappa/0}_{\mathbf{at}}\, P \Rightarrow \&^{\kappa/0}_{\mathbf{at}}\, Q}$$

LftL-in-at-shorten
$$\frac{\kappa' \sqsubseteq \kappa}{\&^{\kappa/0}_{\mathbf{at}}\, P \Rightarrow \&^{\kappa'/0}_{\mathbf{at}}\, P}$$

Finally, the rule LftL-bor-acc-atomic-strong provides a way to access a full borrow *without* having a proof that the lifetime is still ongoing.

A closer look at lifetimes. Before we go on talking about the lifetime logic rules, we have to become more concrete about what a *lifetime* κ is. Lifetimes κ form a partial commutative monoid with unit ε. We will also refer to the composition operation ($\sqcap$) as *intersection* of lifetimes. Moreover, the PCM has to be *cancellable*, which means that the composition function is injective.

Furthermore, we define the following inclusion relation on lifetimes:

$$\kappa \sqsubseteq \kappa' := \Box \left(\forall q.\, \langle [\kappa]_q \Longleftrightarrow q'.\, [\kappa']_{q'} \rangle_{\mathcal{N}_{\mathrm{lft}}} \right)$$

This says that κ is dynamically shorter than κ' if, given any fraction the token for κ, we can produce some fraction of the token for κ'. It is easy to show that this inclusion interacts as expected with lifetime intersection (LftL-incl-isect).

Indexed borrows. While the proof rules given so far bring us pretty far, it turns out that for some of the advanced reasoning we need to do for Rust, they do not suffice. As we start to build more complicated protocols involving full borrows, the fact that $\&^{\kappa}_{\mathbf{full}} P$ is neither timeless nor persistent really becomes a problem.

For this reason, the logic provides a way to *decompose* a full borrow into timeless and persistent pieces (the borrow token and the indexed borrow, respectively), which are tied together by an *index* i (LftL-bor-idx). Indexed borrows can be opened using LftL-idx-acc, but they cannot be strengthened, reborrowed or split. Furthermore, indexed borrows can be *shortened* (LftL-idx-shorten) following the dynamic lifetime inclusion $\kappa' \sqsubseteq \kappa$.

Indexed borrows are used to state the rule LftL-idx-bor-unnest, which will be used later to prove two important derived rules: unnesting and reborrowing.

Internal atomic persistent borrows. They are a primitive form of atomic persistent borrow (see the pargraph below about atomic persistent borrows). They have the same opening and closing rules as atomic peristent borrows, but use $\mathcal{N}_{\mathrm{lft}}$ as namespace, which could not be used with atomic persistent borrows.

Internally, they are implemented in a very similar fashion as atomic persistent borrows. The reason we need them is that they are used for implementing fractured borrows, which are in turn used for creating dynamic lifetime inclusion, and this cannot afford using a different mask as $\mathcal{N}_{\mathrm{lft}}$.

3.2 Derived Forms of Borrowing

Fig. 14 shows some rules that can be derived from the basic rules discussed in the previous subsection.

Furthermore, we introduce in Fig. 15 some derived forms of borrowing – that is, assertions that share are somewhat like $\&^{\kappa}_{\mathbf{full}} P$, but not exactly.

Reborrowing. Two The rule LftL-reborrow lets us *reborrow* a $\&^{\kappa}_{\mathbf{full}} P$, which means that we can pick some statically shorter lifetime $\kappa' \sqsubseteq \kappa$ and obtain P borrowed at κ'. When κ' ends, we can get our original full borrow back.

The rule LftL-bor-unnest is related. It deals with the case that we have a full borrow of a full borrow ($\&^{\kappa'}_{\mathbf{full}} \&^{\kappa}_{\mathbf{full}} P$). If we have already opened that full borrow and stripped a way the $\triangleright$ added

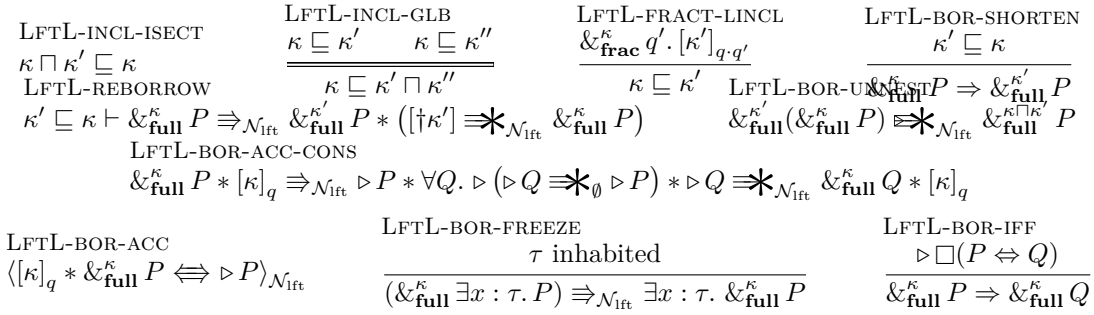

Figure 14: Lifetime logic derived rules

by opening, then we can use LftL-bor-unnest to "unnest" the full borrow in the sense that we end up with a full borrow at the intersected lifetime ($\&^{\kappa'\sqcap\kappa}_{\mathbf{full}} P$).

Both of these rules are *derived* from LftL-idx-bor-unnest.

Persistent borrows. Persistent borrows are a persistent version of borrows. This means that many parties are allowed to get access to its content. In order to avoid reentrant accesses, we can use *two* different mechanisms, giving rise to two flavors of persistent borrows.

Similarly to invariants in Iris, the first possible mechanism is to force only atomic accesses. We then get *atomic persistent borrows*, which are essentially like invariant in Iris with the additional quirk that the invariant is only maintained for the duration of the lifetime of the borrow. They can be defined as follows:

$$\&^{\kappa/\mathcal{N}}_{\mathbf{at}} P := \exists i.\, \&^{\kappa}_{i} P * \mathcal{N} \# \mathcal{N}_{\mathrm{lft}} * \boxed{[\mathrm{Bor} : i]}^{\mathcal{N}}$$

The other possible mechanism is to restrict the persistent borrow to be used in a threaded manner, by using the mechanism of *non-atomic invariants* described in the Iris documentation (and can be adapted to the GPFSL logic with the same rules). The persistent borrows of this other flavor are called *non-atomic persistent borrows*. They can be defined by:

$$\&^{\kappa/p.\mathcal{N}}_{\mathbf{na}} P := \exists i.\, \&^{\kappa}_{i} P * \mathsf{NaInv}^{p.\mathcal{N}}([\mathrm{Bor} : i])$$

Fractured borrows. A *fractured borrow* is a borrow of a permission $\Phi(q)$ that can be *fractured*, *i.e.*, decomposed according to a fraction:

$$\Phi(q_1 + q_2) \Leftrightarrow \Phi(q_1) * \Phi(q_2)$$

Intuitively, it should be possible to share such a borrow, and still obtain some fraction of Φ via a non-atomic accessor, *i.e.*, $\Phi(q)$ can actually be kept around for non-atomic expressions. This is because even if other threads are concurrently accessing the borrow, they will always leave *some* fraction of Φ in the borrow.

Fractured borrows are particularly interesting for giving rise to dynamic lifetime inclusion (LftL-fract-lincl).

