Ghost Signals: Verifying Termination of Busy Waiting (Technical Report)

Tobias Reinhard and Bart Jacobs

imec-DistriNet Research Group, KU Leuven, Belgium
{tobias.reinhard,bart.jacobs}@kuleuven.be

May 20, 2021

Abstract

In this work we propose a separation logic to verify termination of busy-waiting for arbitrary events through so-called *ghost signals*.

Contents

List of Figures

1 Universe

Throughout this work we assume the existence of the following sets:

- $\mathcal{X}$: An infinite set of program variables.

- $\mathcal{L}ocs$: An infinite set of heap locations.
- $\mathcal{L}ocs^{\mathsf{G}}$: An infinite set of ghost locations.
- $\mathcal{L}evs$, $<_{\mathsf{L}}$: An infinite, well-founded partially ordered set of levels.
- Δ, $<_{\Delta}$: An infinite, well-founded partially ordered set of degrees.
- $\mathcal{ID}$: An infinite set of IDs.
- Θ: An infinite, totally ordered and well-founded set of thread IDs.
- *Values*: A set of values which includes:
  - A unit value $\mathtt{tt} \in \mathit{Values}$
  - Booleans $\mathbb{B} = \{\mathsf{True}, \mathsf{False}\} \subset \mathit{Values}$
  - Heap locations $\mathcal{L}ocs \subset \mathit{Values}$
- $\mathit{Values}^{\mathsf{G}}$: A set of ghost values.
- *Ops*: A set of operations (i.e. partial functions) on values.

We denote program variables by x, heap locations by ℓ, ghost locations by $\widehat{\ell}$, levels by L, degrees by δ, IDs by id, thread IDs by θ, values by v, ghost values by $\widehat{v}$, boolean by b and operations by op.

2 General

Definition 2.1 (Projections)**.** *For any Cartesian product $C = \prod_{i \in I} A_i$ and any index $k \in I$, we denote the k^{th} projection by $\pi_k^C : \prod_{i \in I} A_i \to A_k$. We define*

$$\pi_k^C((a_i)_{i \in I}) \quad := \quad a_k.$$

In case the domain C is clear from the context, we write π_k instead of π_k^C.

In the following we define our notion of *bags*, in the literature also referred to as *multisets*.

Definition 2.2 (Bags)**.** *For any set X we define the set of bags $\mathit{Bags}(X)$ and the set of finite bags $\mathit{Bags}_{\mathsf{fin}}(X)$ over X as*

$$\begin{array}{lcl} \mathit{Bags}(X) & := & X \to \mathbb{N}, \\ \mathit{Bags}_{\mathsf{fin}}(X) & := & \{B \in \mathit{Bags}(X) \ \mid \ \{x \in B \mid B(x) > 0\} \text{ finite}\}. \end{array}$$

We define union and subtraction of bags as

$$\begin{array}{lcl} (B_1 \uplus B_2)(x) & := & B_1(x) + B_2(x), \\ (B_1 \setminus B_2)(x) & := & \max(0,\ B_1(x) - B_2(x)). \end{array}$$

For finite bags where the domain is clear from the context, we define the following set-like notation:

$$\begin{array}{lcl}
\emptyset & := & x \mapsto 0, \\
\{\!\!\{x\}\!\!\} & := & \begin{cases} x & \mapsto & 1 \\ y & \mapsto & 0 \quad \textit{for } y \neq x, \end{cases} \\
\{\!\!\{x_1, \ldots, x_n\}\!\!\} & := & \biguplus_{i=1}^{n} \{\!\!\{x_i\}\!\!\}.
\end{array}$$

We define the following set-like notations for element and subset relationship:

$$\begin{array}{lcl}
x \in B & \Leftrightarrow & B(x) > 0, \\
B_1 \subseteq B_2 & \Leftrightarrow & \forall x \in B_1.\ B_1(x) \leq B_2(x), \\
B_1 \subset B_2 & \Leftrightarrow & \exists C \subseteq B_1.\ C \neq \emptyset \ \wedge\ B_1 = B_2 \setminus C.
\end{array}$$

For any bag $B \in Bags(X)$ and predicate $P \subseteq X$ we define the following refinement:

$$\{\!\!\{x \in B \mid P(x)\}\!\!\} \quad := \quad \begin{cases} x & \mapsto & B(x) & \textit{if } P(x), \\ x & \mapsto & 0 & \textit{otherwise.} \end{cases}$$

Definition 2.3 (Disjoint Union)**.** *Let A, B be sets. We define their disjoint union as*

$$A \sqcup B := A \cup B$$

if $A \cap B = \emptyset$ and leave it undefined otherwise.

3 Syntax

Definition 3.1. *We define the sets of commands Cmds and expressions Exps according to the syntax presented in Figure 1.*

We define $c\ ;\ c'$ as shorthand for **let** $x := c$ **in** c' *where x does not occur free in c' but let $\cdot ; \cdot$ bind stronger. Further, we define $e \neq e'$ as abbreviation for $\neg(e = e')$.*

4 Example

Figure 2 presents the example program we plan to verify. For this example we let *Values* include natural numbers.