Notation	Meaning	Timeless	Persistent
$\&^{\kappa/\mathcal{N}}_{\mathbf{at}}\, P$	There is a *atomic persistent borrow* of P for κ in namespace $\mathcal{N}$	No	Yes
$\&^{\kappa}_{\mathbf{frac}}\, \lambda q.\, P$	There is a *fractured borrow* of $\lambda q.\, P$ for κ	No	Yes
$\&^{\kappa/p.\mathcal{N}}_{\mathbf{na}}\, P$	There is a *non-atomic persistent borrow* of P for κ in non-atomic invariant pool p, namespace $\mathcal{N}$	No	Yes

Atomic persistent borrows

LftL-bor-at
$$\mathcal{N} \# \mathcal{N}_{\text{lft}} \vdash \&^{\kappa}_{\mathbf{full}}\, P \Rrightarrow_{\mathcal{N}_{\text{lft}}} \&^{\kappa/\mathcal{N}}_{\mathbf{at}}\, P$$

LftL-at-shorten
$$\frac{\kappa' \sqsubseteq \kappa}{\&^{\kappa/\mathcal{N}}_{\mathbf{at}}\, P \Rightarrow \&^{\kappa'/\mathcal{N}}_{\mathbf{at}}\, P}$$

LftL-at-acc
$$\&^{\kappa/\mathcal{N}}_{\mathbf{at}}\, P \vdash \langle [\kappa]_q \Longleftrightarrow V_b.\, \triangleright \lfloor P \rfloor_{\sqcup V_b} \rangle^{\mathcal{N}_{\text{lft}}}_{\mathcal{N}_{\text{lft}}, \mathcal{N}}$$

LftL-at-iff
$$\frac{\triangleright \Box (P \Leftrightarrow Q)}{\&^{\kappa/\mathcal{N}}_{\mathbf{at}}\, P \Rightarrow \&^{\kappa/\mathcal{N}}_{\mathbf{at}}\, Q}$$

Non-atomic persistent borrows

LftL-bor-na
$$\&^{\kappa}_{\mathbf{full}}\, P \Rrightarrow_{\mathcal{N}} \&^{\kappa/p.\mathcal{N}}_{\mathbf{na}}\, P$$

LftL-na-shorten
$$\frac{\kappa' \sqsubseteq \kappa}{\&^{\kappa/p.\mathcal{N}}_{\mathbf{na}}\, P \Rightarrow \&^{\kappa'/p.\mathcal{N}}_{\mathbf{na}}\, P}$$

LftL-na-acc
$$\&^{\kappa/p.\mathcal{N}}_{\mathbf{na}}\, P \vdash \langle [\kappa]_q * [\mathrm{Na} : p.\mathcal{N}] \Longleftrightarrow \triangleright P \rangle_{\mathcal{N}_{\text{lft}}, \mathcal{N}}$$

LftL-na-iff
$$\frac{\triangleright \Box (P \Leftrightarrow Q)}{\&^{\kappa/\mathcal{N}.P}_{\mathbf{na}} \Rightarrow \&^{\kappa/\mathcal{N}.Q}_{\mathbf{na}}}$$

Fractured borrows

LftL-bor-fracture
$$\frac{\forall q_1, q_2.\, \Phi(q_1 + q_2) \Leftrightarrow \Phi(q_1) * \Phi(q_2)}{\&^{\kappa}_{\mathbf{full}}\, \Phi(1) \Rrightarrow_{\mathcal{N}_{\text{lft}}} \&^{\kappa}_{\mathbf{frac}}\, \Phi}$$

LftL-fract-shorten
$$\frac{\kappa' \sqsubseteq \kappa}{\&^{\kappa}_{\mathbf{frac}}\, \Phi \Rightarrow \&^{\kappa'}_{\mathbf{frac}}\, \Phi}$$

LftL-fract-acc
$$\&^{\kappa}_{\mathbf{frac}}\, \Phi \vdash \langle [\kappa]_q \Longleftrightarrow q'.\, \triangleright \Phi(q') \rangle_{\mathcal{N}_{\text{lft}}}$$

LftL-fract-iff
$$\frac{\triangleright \Box (\forall q.\, \Phi(q) \Leftrightarrow \Psi(q))}{\&^{\kappa}_{\mathbf{frac}}\, \Phi \Rightarrow \&^{\kappa}_{\mathbf{frac}}\, \Phi}$$

Figure 15: Lifetime logic derived forms

4 GPFSL

GPFSL is an extension of iGPS (Kaiser et al. [2017]) that adopts the fence modalities from FSL (Doko and Vafeiadis [2016, 2017]). Fig. 16 lists the rules for traditional points-to assertions (non-atomics). Fig. 17 lists the rules for fork and fences.

GPFSL also combines GPS single-location protocols and iGPS single-write protocols with atomic borrows (Fig. 18, Fig. 19, Fig. 21, Fig. 22, Fig. 23, Fig. 24). These protocols are used to verify `Mutex`, `RwLock`.

GPFSL also develops protocols based on *view-dependent cancellable* invariants, which is a simpler variant of the lifetime logic. The protocols differ slightly from the atomic-borrows-based versions. Some of them are given in Fig. 25 and Fig. 26. These protocols are used to verify `Arc<T>`, `thread::spawn`, and `rayon::join`.

NA-FRAC-AGREE
$$\ell \overset{q}{\mapsto} v * \ell \overset{q'}{\mapsto} v' \Leftrightarrow \ell \overset{q+q'}{\longmapsto} v * v = v'$$

NA-FREEABLE-COMBINE
$$\dagger_q^m\, \ell * \dagger_{q'}^{m'}\, \ell + m \Leftrightarrow \dagger_{q+q'}^{m+m'}\, \ell$$

NA-ALLOC
$$\{\mathsf{True}\}\, \mathtt{alloc}(n)\, \{\ell.\, \exists \overline{v}.\, \ell \mapsto \overline{v} * |\overline{v}| = n * \dagger_1^n\, \ell\}$$

NA-FREE
$$\left\{\ell \mapsto \overline{v} * \dagger_1^{|\overline{v}|}\, \ell\right\} \mathtt{free}(|\overline{v}|, v)\, \{\mathsf{True}\}$$

NA-READ
$$\left\{\ell \overset{q}{\mapsto} v\right\}^* \ell \left\{v'.\, v' = v * \ell \overset{q}{\mapsto} v\right\}$$

NA-WRITE
$$\{\ell \mapsto v\}\, \ell := w\, \{\ell \mapsto w\}$$

NA-MEMCPY
$$\frac{|\overline{v}_1| = |\overline{v}_2| = n}{\left\{\ell_1 \mapsto \overline{v}_1 * \ell_2 \overset{q}{\mapsto} \overline{v}_2\right\} \ell_1 :=_n {}^*\ell_2 \left\{\ell_1 \mapsto \overline{v}_2 * \ell_2 \overset{q}{\mapsto} \overline{v}_2\right\}}$$

Figure 16: Non-atomics rules.

FORK
$$\frac{\forall \rho.\, \{P\}\, e\, \mathsf{in}\, \rho\, \{\mathsf{True}\}}{\{P\}\, \mathtt{fork}\, \{\, e\, \}\, \{\mathsf{True}\}}$$

REL-FENCE
$$\{P\}\, \mathtt{fence}_{\mathtt{rel}}\, \mathsf{in}\, \pi\, \{\Delta_\pi\, P\}$$

ACQ-FENCE
$$\{\nabla_\pi\, P\}\, \mathtt{fence}_{\mathtt{acq}}\, \mathsf{in}\, \pi\, \{P\}$$

Figure 17: Fork and fences rules.

AtBor-N-persistent
$$\&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}} \Rightarrow \Box \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}$$

AtBor-N-local
$$\&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}} \Rightarrow \mathcal{R}(\ell,t,s,v,\mathcal{I})$$

AtBor-N-local-join
$$\mathcal{R}(\ell,t,s,v,\mathcal{I}) * \&^\kappa \boxed{\ell : (t',s',v') \mid \mathcal{I}} \Rightarrow \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}$$

AtBor-N-init
$$[\kappa]_q \mathrel{-\!\!*} \&^\kappa_{\mathbf{full}} (\exists v.\, \ell \mapsto v * P(v)) \mathrel{-\!\!*} (\forall t, v.\, \triangleright P(v) \mathrel{-\!\!*} \triangleright \mathcal{I}_w(\ell,t,s,v)) \mathrel{-\!\!*} (\Box \forall t,s,v.\, \triangleright \mathcal{I}_w(\ell,t,s,v) \Rrightarrow \triangleright P(v)) \Rrightarrow\!\!\!\!\ast\; [\kappa]_q * \exists t, v.\, \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}$$

AtBor-N-rlx-read
$$\frac{\forall t' \sqsupseteq t, s' \sqsupseteq s, v'.\, \mathcal{I}_r(\ell,t',s',v') \Rrightarrow \mathcal{I}_r(\ell,t',s',v') * P(t',s',v') \quad \forall t' \sqsupseteq t, s' \sqsupseteq s, v'.\, \mathcal{I}_w(\ell,t',s',v') \Rrightarrow \mathcal{I}_w(\ell,t',s',v') * P(t',s',v')}{\left\{[\kappa]_q * \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}\right\} {}^{*\mathtt{rlx}}\ell \,\mathsf{in}\, \pi \left\{v'.\, [\kappa]_q * \exists t' \sqsupseteq t, s' \sqsupseteq s.\, \&^\kappa \boxed{\ell : (t',s',v') \mid \mathcal{I}} * \nabla_\pi P(t',s',v')\right\}}$$

AtBor-N-acq-read
$$\frac{\forall t', s', v'.\, \mathcal{I}_r(\ell,t',s',v') \Rrightarrow \mathcal{I}_r(\ell,t',s',v') * P(t',s',v') \quad \forall t', s', v'.\, \mathcal{I}_w(\ell,t',s',v') \Rrightarrow \mathcal{I}_w(\ell,t',s',v') * P(t',s',v')}{\left\{[\kappa]_q * \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}\right\} {}^{*\mathtt{acq}}\ell \left\{v'.\, [\kappa]_q * \exists t' \sqsupseteq t, s' \sqsupseteq s.\, \&^\kappa \boxed{\ell : (t',s',v') \mid \mathcal{I}} * P(t',s',v')\right\}}$$

AtBor-N-rlx-write
$$\left\{[\kappa]_q * \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}} * \Delta_\pi (\forall t' > t.\, \mathcal{R}(\ell,t',s',v',\mathcal{I}) \Rrightarrow \mathcal{I}_w(\ell,t',s',v'))\right\}$$
$$\ell :=_{\mathtt{rlx}} v' \,\mathsf{in}\, \pi$$
$$\left\{[\kappa]_q * \&^\kappa \boxed{\ell : (t',s',v') \mid \mathcal{I}}\right\}$$

AtBor-N-rel-write
$$\left\{[\kappa]_q * \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}} * (\forall t' > t.\, \mathcal{R}(\ell,t',s',v',\mathcal{I}) \Rrightarrow \mathcal{I}_w(\ell,t',s',v'))\right\}$$
$$\ell :=_{\mathtt{rel}} v' \,\mathsf{in}\, \pi$$
$$\left\{[\kappa]_q * \&^\kappa \boxed{\ell : (t',s',v') \mid \mathcal{I}}\right\}$$

Figure 18: Atomic-borrow-based normal GPFSL prototocols.

$$\Delta_\pi^{?o_w}\, P := (\mathsf{if}\ o_w = \texttt{rel}\ \mathsf{then}\ P\ \mathsf{else}\ \Delta_\pi\, P)$$

$$\nabla_\pi^{?o_r}\, P := (\mathsf{if}\ o_r = \texttt{acq}\ \mathsf{then}\ P\ \mathsf{else}\ \nabla_\pi\, P)$$

AtBor-N-cas

$$\frac{\begin{array}{c} o_f, o_r \in \{\texttt{rlx}, \texttt{acq}\} \qquad o_w \in \{\texttt{rlx}, \texttt{rel}\} \\ \forall t' \sqsupseteq t, s' \sqsupseteq s, v'.\, \mathcal{I}_w(\ell, t', s', v') \lor \mathcal{I}_r(\ell, t', s', v') \Rrightarrow (\vdash v_r =^? v') \\ \forall t' \sqsupseteq t, s' \sqsupseteq s, v'.\, (\vdash v' \neq v_r) \Rrightarrow \mathcal{I}_r(\ell, t', s', v') \Rrightarrow \mathcal{I}_r(\ell, t', s', v') * R(t', s', v') \\ \forall t' \sqsupseteq t, s' \sqsupseteq s, v'.\, (\vdash v' \neq v_r) \Rrightarrow \mathcal{I}_w(\ell, t', s', v') \Rrightarrow \mathcal{I}_w(\ell, t', s', v') * R(t', s', v') \\ \forall t' \sqsupseteq t, s' \sqsupseteq s.\, \triangleright \mathcal{I}_w(\ell, t', s', v_r) \Rrightarrow \triangleright Q_1(t', s') * \triangleright Q_2(t', s') \\ \Delta_\pi^{?o_w} \left(\begin{array}{l} \forall t' \sqsupseteq t, s' \sqsupseteq s.\, P \mathrel{-\!\!*} \triangleright Q_2(t', s') \Rrightarrow \exists s'' \sqsupseteq s'.\, \forall t'' > t.\, \triangleright \mathcal{R}(\ell, t'', s'', v_w, \mathcal{I}) \\ \quad \Rrightarrow (\langle obj \rangle\, (\triangleright Q_1(t', s') \Rrightarrow \triangleright \mathcal{I}_m(\ell, t', s', v_r))) * \Rrightarrow\!\!\!\!\!\!\ast\ (Q(t'', s'') * \mathcal{I}_w(\ell, t'', s'', v_w)) \end{array} \right) \end{array}}{\left\{ [\kappa]_q * \&^\kappa \boxed{\ell : (t, s, v) \mid \mathcal{I}} * \Delta_\pi^{?o_w}\, P \right\}\ \texttt{CAS}(\ell, v_r, v_w, o_f, o_r, o_w)\ \mathsf{in}\ \pi\ \left\{ b.\, [\kappa]_q * \exists s' \sqsupseteq s. \begin{array}{l} b = 1 * \exists t' > t.\, \&^\kappa \boxed{\ell : (t', s', v_w) \mid \mathcal{I}} * \nabla_\pi^{?o_r}\, Q(t'', s'') \\ \lor\ b = 0 * \Delta_\pi^{?o_w}\, P * \exists t' \geq t, v'.\, (\vdash v' \neq v_r) * \&^\kappa \boxed{\ell : (t', s', v') \mid \mathcal{I}} * \nabla_\pi^{?o_f}\, R(t', s', v') \end{array} \right\}}$$

Figure 19: CAS rule for atomic-borrow-based normal GPFSL prototocols.