5 Resources

In this section we define physical resources. We will use the physical resources to define the semantics of our programming language.

$$v \in \mathit{Values} \quad x \in \mathcal{X} \quad op \in \mathit{Ops}$$

$$\begin{array}{lcl}
e \in \mathit{Exps} & ::= & x \;\mid\; v \;\mid\; e = e \;\mid\; \neg e \;\mid\; op(\bar{e}) \\
c \in \mathit{Cmds} & ::= & \textbf{while } c \textbf{ do skip} \;\mid\; \textbf{fork } c \;\mid \\
 & & \textbf{let } x := c \textbf{ in } c \;\mid\; \textbf{if } c \textbf{ then } c \;\mid \\
 & & \textbf{cons}(e) \;\mid\; [e] \;\mid\; [e] := e \;\mid \\
 & & \textbf{new_mutex} \;\mid\; \textbf{acquire } e \;\mid\; \textbf{release } e \;\mid \\
 & & e \;\mid \\
 & & \textbf{consumeItPerm} \qquad\qquad \text{intermediate representation}
\end{array}$$

Figure 1: Syntax.

```
let x := cons(0) in
let m := new_mutex in
fork (while (acquire m;
             let y := [x] in
             release m;
             y = 0)
      do skip);
acquire m;
[x] := 1;
release m
```

Figure 2: Example Program.

Definition 5.1 (Physical Resources)**.** *We define the set of physical resources $\mathcal{R}^{\mathsf{phys}}$ syntactically as follows:*

$$r^{\mathsf{p}} \in \mathcal{R}^{\mathsf{phys}} \quad ::= \quad \ell \mapsto v \;\;|\;\; \mathrm{unlocked}_{\mathsf{pRes}}(\ell) \;\;|\;\; \mathrm{locked}_{\mathsf{pRes}}(\ell)$$

$$\ell \in \mathcal{L}ocs \quad v \in \mathit{Values}$$

Definition 5.2 (Physical Heaps)**.** *We define the set of physical heaps as*

$$\mathit{Heaps}^{\mathsf{phys}} \;:=\; \mathcal{P}_{\mathsf{fin}}(\mathcal{R}^{\mathsf{phys}})$$

and the function $\mathsf{locs}_{\mathsf{pRes}} : \mathit{Heaps}^{\mathsf{phys}} \to \mathcal{P}_{\mathsf{fin}}(\mathcal{L}ocs)$ mapping physical heaps to the sets of allocated heap locations as

$$\begin{array}{lcl} \mathsf{locs}_{\mathsf{pRes}}(h) & := & \{\ell \in \mathcal{L}ocs \;\;|\;\; \mathrm{unlocked}_{\mathsf{pRes}}(\ell) \in h \;\vee\; \mathrm{locked}_{\mathsf{pRes}}(\ell) \in h \;\vee \\ & & \qquad\qquad\qquad \exists v \in \mathit{Values}.\ \ell \mapsto v \in h\} \end{array}$$

We denote physical heaps by h.

6 Semantics

Definition 6.1 (Evaluation of Closed Expressions)**.** *We define a partial evaluation function $[\![\cdot]\!] : \mathit{Exps} \rightharpoonup \mathit{Values}$ on expressions by recursion on the structure of expressions as follows:*

$$\begin{array}{lcll} [\![v]\!] & := & v & \textit{if} \;\; v \in \mathit{Values} \\ [\![e = e']\!] & := & \mathsf{True} & \textit{if} \;\; [\![e]\!] = [\![e']\!] \neq \bot \\ [\![e = e']\!] & := & \mathsf{False} & \textit{if} \;\; [\![e]\!] \neq [\![e']\!] \;\wedge\; [\![e]\!] \neq \bot \;\wedge\; [\![e']\!] \neq \bot \\ [\![\neg e]\!] & := & \mathsf{False} & \textit{if} \;\; [\![e]\!] = \mathsf{True} \\ [\![\neg e]\!] & := & \mathsf{True} & \textit{if} \;\; [\![e]\!] = \mathsf{False} \\ [\![e]\!] & := & \bot & \textit{otherwise} \end{array}$$

We identify closed expressions e with their ascribed value $[\![e]\!]$.

Definition 6.2 (Evaluation Context)**.** *We define the set of evaluation contexts EvalCtxts as follows:*

$$E \in \mathit{EvalCtxts} \quad ::= \quad \textbf{if } \Box \textbf{ then } c \;\;|\;\; \textbf{let } x := \Box \textbf{ in } c$$

$$c \in \mathit{Cmds} \quad x \in \mathcal{X}$$

For any $c \in \mathit{Cmds}$ and $E \in \mathit{EvalCtxts}$, we define $E[c] := E[c/\Box]$.

Note that for every $c \in \mathit{Cmds}, E \in \mathit{EvalCtxts}$, we have $E[c] \in \mathit{Cmds}$.