SW-WRITER-LOCAL-EXCLUSIVE
$\mathcal{W}(\ell, t, s, v, \mathcal{I}) * \mathcal{W}(\ell, t, s, v, \mathcal{I}) \Rightarrow \mathsf{False}$

SW-LOCAL-WRITER-READER
$\mathcal{W}(\ell, t, s, v, \mathcal{I}) \Rightarrow \mathcal{R}(\ell, t, s, v, \mathcal{I})$

SW-CREADERS-LOCAL-JOIN
$\mathcal{R}^q_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I}) * \mathcal{R}^{q'}_{\mathrm{shr}}(\ell, t', s', v', \mathcal{I}) \Rightarrow \mathcal{R}^{q+q'}_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I})$

SW-CREADERS-LOCAL-SPLIT
$\mathcal{R}^{q+q'}_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I}) \Rightarrow \mathcal{R}^q_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I}) * \mathcal{R}^{q'}_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I})$

SW-CWRITER-LOCAL-EXCLUSIVE
$\mathcal{W}_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I}) * \mathcal{W}_{\mathrm{shr}}(\ell, t', s', v', \mathcal{I}) \Rightarrow \mathsf{False}$

SW-SHARE-LOCAL-CWRITER
$\mathcal{W}(\ell, t, s, v, \mathcal{I}) \Rightarrow \mathcal{W}_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I}) * \mathcal{R}^1_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I})$

SW-READER-CREADER-LOCAL
$\mathcal{R}^q_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I}) \Rightarrow \mathcal{R}(\ell, t, s, v, \mathcal{I})$

SW-CW-LOCAL-EXCLUSIVE
$\mathcal{W}(\ell, t, s, v, \mathcal{I}) * \mathcal{W}_{\mathrm{shr}}(\ell, t', s', v', \mathcal{I}) \Rightarrow \mathsf{False}$

SW-CR-LOCAL-EXCLUSIVE
$\mathcal{W}(\ell, t, s, v, \mathcal{I}) * \mathcal{R}^q_{\mathrm{shr}}(\ell, t', s', v', \mathcal{I}) \Rightarrow \mathsf{False}$

Figure 20: Local assertions of single-writer GPFSL prototocols.

AtBor-sw-reader-persistent
$$\&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}_R \Rightarrow \Box \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}_R$$

AtBor-sw-reader-local
$$\&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}_R \Rightarrow \mathcal{R}(\ell,t,s,v,\mathcal{I})$$

AtBor-sw-reader-local-join
$$\mathcal{R}(\ell,t,s,v,\mathcal{I}) * \&^\kappa \boxed{\ell : (t',s',v') \mid \mathcal{I}}_R \Rightarrow \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}_R$$

AtBor-sw-writer-local
$$\&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}_W \Rightarrow \mathcal{W}(\ell,t,s,v,\mathcal{I})$$

AtBor-sw-writer-local-join
$$\mathcal{W}(\ell,t,s,v,\mathcal{I}) * \&^\kappa \boxed{\ell : (t',s',v') \mid \mathcal{I}}_R \Rightarrow \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}_W$$

AtBor-sw-creader-local
$$\&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}^q_{CR} \Rightarrow \mathcal{R}^q_{\mathrm{shr}}(\ell,t,s,v,\mathcal{I})$$

AtBor-sw-creader-local-join
$$\mathcal{R}^q_{\mathrm{shr}}(\ell,t,s,v,\mathcal{I}) * \&^\kappa \boxed{\ell : (t',s',v') \mid \mathcal{I}}_R \Rightarrow \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}^q_{CR}$$

AtBor-sw-cwriter-local
$$\&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}_{CW} \Rightarrow \mathcal{W}_{\mathrm{shr}}(\ell,t,s,v,\mathcal{I})$$

AtBor-sw-cwriter-local-join
$$\mathcal{W}_{\mathrm{shr}}(\ell,t,s,v,\mathcal{I}) * \&^\kappa \boxed{\ell : (t',s',v') \mid \mathcal{I}}_R \Rightarrow \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}_{CW}$$

AtBor-sw-unshare-local-cwriter
$$[\kappa]_q * \mathcal{W}_{\mathrm{shr}}(\ell,t,s,v,\mathcal{I}) * \&^\kappa \boxed{\ell : (t',s',v') \mid \mathcal{I}}^1_{CR} \Rrightarrow [\kappa]_q * \&^\kappa \boxed{\ell : (t,s,v) \mid \mathcal{I}}_W$$

Figure 21: Atomic-borrow-based single-writer GPFSL prototocols (1).

AtBor-sw-init

$$[\kappa]_q * \&^{\kappa}_{\mathbf{full}} (\exists v.\, \ell \mapsto v * P(v)) \mathrel{-\!\!*} (\forall t, v.\, \triangleright P(v) \mathrel{-\!\!*} \mathcal{W}(\ell, t, s, v, \mathcal{I}) \Rrightarrow \triangleright \mathcal{I}_w(\ell, t, s, v) * Q(t, v)) \mathrel{-\!\!*} (\square \forall t, s, v.\, \triangleright \mathcal{I}_w(\ell, t, s, v) \Rrightarrow \triangleright P(v)) \Rrightarrow\!\!\!\!\!\!\ast \; [\kappa]_q * \exists t, v.\, \&^{\kappa} \boxed{\ell : (t, s, v) \mid \mathcal{I}}_R * Q(t, v)$$

AtBor-sw-read

$$\frac{o \in \{\mathtt{rlx}, \mathtt{acq}\} \quad \forall t' \sqsupseteq t, s' \sqsupseteq s, v'.\, \mathcal{I}_r(\ell, t', s', v') \Rrightarrow \mathcal{I}_r(\ell, t', s', v') * P(t', s', v') \quad \forall t' \sqsupseteq t, s' \sqsupseteq s, v'.\, \mathcal{I}_w(\ell, t', s', v') \Rrightarrow \mathcal{I}_w(\ell, t', s', v') * P(t', s', v') \quad \forall t' \sqsupseteq t, s' \sqsupseteq s, v'.\, \mathcal{I}_m(\ell, t', s', v') \Rrightarrow \mathcal{I}_m(\ell, t', s', v') * P(t', s', v')}{\left\{[\kappa]_q * \&^{\kappa} \boxed{\ell : (t, s, v) \mid \mathcal{I}}_R\right\} \; {}^{*o}\ell \text{ in } \pi \; \left\{v'.\, [\kappa]_q * \exists t' \sqsupseteq t, s' \sqsupseteq s.\, \&^{\kappa} \boxed{\ell : (t', s', v') \mid \mathcal{I}}_R * \nabla^{?o}_{\pi} P(t', s', v')\right\}}$$

AtBor-sw-exclusive-read

$$\frac{o \in \{\mathtt{rlx}, \mathtt{acq}\} \qquad \mathcal{I}_w(\ell, t, s, v) \Rrightarrow \mathcal{I}_w(\ell, t, s, v) * P}{\left\{[\kappa]_q * \&^{\kappa} \boxed{\ell : (t, s, v) \mid \mathcal{I}}_W *\right\} \; {}^{*o}\ell \text{ in } \pi \; \left\{v.\, [\kappa]_q * \&^{\kappa} \boxed{\ell : (t, s, v) \mid \mathcal{I}}_W * \nabla^{?o}_{\pi} P\right\}}$$

AtBor-sw-creader-read

$$\frac{o \in \{\mathtt{rlx}, \mathtt{acq}\} \quad \forall t' \sqsupseteq t, s' \sqsupseteq s, v'.\, \mathcal{I}_r(\ell, t', s', v') \Rrightarrow \mathcal{I}_r(\ell, t, s, v) * P(t', s', v') \quad \forall t' \sqsupseteq t, s' \sqsupseteq s, v'.\, \mathcal{I}_w(\ell, t', s', v') \Rrightarrow \mathcal{I}_w(\ell, t', s', v') * P(t', s', v')}{\left\{[\kappa]_{q_0} * \&^{\kappa} \boxed{\ell : (t, s, v) \mid \mathcal{I}}^{q}_{CR}\right\} \; {}^{*o}\ell \text{ in } \pi \; \left\{v'.\, [\kappa]_{q_0} * \exists t' \sqsupseteq t, s' \sqsupseteq s.\, \&^{\kappa} \boxed{\ell : (t', s', v') \mid \mathcal{I}}^{q}_{CR} * \nabla^{?o}_{\pi} P(t', s', v')\right\}}$$

Figure 22: Atomic-borrow-based single-writer GPFSL prototocols (2).

AtBor-sw-write

$$\frac{o \in \{\texttt{rlx}, \texttt{rel}\} \qquad s \sqsubseteq s' \qquad \triangleright \langle obj \rangle \, (\mathcal{I}_w(\ell, t, s, v) \Rrightarrow \mathcal{I}_m(\ell, t, s, v) * Q)}{\left\{[\kappa]_q * \&^\kappa \boxed{\ell : (t, s, v) \mid \mathcal{I}}_W * \Delta_\pi^{?o} \, (\forall t' > t. \, \mathcal{R}(\ell, t', s', v', \mathcal{I}) \Rrightarrow \mathcal{I}_w(\ell, t', s', v'))\right\}}$$

$$\ell :=_o v' \text{ in } \pi$$

$$\left\{[\kappa]_q * \&^\kappa \boxed{\ell : (t', s', v') \mid \mathcal{I}}_W * Q\right\}$$

AtBor-sw-rel-write

$$\frac{s \sqsubseteq s' \qquad \triangleright \langle obj \rangle \, (\mathcal{I}_w(\ell, t, s, v) \Rrightarrow Q_1 * Q_2)}{\left\{\begin{array}{l}[\kappa]_q * \&^\kappa \boxed{\ell : (t, s, v) \mid \mathcal{I}}_W * \\ \triangleright \left(\forall t' > t. \, \mathcal{W}(\ell, t', s', v', \mathcal{I}) \mathrel{-\!\!*} Q_2 \Rrightarrow\!\!\!\!* \left(\langle obj \rangle \, (Q_1 \Rrightarrow\!\!\!\!* \, \mathcal{I}_m(\ell, t, s, v)) * \Rrightarrow\!\!\!\!\Rightarrow (\mathcal{I}_w(\ell, t', s', v') * Q(t'))\right)\right)\end{array}\right\}}$$

$$\ell :=_{\texttt{rel}} v'$$

$$\left\{[\kappa]_q * \exists t' > t. \, \&^\kappa \boxed{\ell : (t', s', v') \mid \mathcal{I}}_R * Q(t')\right\}$$

Figure 23: Atomic-borrow-based single-writer GPFSL prototocols (3).

$$b\ ?\ P := \mathsf{if}\ b\ \mathsf{then}\ P\ \mathsf{else}\ \mathsf{True} \qquad b\ ?\ P\ :\ Q := \mathsf{if}\ b\ \mathsf{then}\ P\ \mathsf{else}\ Q$$

AtBor-sw-creader-cas

$$\frac{\begin{array}{c} o_f, o_r \in \{\mathtt{rlx}, \mathtt{acq}\} \qquad o_w \in \{\mathtt{rlx}, \mathtt{rel}\} \\ \forall t' \sqsupseteq t, s' \sqsupseteq s, v'. \mathcal{I}_w(\ell, t', s', v') \lor \mathcal{I}_r(\ell, t', s', v') \Rrightarrow (\vdash v_r =^? v') \\ \forall t' \sqsupseteq t, s' \sqsupseteq s, v'. (\vdash v' \neq v_r) \Rrightarrow \mathcal{I}_r(\ell, t', s', v') \Rrightarrow \mathcal{I}_r(\ell, t', s', v') * R(t', s', v') \\ \forall t' \sqsupseteq t, s' \sqsupseteq s, v'. (\vdash v' \neq v_r) \Rrightarrow \mathcal{I}_w(\ell, t', s', v') \Rrightarrow \mathcal{I}_w(\ell, t', s', v') * R(t', s', v') \\ \forall t' \sqsupseteq t, s' \sqsupseteq s. \triangleright \mathcal{I}_w(\ell, t', s', v_r) \Rrightarrow \triangleright Q_1(t', s') * \triangleright Q_2(t', s') \\ \Delta_\pi^{?o_w} \begin{pmatrix} \forall t' \sqsupseteq t, s' \sqsupseteq s. P \mathrel{-\!\!*} \triangleright Q_2(t', s') \Rrightarrow \triangleright \mathcal{W}_{\mathrm{shr}}(\ell, t', s', v_r) * \exists s'' \sqsupseteq s'. \\ \quad \forall t'' > t. \triangleright \mathcal{W}_{\mathrm{shr}}(\ell, t'', s'', v_w, \mathcal{I}) \mathrel{-\!\!*} b_{\mathsf{drop}}\ ?\ \triangleright \mathcal{R}^q_{\mathrm{shr}}(\ell, t'', s'', v_w, \mathcal{I}) \\ \quad \Rrightarrow (\langle obj \rangle\, (\triangleright Q_1(t', s') \Rrightarrow \triangleright \mathcal{I}_m(\ell, t', s', v_r))) * \Rrightarrow\!\!\!\!\!\!\diagup\ (Q(t'', s'') * \mathcal{I}_w(\ell, t'', s'', v_w)) \end{pmatrix} \end{array}}{\begin{array}{l} \left\{ [\kappa]_{q_0} * \&^\kappa \boxed{\ell : (t, s, v) \mid \mathcal{I}}^q_{CR} * \Delta_\pi^{?o_w} P \right\} \\ \quad \mathtt{CAS}(\ell, v_r, v_w, o_f, o_r, o_w)\ \mathsf{in}\ \pi \\ \left\{ b. [\kappa]_{q_0} * \exists s' \sqsupseteq s. \begin{array}{c} b = 1 * \exists t' > t. \left(b_{\mathsf{drop}}\ ?\ \&^\kappa \boxed{\ell : (t', s', v_w) \mid \mathcal{I}}_R\ :\ \&^\kappa \boxed{\ell : (t', s', v_w) \mid \mathcal{I}}^q_{CR} \right) * \\ \nabla_\pi^{?o_r} Q(t', s') \\ \lor\ b = 0 * \Delta_\pi^{?o_w} P * \exists t' \geq t, v'. (\vdash v' \neq v_r) * \&^\kappa \boxed{\ell : (t', s', v') \mid \mathcal{I}}^q_{CR} * \\ \nabla_\pi^{?o_f} R(t', s', v') \end{array} \right\} \end{array}}$$