Definition 6.3 (Single Thread Reduction Relation)**.** *We define a reduction relation $\rightsquigarrow_{\mathsf{st}}$ for single threads according to the rules presented in Figure 3. A reduction step has the form*

$$h, c \rightsquigarrow_{\mathsf{st}} h', c', T$$

for a set of forked threads $T \subset \mathit{Cmds}$ with $|T| \leq 1$.

For simplicity of notation, we omit T if it is clear from the context that no thread is forked and $T = \emptyset$.

Note that we do not provide a reduction rule for **consumeItPerm**, since we only use it as an intermediate representation for the annotated reduction relation presented in Section 9.

Definition 6.4 (Thread Pools)**.** *We define the set of thread pools* $\mathcal{TP}$ *as the set of finite partial functions mapping thread IDs to threads:*

$$\mathcal{TP} \quad := \quad \Theta \rightharpoonup_{\mathsf{fin}} (Cmds \cup \{\mathsf{term}\}).$$

The symbol term *represents a terminated thread. We denote thread pools by* P*, thread IDs by* θ *and the empty thread pool by* $\emptyset_{\mathsf{tp}}$*, i.e.,*

$$\begin{aligned} &\emptyset_{\mathsf{tp}} \;:\; \Theta \rightharpoonup_{\mathsf{fin}} (Cmds \cup \{\mathsf{term}\}),\\ &\mathsf{dom}(\emptyset_{\mathsf{tp}}) \;=\; \emptyset. \end{aligned}$$

We define the operation $+_{\mathsf{tp}} : \mathcal{TP} \times \{C \subset Cmds \mid |C| \leq 1\} \rightarrow \mathcal{TP}$ *as follows:*

$$\begin{array}{lcl} P +_{\mathsf{tp}} \emptyset & := & P,\\ P +_{\mathsf{tp}} \{c\} & := & P[\theta_{new} := c] \quad \textit{for} \quad \theta_{new} := \min(\Theta \setminus \mathsf{dom}(P)). \end{array}$$

Definition 6.5 (Thread Pool Reduction Relation)**.** *We define a thread pool reduction relation* $\rightsquigarrow_{\mathsf{tp}}$ *according to the rules presented in Figure 4. A reduction step has the form*

$$h, P \overset{\theta}{\rightsquigarrow}_{\mathsf{tp}} h', P'.$$

Definition 6.6 (Reduction Sequence)**.** *Let* $(h_i)_{i\in\mathbb{N}}$ *and* $(P_i)_{i\in\mathbb{N}}$ *be infinite sequences of physical heaps and thread pools, respectively.*

We call $(h_i, P_i)_{i\in\mathbb{N}}$ a reduction sequence *if there exists a sequence of thread IDs* $(\theta_i)_{i\in\mathbb{N}}$ *such that*

$$h_i, P_i \overset{\theta_i}{\rightsquigarrow}_{\mathsf{tp}} h_{i+1}, P_{i+1}$$

holds for every $i \in \mathbb{N}$*.*

Definition 6.7 (Fairness)**.** *We call a reduction sequence* $(h_i, P_i)_{i\in\mathbb{N}}$ fair *iff for all* $i \in \mathbb{N}$ *and* $\theta \in \mathsf{dom}(P_i)$ *with* $P_i(\theta) \neq \mathsf{term}$ *there exists some* $k \geq i$ *with*

$$h_k, P_k \overset{\theta}{\rightsquigarrow}_{\mathsf{tp}} h_{k+1}, P_{k+1}.$$

7 Assertions

Definition 7.1 (Fractions)**.** *We define the set of fractions as*

$$\mathcal{F} \quad := \quad \{f \in \mathbb{Q} \mid 0 < f \leq 1\}.$$

Definition 7.2 (Thread Phase IDs)**.** *We define the set of thread phase literals as*

$$\mathcal{T} \quad := \quad \{\mathit{Forker}, \mathit{Forkee}\}.$$

We call a finite sequence of thread phase literals a phase ID *and denote it by* $\tau \in \mathcal{T}^*$*. We write* $\tau_1 \sqsubseteq \tau_2$ *to express that* τ_1 *is a (non-strict) prefix of* τ_2*.*

$$\textsc{ST-Red-EvalCtxt}\quad \frac{h, c \rightsquigarrow_{\mathsf{st}} h', c', T}{h, E[c] \rightsquigarrow_{\mathsf{st}} h', E[c'], T}$$

$$\textsc{ST-Red-Fork}\quad h, \textbf{fork } c \rightsquigarrow_{\mathsf{st}} h, \mathsf{tt}, \{c\}$$

(a) Basic Constructs.

$$\textsc{ST-Red-While}\quad h, \textbf{while } c \textbf{ do skip} \rightsquigarrow_{\mathsf{st}} h, \textbf{if } c \textbf{ then while } c \textbf{ do skip}$$

$$\textsc{ST-Red-IfTrue}\quad h, \textbf{if } \mathsf{True} \textbf{ then } c \rightsquigarrow_{\mathsf{st}} h, c$$

$$\textsc{ST-Red-IfFalse}\quad h, \textbf{if } \mathsf{False} \textbf{ then } c \rightsquigarrow_{\mathsf{st}} h, \mathsf{tt}$$

$$\textsc{ST-Red-Let}\quad h, \textbf{let } x := v \textbf{ in } c \rightsquigarrow_{\mathsf{st}} h, c[v/x]$$

(b) Control Structures.

$$\textsc{ST-Red-Cons}\quad \frac{\ell \notin \mathsf{locs}_{\mathsf{pRes}}(h)}{h, \textbf{cons}(v) \rightsquigarrow_{\mathsf{st}} h \cup \{\ell \mapsto v\}, \ell}$$

$$\textsc{ST-Red-ReadHeapLoc}\quad \frac{\ell \mapsto v \in h}{h, [\ell] \rightsquigarrow_{\mathsf{st}} h, v}$$

$$\textsc{ST-Red-Assign}\quad h \sqcup \{\ell \mapsto v'\}, [\ell] := v \rightsquigarrow_{\mathsf{st}} h \sqcup \{\ell \mapsto v\}, \mathsf{tt}$$

(c) Heap Access.

$$\textsc{ST-Red-NewMutex}\quad \frac{\ell \notin \mathsf{locs}_{\mathsf{pRes}}(h)}{h, \textbf{new\_mutex} \rightsquigarrow_{\mathsf{st}} h \cup \{\text{unlocked}_{\mathsf{pRes}}(\ell)\}, \ell}$$

$$\textsc{ST-Red-Acquire}\quad h \sqcup \{\text{unlocked}_{\mathsf{pRes}}(\ell)\}, \textbf{acquire } \ell \rightsquigarrow_{\mathsf{st}} h \sqcup \{\text{locked}_{\mathsf{pRes}}(\ell)\}, \mathsf{tt}$$

$$\textsc{ST-Red-Release}\quad h \sqcup \{\text{locked}_{\mathsf{pRes}}(\ell)\}, \textbf{release } \ell \rightsquigarrow_{\mathsf{st}} h \sqcup \{\text{unlocked}_{\mathsf{pRes}}(\ell)\}, \mathsf{tt}$$

(d) Mutexes.

Figure 3: Single thread reduction rules.

$$\frac{P(\theta) = c \qquad h, c \rightsquigarrow_{\mathsf{st}} h', c', T}{h, P \overset{\theta}{\rightsquigarrow}_{\mathsf{tp}} h', P[\theta := c'] +_{\mathsf{tp}} T}\ \textsc{TP-Red-Lift} \qquad \frac{P(\theta) = v}{h, P \overset{\theta}{\rightsquigarrow}_{\mathsf{tp}} h, P[\theta := \mathsf{term}]}\ \textsc{TP-Red-Term}$$

Figure 4: Thread pool reduction rules.

Definition 7.3. *We define the sets of* ghost signals $\mathcal{S}$, obligations $\mathcal{O}$, wait permission Ω *and* iteration permissions Λ *as follows:*

$$\begin{array}{rcl} \mathcal{S} & := & \mathcal{ID} \times \mathcal{L}evs, \\ \mathcal{O} & := & (\mathcal{L}ocs \cup \mathcal{ID}) \times \mathcal{L}evs, \\ \Omega & := & \mathcal{T}^* \times \mathcal{ID} \times \Delta, \\ \Lambda & := & \mathcal{T}^* \times \Delta. \end{array}$$

We denote ghost signals by s, obligations by o, and bags of obligations by O. For convenience of notation we define the selector function:

$$(id, L).\mathsf{id} \quad := \quad L.$$

Definition 7.4 (Assertions)**.** *We define the set of* assertions $\mathcal{A}$ *according to the syntax presented in Figure 5.*[1] *Further, we define implication and equivalence as the usual abbreviations:*

$$\begin{array}{rcl} a_1 \rightarrow a_2 & := & \neg a_1 \vee a_2, \\ a_1 \leftrightarrow a_2 & := & (a_1 \rightarrow a_2) \wedge (a_2 \rightarrow a_1). \end{array}$$

Let $(a(i))_{i \in I}$ be a family of assertions indexed by some set I. We define quantification over I as the following abbreviations:

$$\begin{array}{rcl} \exists i \in I.\, a(i) & := & \bigvee \{a(i) \mid i \in I\}, \\ \forall i \in I.\, a(i) & := & \neg \exists i \in I.\, \neg a(i). \end{array}$$

We omit the index set I when its choice becomes clear from the context and write $\exists i.\, a(i)$ and $\forall i.\, a(i)$ instead of $\exists i \in I.\, a(i)$ and $\forall i \in I.\, a(i)$, respectively.