AtBor-sw-reader-cas

$$\frac{\begin{array}{c} o_f, o_r \in \{\mathtt{rlx}, \mathtt{acq}\} \qquad o_w \in \{\mathtt{rlx}, \mathtt{rel}\} \\ \forall t' \sqsupseteq t, s' \sqsupseteq s, v'. \mathcal{I}_w(\ell, t', s', v') \lor \mathcal{I}_r(\ell, t', s', v') \lor \mathcal{I}_m(\ell, t', s', v') \Rrightarrow (\vdash v_r =^? v') \\ \forall t' \sqsupseteq t, s' \sqsupseteq s, v'. (\vdash v' \neq v_r) \Rrightarrow \mathcal{I}_r(\ell, t', s', v') \Rrightarrow \mathcal{I}_r(\ell, t', s', v') * R(t', s', v') \\ \forall t' \sqsupseteq t, s' \sqsupseteq s, v'. (\vdash v' \neq v_r) \Rrightarrow \mathcal{I}_w(\ell, t', s', v') \Rrightarrow \mathcal{I}_w(\ell, t', s', v') * R(t', s', v') \\ \forall t' \sqsupseteq t, s' \sqsupseteq s, v'. (\vdash v' \neq v_r) \Rrightarrow \mathcal{I}_m(\ell, t', s', v') \Rrightarrow \mathcal{I}_m(\ell, t', s', v') * R(t', s', v') \\ \forall t' \sqsupseteq t, s' \sqsupseteq s. \triangleright (P \mathrel{-\!\!*} \mathcal{I}_m(\ell, t', s', v_r) \Rrightarrow \mathsf{False}) \\ \forall t' \sqsupseteq t, s' \sqsupseteq s. \triangleright \mathcal{I}_w(\ell, t', s', v_r) \Rrightarrow \triangleright Q_1(t', s') * \triangleright Q_2(t', s') \\ \Delta_\pi^{?o_w} \begin{pmatrix} \forall t' \sqsupseteq t, s' \sqsupseteq s. P \mathrel{-\!\!*} \triangleright Q_2(t', s') \Rrightarrow \triangleright \mathcal{W}_{\mathrm{shr}}(\ell, t', s', v_r) * \exists s'' \sqsupseteq s'. \forall t'' > t. \triangleright \mathcal{W}_{\mathrm{shr}}(\ell, t'', s'', v_w, \mathcal{I}) \\ \quad \Rrightarrow (\langle obj \rangle\, (\triangleright Q_1(t', s') \Rrightarrow \triangleright \mathcal{I}_m(\ell, t', s', v_r))) * \Rrightarrow\!\!\!\!\!\!\diagup\ (Q(t'', s'') * \mathcal{I}_w(\ell, t'', s'', v_w)) \end{pmatrix} \end{array}}{\begin{array}{l} \left\{ [\kappa]_q * \&^\kappa \boxed{\ell : (t, s, v) \mid \mathcal{I}}_R * \Delta_\pi^{?o_w} P \right\} \\ \quad \mathtt{CAS}(\ell, v_r, v_w, o_f, o_r, o_w)\ \mathsf{in}\ \pi \\ \left\{ b. [\kappa]_q * \exists s' \sqsupseteq s. \begin{array}{c} b = 1 * \exists t' > t. \&^\kappa \boxed{\ell : (t', s', v_w) \mid \mathcal{I}}_R * \nabla_\pi^{?o_r} Q(t', s') \\ \lor\ b = 0 * \Delta_\pi^{?o_w} P * \exists t' \geq t, v'. (\vdash v' \neq v_r) * \&^\kappa \boxed{\ell : (t', s', v') \mid \mathcal{I}}_R * \\ \nabla_\pi^{?o_f} R(t', s', v') \end{array} \right\} \end{array}}$$

Figure 24: Atomic-borrow-based single-writer GPFSL prototocols (4).

ViewInv-sw-reader-persistent
$$\iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]_R \Rightarrow \Box\, \iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]_R$$

ViewInv-sw-reader-local
$$\iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]_R \Rightarrow \mathcal{R}(\ell, t, s, v, \mathcal{I})$$

ViewInv-sw-reader-local-join
$$\mathcal{R}(\ell, t, s, v, \mathcal{I}) * \iota\big[\overline{\ell} : \overline{(t', s', v')}\big]\big|\overline{\mathcal{I}}\big]_R \Rightarrow \iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]_R$$

ViewInv-sw-writer-local
$$\iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]_W \Rightarrow \mathcal{W}(\ell, t, s, v, \mathcal{I})$$

ViewInv-sw-writer-local-join
$$\mathcal{W}(\ell, t, s, v, \mathcal{I}) * \iota\big[\overline{\ell} : \overline{(t', s', v')}\big]\big|\overline{\mathcal{I}}\big]_R \Rightarrow \iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]_W$$

ViewInv-sw-creader-local
$$\iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]^q_{CR} \Rightarrow \mathcal{R}^q_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I})$$

ViewInv-sw-creader-local-join
$$\mathcal{R}^q_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I}) * \iota\big[\overline{\ell} : \overline{(t', s', v')}\big]\big|\overline{\mathcal{I}}\big]_R \Rightarrow \iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]^q_{CR}$$

ViewInv-sw-cwriter-local
$$\iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]_{CW} \Rightarrow \mathcal{W}_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I})$$

ViewInv-sw-cwriter-local-join
$$\mathcal{W}_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I}) * \iota\big[\overline{\ell} : \overline{(t', s', v')}\big]\big|\overline{\mathcal{I}}\big]_R \Rightarrow \iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]_{CW}$$

ViewInv-sw-unshare-local-cwriter
$$[\iota]_q * \mathcal{W}_{\mathrm{shr}}(\ell, t, s, v, \mathcal{I}) * \iota\big[\overline{\ell} : \overline{(t', s', v')}\big]\big|\overline{\mathcal{I}}\big]^1_{CR} \Rrightarrow [\iota]_q * \iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]_W$$

Figure 25: View-invariant-based single-writer GPFSL prototocols (1).

ViewInv-sw-init
$$\ell \mapsto v \mathrel{-\!\!*} (\forall \iota, t, v.\ \triangleright \mathcal{I}_w(\ell, t, s, v)) \Rrightarrow \exists \iota, t, v.\ [\iota]_1 * \iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]_W$$

ViewInv-sw-rel-write
$$\frac{s \sqsubseteq s' \qquad \triangleright \langle obj \rangle\, (\mathcal{I}_w(\ell, t, s, v) \Rrightarrow Q_1 * Q_2)}{\left\{\begin{array}{l}[\iota]_q * \iota\big[\overline{\ell} : \overline{(t, s, v)}\big]\big|\overline{\mathcal{I}}\big]_W * \\ \triangleright \Big(\forall t' > t.\, \mathcal{W}(\ell, t', s', v', \mathcal{I}) \mathrel{-\!\!*} Q_2 \mathrel{-\!\!*} [\iota]_q \Rrightarrow \big(\langle obj \rangle\, (Q_1 \Rrightarrow \mathcal{I}_m(\ell, t, s, v)) * \Rrightarrow (\mathcal{I}_w(\ell, t', s', v') * Q(t'))\big)\Big)\end{array}\right\}}$$
$$\ell :=_{\mathtt{rel}} v'$$
$$\left\{\exists t' > t.\, \iota\big[\overline{\ell} : \overline{(t', s', v')}\big]\big|\overline{\mathcal{I}}\big]_R * Q(t')\right\}$$

Figure 26: View-invariant-based single-writer GPFSL prototocols (2).

$$\mathcal{R}(\ell, t, s, v, \mathcal{I}) \Rrightarrow \Box\, \mathcal{R}(\ell, t, s, v, \mathcal{I})$$

$$\frac{\begin{array}{c}\langle obj\rangle\, \forall v', t' \geq t, s' \sqsupseteq s. \mathcal{I}_r(\ell, t', s', v') \Rrightarrow \mathcal{I}_r(\ell, t', s', v') * P(v') \\ \langle obj\rangle\, \forall v', t' \geq t, s' \sqsupseteq s. \mathcal{I}_w(\ell, t', s', v') \Rrightarrow \mathcal{I}_w(\ell, t', s', v') * P(v') \\ \langle obj\rangle\, \forall v', t' \geq t, s' \sqsupseteq s. \mathcal{I}_m(\ell, t', s', v') \Rrightarrow \mathcal{I}_m(\ell, t', s', v') * P(v')\end{array}}{\begin{array}{l}\{\mathcal{R}(\ell, t, s, v, \mathcal{I}) * \lfloor \triangleright\, \mathcal{GS}(\ell, \mathcal{I})\rfloor_V\} \\ \quad {}^{*\texttt{rlx}}\ell \text{ in } \pi \\ \{v'.\, \nabla_\pi\, P(v') * t \leq t' * s \sqsubseteq s' * \mathcal{R}(\ell, t', s', v', \mathcal{I}_r) * \lfloor \triangleright\, \mathcal{GS}(\ell, \mathcal{I})\rfloor_V\}\end{array}}$$