Definition 7.5 (Logical Resources)**.** *We define the set of logical resources $\mathcal{R}^{\mathsf{log}}$ syntactically as follows:*

$$\begin{array}{rcl} r^{\mathsf{l}} \in \mathcal{R}^{\mathsf{log}} & ::= & \ell \mapsto v \;\mid\; \widehat{\ell} \mapsto \widehat{v} \;\mid\; \mathrm{signal}_{\mathsf{lRes}}((id, L), b) \;\mid \\ & & \mathrm{uninit}_{\mathsf{lRes}}(\ell) \;\mid\; \mathrm{mutex}_{\mathsf{lRes}}((\ell, L), a) \;\mid\; \mathrm{locked}_{\mathsf{lRes}}((\ell, L), a, f) \;\mid \\ & & \mathrm{phase}_{\mathsf{lRes}}(\tau) \;\mid\; \mathrm{obs}_{\mathsf{lRes}}(O) \;\mid\; \mathrm{wperm}_{\mathsf{lRes}}(\tau, id, \delta) \;\mid \\ & & \mathrm{itperm}_{\mathsf{lRes}}(\tau, \delta) \end{array}$$

[1] That is, we define $\mathcal{A}$ as the least fixpoint of F where $F(A) = \{\mathsf{True}, \mathsf{False}\} \cup \{\neg a \mid a \in A\} \cup \{a_1 \wedge a_2 \mid a_1, a_2 \in A\} \cup \cdots \cup \{\bigvee A' \mid A' \subseteq A\} \cup \ldots$. Since F is a monotonic function over a complete lattice, it has a least fixpoint according to the Knaster-Tarski theorem [Tarski(1955)].

$$
\begin{array}{rcl}
a \in \mathcal{A} & := & \mathsf{True} \;\mid\; \mathsf{False} \;\mid\; \neg a \;\mid \\
& & a \wedge a \;\mid\; a \vee a \;\mid\; a * a \;\mid\; [f]\ell \mapsto v \;\mid\; [f]\widehat{\ell} \mapsto \widehat{v} \;\mid \\
& & \bigvee A \;\mid \\
& & [f]\mathsf{uninit}(\ell) \;\mid \\
& & [f]\mathsf{mutex}((\ell, L), a) \;\mid\; [f]\mathsf{locked}((\ell, L), a, f) \;\mid \\
& & [f]\mathsf{signal}((id, L), b) \;\mid \\
& & \mathsf{phase}(\tau) \;\mid\; \mathsf{obs}(O) \;\mid\; \mathsf{wperm}(\tau, id, \delta) \;\mid\; \mathsf{itperm}(\tau, \delta)
\end{array}
$$

$$
f \in \mathcal{F} \quad v \in \mathit{Values} \quad \widehat{v} \in \mathit{Values}^{\mathsf{G}} \quad \ell \in \mathcal{L}ocs \quad \widehat{\ell} \in \mathcal{L}ocs^{\mathsf{G}}
$$

$$
L \in \mathcal{L}evs \quad id \in \mathcal{ID} \quad b \in \mathbb{B} = \{\mathsf{True}, \mathsf{False}\} \quad \delta \in \Delta
$$

$$
A \subseteq \mathcal{A} \quad O \in \mathit{Bags}(\mathcal{O}) \quad \tau \in \mathcal{T}^*
$$

Figure 5: Assertion syntax.

Further, we define the functions $\mathsf{getHLocs}_{\mathsf{lRes}} : \mathcal{R}^{\mathsf{log}} \to \mathcal{L}ocs$ *and* $\mathsf{getGLocs}_{\mathsf{lRes}} : \mathcal{R}^{\mathsf{log}} \to \mathcal{P}_{\mathsf{fin}}(\mathcal{L}ocs^{\mathsf{G}})$ *mapping logical resources to their respective (either empty or singleton) set of involved heap locations and ghost locations, respectively, as*

$$
\begin{array}{lcll}
\mathsf{getHLocs}_{\mathsf{lRes}}(\ell \mapsto v) & := & \{\ell\}, & \\
\mathsf{getHLocs}_{\mathsf{lRes}}(\mathrm{uninit}_{\mathsf{lRes}}(\ell)) & := & \{\ell\}, & \\
\mathsf{getHLocs}_{\mathsf{lRes}}(\mathrm{mutex}_{\mathsf{lRes}}((\ell, L), a)) & := & \{\ell\}, & \\
\mathsf{getHLocs}_{\mathsf{lRes}}(\mathrm{locked}_{\mathsf{lRes}}((\ell, L), a, f)) & := & \{\ell\}, & \\
\mathsf{getHLocs}_{\mathsf{lRes}}(\_) & := & \emptyset & \textit{otherwise}, \\
 & & & \\
\mathsf{getGLocs}_{\mathsf{lRes}}(\widehat{\ell} \mapsto \widehat{v}) & := & \{\widehat{\ell}\}, & \\
\mathsf{getGLocs}_{\mathsf{lRes}}(\_) & := & \emptyset & \textit{otherwise.}
\end{array}
$$

Definition 7.6 (Mutexes)**.** *We define the set of mutexes as* $\mathcal{M} := \mathcal{L}ocs \times \mathcal{L}evs$ *and denote mutexes by* m*. For convenience of notation we define the selector function*

$$
(\ell, L).\mathsf{loc} \quad := \quad \ell.
$$

Definition 7.7 (Logical Heaps)**.** *We define the set of logical heaps as*

$$
\mathit{Heaps}^{\mathsf{log}} \quad := \quad \mathcal{R}^{\mathsf{log}} \to \{q \in \mathbb{Q} \mid q \geq 0\}.
$$