$$\frac{\begin{array}{c}\langle obj\rangle\, \forall v', t' \geq t, s' \sqsupseteq s. \mathcal{I}_r(\ell, t', s', v') \Rrightarrow \mathcal{I}_r(\ell, t', s', v') * P(v') \\ \langle obj\rangle\, \forall v', t' \geq t, s' \sqsupseteq s. \mathcal{I}_w(\ell, t', s', v') \Rrightarrow \mathcal{I}_w(\ell, t', s', v') * P(v') \\ \langle obj\rangle\, \forall v', t' \geq t, s' \sqsupseteq s. \mathcal{I}_m(\ell, t', s', v') \Rrightarrow \mathcal{I}_m(\ell, t', s', v') * P(v')\end{array}}{\begin{array}{l}\{\mathcal{R}(\ell, t, s, v, \mathcal{I}) * \lfloor \triangleright\, \mathcal{GS}(\ell, \mathcal{I})\rfloor_V\} \\ \quad {}^{*\texttt{acq}}\ell \\ \{v'.\, P(v') * t \leq t' * s \sqsubseteq s' * \mathcal{R}(\ell, t', s', v', \mathcal{I}_r) * \lfloor \triangleright\, \mathcal{GS}(\ell, \mathcal{I})\rfloor_V\}\end{array}}$$

$$\mathcal{W}(\ell, \mathcal{I}) * \mathcal{W}(\ell, \mathcal{I}) \Rightarrow \mathsf{False} \qquad \{\mathcal{W}(\ell)\}\, \ell :=_{\texttt{rlx}} w\, \{\mathsf{True}\} \qquad \{\mathcal{W}(\ell)\}\, \ell :=_{\texttt{rel}} w\, \{\mathsf{True}\}$$

$$\{\mathcal{R}(\ell)\}\, \texttt{CAS}(\ell, v_1, v_2, o_f, o_r, o_w)\, \{\mathsf{True}\} \qquad \{\mathcal{R}^q_{\text{shr}}(\ell)\}\, {}^{*\texttt{rlx}}\ell\, \{\mathsf{True}\} \qquad \{\mathcal{R}^q_{\text{shr}}(\ell)\}\, {}^{*\texttt{acq}}\ell\, \{\mathsf{True}\}$$

$$\{\mathcal{R}^q_{\text{shr}}(\ell)\}\, \texttt{CAS}(\ell, v_1, v_2, o_f, o_r, o_w)\, \{\mathcal{R}^q_{\text{shr}}(\ell)\}$$

Figure 27: Intermediate-level rules for GPS single-writers.

5 Case Study: `Arc`

Using the lifetime logic for views (§3), the GPFSL logic (§4), and the combination of them, we successfully ported RustBelt's semantic typing proofs of the concurrent libraries `Arc`, `Mutex`, `RwLock`, `thread::spawn`, and `rayon::join`. Of these seemingly simple libraries, `Arc`, `Mutex`, and `RwLock` encapsulate very subtle correctness conditions, since they support APIs for "extensive" resource reclamation. Our definition of "extensive" reclamation is twofold. First, by extensive reclamation we refer to the ability to fully recover all resources for deallocation at the end, which previous logics were not able to demonstrate. For example, FSL++ (Doko and Vafeiadis [2017]) verifies an `Arc` implementation that can reclaim the content (the ownership of the `T` in `Arc<T>`), but not the internal reference counter of the `Arc` itself. Meanwhile, iGPS (Kaiser et al. [2017]) is able to reclaim the ownership of the atomic integer that implements a spin lock, but is not able to reclaim the shared resource that the lock is protecting in the case the lock is unlocked. Second, by extensive reclamation, we refer to the ability to *temporarily* obtain full ownership of shared resources. For example, `Arc<T>`, `Mutex<T>`, and `RwLock<T>` have APIs that allow a thread to temporarily gain full access to the underlying content `T` when the thread can prove that it is the *unique owner* of the `Arc`, `Mutex`, or `RwLock`. Verifying this kind of temporary reclamation has never been attempted before. To the best of our knowledge, these are the very first formal, machine-checked proofs of relaxed memory algorithms that perform extensive resource reclamation.

In this section, we discuss at a high level the technical challenges in verifying `Arc` with its powerful yet delicate temporary resource reclamation scheme. This attempt to verify the full APIs of `Arc` has led us to the discovery of a data race in one of its temporary reclamation API—namely, `Arc::get_mut`. The data race is due to the use of a relaxed access that does not provide sufficient synchronization, which was not discovered by the original SC RustBelt verification because all atomic accesses had been strengthened to sequential consistency.

5.1 The Full APIs of `Arc`

`Arc<T>`, short for *Atomically Reference Counted*, is used to share atomically a value of type `T`, whose mutation is disabled by default. In order to mutate `T`, programmers need to instantiate `Arc` with wrappers that support thread-safe mutability, for example `Arc<Mutex<T>>` or `Arc<RwLock<T>>`. The following Rust example instantiates `Arc` with an atomic integer `AtomicUsize` and demonstrates how `Arc` is typically used:

```
1  let arc1 = Arc::new(AtomicUsize::new(5));
2  let arc2 = Arc::clone(&arc1);
3  thread::spawn(move || {
4    println!("child thread: {:?}", arc2.fetch_add(1, Ordering::Relaxed));
5  });
6  println!("main thread: {:?}", arc1.fetch_add(2, Ordering::Relaxed));
```

In line 1 in the main thread, a new `Arc` value `arc1` is created to govern an atomic integer allocated in shared memory. The `Arc`'s internal counter for the number of references to the content is set to 1. An `Arc` pointer acts almost like its underlying content, so in line 6 we can call `fetch_add` on `arc1` as if on the atomic integer itself. To share the content with the child thread, we create another `arc2` by `clone`-ing `arc1` (line 2), which effectively increments the internal counter to 2: there are now 2 pointers sharing the atomic integer. When the `Arc` pointers go out of scope (line 4 and 6), their destructors—the `drop` function—are called and the internal counter is decremented accordingly. The last call of `drop` will additionally deallocate the underlying content and the internal counter itself.

Arc	Weak
`new: fn(T) -> Arc<T>`	`new: fn() -> Weak<T>`
`deref: fn(&Arc<T>) -> &T`	
`clone: fn(&Arc<T>) -> Arc<T>`	`clone: fn(&Weak<T>) -> Weak<T>`
`downgrade: fn(&Arc<T>) -> Weak<T>`	`upgrade: fn(&Weak<T>) -> Option<Arc<T>>`
`drop: fn(Arc<T>) -> ()`	`drop: fn(Weak<T>) -> ()`
`get_mut: fn(&mut Arc<T>) -> Option<&mut T>`	
`make_mut: fn(&mut Arc<T>) -> &mut T`	

Figure 28: An excerpt of Rust's `Arc<T>` and `Weak<T>` APIs.

As expected, to allow concurrent updates by multiple threads, the internal counter is implemented with an atomic integer.

Fig. 28 gives an excerpt of the most intestring APIs from the `Arc` library. The four functions `Arc::new`, `Arc::deref`, `Arc::clone`, and `Arc::drop` were successfully verified by FSL++ Doko and Vafeiadis [2017]. The full APIs of Rust's `Arc<T>`, however, was not attempted in the relaxed memory setting before. Our verification must therefore tackle two extra sets of behaviors, presented as the following two main challenges.

`Arc<T>` has a subordinate type `Weak<T>` The first challenge involves a type called `Weak<T>`. `Weak<T>` itself is a variant of `Arc<T>`: it has a counter to count how many `Weak` pointers are existing, and also has the similar `clone` and `drop` functions (Fig. 28). However, `Weak<T>` is not tied to the underlying value of type `T`: while owning an `Arc<T>` guarantees that the value is still available, owning a `Weak<T>` does not prevent the underlying value to be reclaimed. Therefore, in order to access the underlying value with a `Weak` pointer, one first has to call `Weak::upgrade` to obtain an `Arc`. `Weak::upgrade` can fail when the value has already been reclaimed, that is when there is no `Arc` pointer left. A `Weak` pointer are typically created by calling `Arc::downgrade` on a shared reference of `Arc`.

The challenge for verifying `Arc` and `Weak` in a relaxed memory model is that they involve two tightly coupled atomic locations. As multi-location invariants are in general unsound for relaxed memory models, we need to use separate GPFSL single-location invariants for each counter and at the same time maintain their relation. This is a known challenge, as has been observed by GPS (Turon et al. [2014]). The general solution is to construct ghost states and/or ghost tokens to encode the relation between the locations and prevent their invariants from breaking the relation. We were able to design ghost states for the two counters of `Arc` and `Weak` without much difficulty.

`Arc<T>` supports temporary borrows of the underlying content The second challenge involves the support to temporarily reclaim full ownership of the underlying content when the thread knows it is the unique `Arc` and `Weak` pointers. The functions `Arc::get_mut` and `Arc::make_mut` provide these capabilities. In particular, `get_mut` (Fig. 29) checks that the thread owns the unique `Arc` and `Weak` pointers in two steps (in `is_unique`):

- First, in line 3 (Fig. 29), it acquires a "lock" on the `Weak` counter to make sure that there is no other `Weak` pointers, so that there cannot be other `Arc` pointers created by `upgrade`-ing.

```
1  fn is_unique(&mut self) -> bool {
2    // lock the weak pointer count if we appear to be the sole weak pointer holder.
3    if self.inner().weak.compare_exchange(1, usize::MAX, Acquire, Relaxed).is_ok() {
4      let unique = self.inner().strong.load(Relaxed) == 1;
5
6      self.inner().weak.store(1, Release); // release the lock
7      unique
8    } else { false }
9  }
10 fn get_mut(this: &mut Self) -> Option<&mut T> {
11   if this.is_unique() {
12     unsafe { Some(&mut this.ptr.as_mut().data) }
13   } else { None }
14 }
15 fn drop(&mut self) {
16   if self.inner().strong.fetch_sub(1, Release) != 1 {
17     return;
18   } ...
19 }
```

Figure 29: Rust's implementation (excerpt) of `Arc::get_mut` and `Arc::drop`.

- Second, in line 4, it reads the `Arc` counter and check if the value read from the counter is 1.

If the value read in line 4 is 1, `get_mut` concludes that thread owns the unique `Arc` and `Weak` pointers, and gives the thread temporary full access to the underlying content with type `&mut T` (line 12). Otherwise, it releases the lock on the `Weak` counter (line 6) and fails. `Arc::make_mut` also follows the similar pattern of `Arc::get_mut`, but the targets are reversed: it first acquires a "lock" on the `Arc` counter and then reads the `Weak` counter.

It is not obvious at all how to justify to the correctness of `Arc::get_mut` and `Arc::make_mut`. Unfortunately here we cannot discuss the proofs in detail here. A superficial explanation would be that this is the combined result of multiple synchronization points scattered in the code of `Arc` and `Weak`. In other words, their correctness depends on tracking global information across multiple functions of `Arc` and `Weak` (`clone`'s, `drop`'s, `downgrade`, and `upgrade`). For example, in the case of `Arc::get_mut`, the combined information lets the thread know that it has observed all the creation of any possible `Arc` pointers, either by `clone`-ing or `upgrade`-ing. Since every creation of an `Arc` pointer must increment the `Arc` counter by 1, this means that the thread has observed all the increments to the `Arc` counter. So when thread reads 1 from the `Arc` counter, it must be the case that all other `Arc` pointers have been `drop`-ed (each `drop` decrements the counter by 1), and the thread must be owning the last `Arc` pointer. It is then safe to "trade" the ownership of the `Arc` pointer to get temporary access to the underlying content.

For the verification, we design several ghost states to track the history of actions by the `Arc` and `Weak` pointers, and ultimately accumulate their effects to achieve the guarantee needed in `get_mut` and `take_mut`. The ghost states and their derived rules show that, at the read in the check's second step (line 4, Fig. 29), the thread has collected synchronized, sufficient resources to trade for the full ownership of the content *and* the counters. Such a reasoning has never been attempted before in relaxed memory logics.

5.2 Insufficient Synchronization in `get_mut`

Unfortunately, our setup was not strong enough to verify `Arc` and `Weak` without change. The two reads of the counters in the second check of `get_mut` and `make_mut` were **`rlx`** in the original code (line 4, Fig. 29), and we had to strengthen them both to **`acq`** in order to make the verification go through. The reason is that, while we managed to temporarily get the full resources out by a read, the **`rlx`** reads do not give us those resources in the current view (they are under a ∇ modality). While we conjecture that a **`rlx`** read in `make_mut` is in fact sufficient, a **`rlx`** read in `get_mut` turned out to be insufficient and we have reported the bug and the fix has been merged into Rust codebase. The following example invokes a data race when using `get_mut`:

```
1  let mut arc1 = Arc::new(0);
2  let     arc2 = Arc::clone(&arc1);
3  thread::spawn( move || { let _ : u32 = *arc2; /* drop(arc2); */ } );
4  loop { match Arc::get_mut(&mut arc1) {
5           None => {}
6           Some(m) => { *m = 1u32; return; }}}
```

In this example there are two non-atomic operations: the read of the underlying integer in line 3 (child thread) and the write to the same integer in line 6 (parent thread). The read should be safe because the child thread owns `arc2`, thus the underlying integer is shared and *immutable*. The write should be safe because `get_mut` guarantees that the parent thread owns the unique `Arc` pointer (`arc1`) and should temporarily gain full access to the non-atomic integer. This can only happens after the child thread finishes and `arc2` has been dropped. However, the two non-atomic operations constitute a data race by C11 standard, because neither one happens-before the other. More specifically, in line 3 of the child thread, when `arc2` goes out of scope, it will be destructed by `Arc::drop`, which uses a release (**`rel`**) RMW (see the code at line 16, Fig. 29). This release RMW will be read by `get_mut` (line 4, Fig. 29) in the parent thread (line 4). If this read had been **`acq`**, then there would have been a release-acquire synchronization between the release RMW of `drop` and the acquire read of `get_mut`, and the non-atomic read of the child thread would have been guaranteed to happen-before the non-atomic write of the parent thread. However, the read was **`rlx`**, thus no happen-before relationship can be established between the two non-atomic operations.

References

M. Doko and V. Vafeiadis. A program logic for C11 memory fences. In *VMCAI*, LNCS, pages 413–430. Springer, 2016.

M. Doko and V. Vafeiadis. Tackling real-life relaxed concurrency with FSL++. In *ESOP*, 2017.

R. Jung, J.-H. Jourdan, R. Krebbers, and D. Dreyer. RustBelt: Securing the foundations of the Rust programming language – technical appendix and coq development, 2017. https://plv.mpi-sws.org/rustbelt/popl18/.

J.-O. Kaiser, H.-H. Dang, D. Dreyer, O. Lahav, and V. Vafeiadis. Strong logic for weak memory: Reasoning about release-acquire consistency in Iris. In *ECOOP*, LIPIcs, pages 17:1–17:29, 2017.

O. Lahav, V. Vafeiadis, J. Kang, C.-K. Hur, and D. Dreyer. Repairing sequential consistency in C/C++11. In *PLDI*, 2017.

A. Turon, V. Vafeiadis, and D. Dreyer. GPS: Navigating weak memory with ghosts, protocols, and separation. In *OOPSLA*, pages 691–707. ACM, 2014.