We define the empty logical heap $\emptyset_{\mathsf{log}}$ *as the constant zero function*

$$
\emptyset_{\mathsf{log}} : r^{\mathsf{l}} \mapsto 0.
$$

We denote logical heaps by H*, point-wise addition by* $+$ *and multiplication with non-negative rationals by* $\cdot$*, i.e.,*

$$
\begin{array}{lcl}
(H_1 + H_2)(r^{\mathsf{l}}) & := & H_1(r^{\mathsf{l}}) + H_2(r^{\mathsf{l}}), \\
(q \cdot H)(r^{\mathsf{l}}) & := & q \cdot (H(r^{\mathsf{l}}))
\end{array}
$$

for $q \in \mathbb{Q}$ *with* $q \geq 0$*. For convenience of notation we represent logical heaps containing finitely many resources by sets of resources and define left-associative functions* $+_{\mathsf{lh}}, -_{\mathsf{lh}} : \mathit{Heaps}^{\mathsf{log}} \to \mathcal{R}^{\mathsf{log}} \to \mathit{Heaps}^{\mathsf{log}}$ *as follows*

$$\begin{array}{lcl} \{r^{\mathsf{l}}_1, \ldots, r^{\mathsf{l}}_n\} & := & \begin{cases} r^{\mathsf{l}}_i & \mapsto 1 \\ x & \mapsto 0 \quad \text{if } x \notin \{r^{\mathsf{l}}_1, \ldots, r^{\mathsf{l}}_n\}, \end{cases} \\ H +_{\mathsf{lh}} r^{\mathsf{l}} & := & H[r^{\mathsf{l}} := H(r^{\mathsf{l}}) + 1], \\ H -_{\mathsf{lh}} r^{\mathsf{l}} & := & H[r^{\mathsf{l}} := \max(0,\, H(r^{\mathsf{l}}) - 1)]. \end{array}$$

We give $\cdot$ *a higher precedence than* $+$*,* $+_{\mathsf{lh}}$ *and* $-_{\mathsf{lh}}$*.*

Further, we define the function $\mathsf{getGLocs}_{\mathsf{lh}} : \mathit{Heaps}^{\mathsf{log}} \to \mathcal{P}(\mathcal{L}ocs^{\mathsf{G}})$ *mapping logical heaps to their respective set of allocated ghost locations as*

$$\mathsf{getGLocs}_{\mathsf{lh}}(H) \quad := \quad \bigcup_{\substack{r^{\mathsf{l}} \in \mathcal{R}^{\mathsf{log}} \\ H(r^{\mathsf{l}}) > 0}} \mathsf{getGLocs}_{\mathsf{lRes}}(r^{\mathsf{l}}).$$

We call a logical heap H complete *and write* $\mathsf{complete}_{\mathsf{lh}}(H)$ *if it contains exactly one obligations chunk and exactly one phase chunk, i.e., if there exist a bag of obligations* O *and a phase ID* τ *with* $H(\mathrm{obs}_{\mathsf{lRes}}(O)) = 1$ *and* $H(\mathrm{phase}_{\mathsf{lRes}}(\tau)) = 1$ *and if there do not exist any bag of obligations* O' *nor any phase ID* τ' *with (i)* $O \neq O'$ *and* $H(\mathrm{obs}_{\mathsf{lRes}}(O')) > 0$ *or with (ii)* $\tau \neq \tau'$ *and* $H(\mathrm{phase}_{\mathsf{lRes}}(\tau')) > 0$*.*

We call a logical heap H finite *and write* $\mathsf{finite}_{\mathsf{lh}}(H)$ *if it contains only finitely many resources, i.e., if the set* $\{r^{\mathsf{l}} \in \mathcal{R}^{\mathsf{log}} \mid H(r^{\mathsf{l}}) > 0\}$ *is finite.*

We call a logical heap H consistent *and write* $\mathsf{consistent}_{\mathsf{lh}}(H)$ *if (i) it contains only full phase, obligations, wait and iteration permission chunks, i.e., if*

$$\begin{array}{lcl} H(\mathrm{phase}_{\mathsf{lRes}}(\tau)) & \in & \mathbb{N}, \\ H(\mathrm{obs}_{\mathsf{lRes}}(O)) & \in & \mathbb{N}, \\ H(\mathrm{wperm}_{\mathsf{lRes}}(\tau, id, \delta)) & \in & \mathbb{N}, \\ H(\mathrm{itperm}_{\mathsf{lRes}}(\tau, \delta)) & \in & \mathbb{N} \end{array}$$

holds for all $\tau \in \mathcal{T}^*$*,* $O \in \mathit{Bags}(\mathcal{O})$*,* $id \in \mathcal{ID}$ *and* $\delta \in \Delta$ *and if (ii) heap locations and ghost locations are unique in* H*, i.e., if there are no* $r^{\mathsf{l}}_1, r^{\mathsf{l}}_2 \in \mathcal{R}^{\mathsf{log}}$ *with* $r^{\mathsf{l}}_1 \neq r^{\mathsf{l}}_2$*,* $H(r^{\mathsf{l}}_1) > 0$*,* $H(r^{\mathsf{l}}_2) > 0$ *and with* $\mathsf{getHLocs}_{\mathsf{lRes}}(r^{\mathsf{l}}_1) \cap \mathsf{getHLocs}_{\mathsf{lRes}}(r^{\mathsf{l}}_2) \neq \emptyset$ *or* $\mathsf{getGLocs}_{\mathsf{lRes}}(r^{\mathsf{l}}_1) \cap \mathsf{getGLocs}_{\mathsf{lRes}}(r^{\mathsf{l}}_2) \neq \emptyset$*.*

To simplify the specification of logical heaps containing only a single obligations chunk with certain properties, we introduce the abbreviation

$$(H.\mathsf{obs} = O) \quad := \quad (\mathsf{complete}_{\mathsf{lh}}(H) \;\wedge\; H(\mathrm{obs}_{\mathsf{lRes}}(O)) = 1).$$

Definition 7.8 (Assertion Model Relation)**.** *We define a model relation* $\vDash_{\mathsf{A}} \subset \mathit{Heaps}^{\mathsf{log}} \times \mathcal{A}$ *for assertions by recursion on the structure of assertions according to the rules presented in Figure 6. We write* $H \vDash_{\mathsf{A}} a$ *to express that logical heap* H *models assertion* a *and* $H \nvDash_{\mathsf{A}} a$ *to express that* $H \vDash_{\mathsf{A}} a$ *does not hold.*

$$
\begin{array}{ll}
H \vDash_{\mathsf{A}} \mathsf{True} & \\
H \nvDash_{\mathsf{A}} \mathsf{False} & \\
H \vDash_{\mathsf{A}} \neg a & \text{if} \quad H \nvDash_{\mathsf{A}} a \\
H \vDash_{\mathsf{A}} a_1 \wedge a_2 & \text{if} \quad H \vDash_{\mathsf{A}} a_1 \;\wedge\; H \vDash_{\mathsf{A}} a_2 \\
H \vDash_{\mathsf{A}} a_1 \vee a_2 & \text{if} \quad H \vDash_{\mathsf{A}} a_1 \;\vee\; H \vDash_{\mathsf{A}} a_2 \\
H \vDash_{\mathsf{A}} a_1 * a_2 & \text{if} \quad \exists H_1, H_2 \in \mathit{Heaps}^{\mathsf{log}}. \\
& \qquad H = H_1 + H_2 \;\wedge \\
& \qquad H_1 \vDash_{\mathsf{A}} a_1 \;\wedge\; H_2 \vDash_{\mathsf{A}} a_2 \\
H \vDash_{\mathsf{A}} [f]\ell \mapsto v & \text{if} \quad H(\ell \mapsto v) \geq f \\
H \vDash_{\mathsf{A}} [f]\widehat{\ell} \mapsto \widehat{v} & \text{if} \quad H(\widehat{\ell} \mapsto \widehat{v}) \geq f \\
H \vDash_{\mathsf{A}} \bigvee A & \text{if} \quad \exists a \in A.\ H \vDash_{\mathsf{A}} a \\
H \vDash_{\mathsf{A}} [f]\mathsf{uninit}(\ell) & \text{if} \quad H(\mathrm{uninit}_{\mathsf{lRes}}(\ell)) \geq f \\
H \vDash_{\mathsf{A}} [f]\mathsf{mutex}(m, P) & \text{if} \quad H(\mathrm{mutex}_{\mathsf{lRes}}(m, P)) \geq f \\
H \vDash_{\mathsf{A}} [f]\mathsf{locked}(m, P, f_u) & \text{if} \quad H(\mathrm{locked}_{\mathsf{lRes}}(m, P, f_u)) \geq f \\
H \vDash_{\mathsf{A}} [f]\mathsf{signal}(s, b) & \text{if} \quad H(\mathrm{signal}_{\mathsf{lRes}}(s, b)) \geq f \\
H \vDash_{\mathsf{A}} \mathsf{phase}(\tau) & \text{if} \quad H(\mathrm{phase}_{\mathsf{lRes}}(\tau)) \geq 1 \\
H \vDash_{\mathsf{A}} \mathsf{obs}(O) & \text{if} \quad H(\mathrm{obs}_{\mathsf{lRes}}(O)) \geq 1 \\
H \vDash_{\mathsf{A}} \mathsf{wperm}(\tau, id, \delta) & \text{if} \quad H(\mathrm{wperm}_{\mathsf{lRes}}(\tau, id, \delta)) \geq 1 \\
H \vDash_{\mathsf{A}} \mathsf{itperm}(\tau, \delta) & \text{if} \quad H(\mathrm{itperm}_{\mathsf{lRes}}(\tau, \delta)) \geq 1
\end{array}
$$

Figure 6: Assertion model relation.

8 Proof Rules

Definition 8.1 (Level Ascriptions)**.** *We define a function* $\mathsf{lev} : (\mathcal{ID} \cup \mathcal{L}ocs) \times \mathcal{L}evs \to \mathcal{L}evs$ *as*

$$\mathsf{lev}((\_, L)) \quad := \quad L.$$

Definition 8.2 (View Shift)**.** *We define a view shift relation* $\Rrightarrow \;\subset\; \mathcal{A} \times \mathcal{A}$ *according to the rules presented in Figure 7.*

Definition 8.3 (Proof Relation)**.** *We define a proof relation* $\vdash \;\subset\; \mathcal{A} \times \mathit{Cmds} \times (\mathit{Values} \to \mathcal{A})$ *according to the rules presented in Figures 8 and 9.*

Note that our proof rules do not allow us to reason about the command **consumeItPerm**, since we only use it as an intermediate representation during reduction.

Lemma 8.4. *We can derive the proof rule presented in Figure 10.*

Proof. Trivial. □

$$\frac{\forall H.\ \mathsf{consistent}_{\mathsf{lh}}(H) \wedge H \vDash_{\mathsf{A}} A \Rightarrow H \vDash_{\mathsf{A}} B}{A \Rrightarrow B}\ \textsc{VS-SemImp}$$

$$\frac{A \Rrightarrow C \qquad C \Rrightarrow B}{A \Rrightarrow B}\ \textsc{VS-Trans}$$

$$\frac{A_1 \Rrightarrow B \qquad A_2 \Rrightarrow B}{A_1 \vee A_2 \Rrightarrow B}\ \textsc{VS-Or}$$

$$\frac{L \in \mathcal{L}evs}{\mathsf{obs}(O) \Rrightarrow \exists id.\ \mathsf{obs}(O \uplus \{\!\![(id, L)]\!\!\}) * \mathsf{signal}((id, L), \mathsf{False})}\ \textsc{VS-NewSignal}$$

$$\textsc{VS-SetSignal}\quad \mathsf{obs}(O \uplus \{\!\![s]\!\!\}) * \mathsf{signal}(s, \_) \Rrightarrow \mathsf{obs}(O) * \mathsf{signal}(s, \mathsf{True})$$

$$\frac{\delta' <_{\mathsf{\Delta}} \delta}{\mathsf{itperm}(\tau', \delta) \Rrightarrow \mathsf{wperm}(\tau', id, \delta')}\ \textsc{VS-WaitPerm}$$

$$\frac{\tau_{\mathrm{anc}} \sqsubseteq \tau \qquad \forall o \in O.\ \mathsf{lev}(s) <_{\mathsf{L}} \mathsf{lev}(o)}{\begin{array}{l}\mathsf{phase}(\tau) * \mathsf{obs}(O) * \mathsf{wperm}(\tau_{\mathrm{anc}}, s.\mathsf{id}, \delta) * \mathsf{signal}(s, b)\\ \Rrightarrow \mathsf{phase}(\tau) * \mathsf{obs}(O) * \mathsf{wperm}(\tau_{\mathrm{anc}}, s.\mathsf{id}, \delta) * \mathsf{signal}(s, b) * (\neg b \leftrightarrow \mathsf{itperm}(\tau, \delta))\end{array}}\ \textsc{VS-Wait}$$

$$\frac{\tau_{\mathrm{anc}} \sqsubseteq \tau}{\mathsf{itperm}(\tau_{\mathrm{anc}}, \delta) \Rrightarrow \mathsf{itperm}(\tau, \delta)}\ \textsc{VS-SpecItPerm}$$

$$\frac{\tau_{\mathrm{anc}} \sqsubseteq \tau}{\mathsf{wperm}(\tau_{\mathrm{anc}}, id, \delta) \Rrightarrow \mathsf{wperm}(\tau, id, \delta)}\ \textsc{VS-SpecWaitPerm}$$

$$\frac{\delta' <_{\mathsf{\Delta}} \delta \qquad N \in \mathbb{N}}{\mathsf{itperm}(\tau', \delta) \Rrightarrow \mathop{\Huge *}_{1,\ldots,N} \mathsf{itperm}(\tau', \delta')}\ \textsc{VS-WeakPerm}$$

$$\frac{L \in \mathcal{L}evs}{\mathsf{uninit}(\ell) * P \Rrightarrow \mathsf{mutex}((\ell, L), P)}\ \textsc{VS-MutInit}$$

$$\textsc{VS-NewGCell}\quad \mathsf{True} \Rrightarrow \exists \widehat{\ell}.\ \widehat{\ell} \mapsto \widehat{v}$$

$$\textsc{VS-SetGCell}\quad \widehat{\ell} \mapsto \widehat{v} \Rrightarrow \widehat{\ell} \mapsto \widehat{v}'$$

Figure 7: View shift rules.

PR-Frame

$$\frac{\vdash \{A\}\ c\ \{B\}}{\vdash \{A * F\}\ c\ \{B * F\}}$$

PR-ViewShift

$$\frac{A \Rrightarrow A' \land \mathsf{phase}(\tau) \qquad \vdash \{A'\}\ c\ \{B'\} \qquad \forall \tau'.\ \big(B' \land \mathsf{phase}(\tau') \land \tau \sqsubseteq \tau' \Rrightarrow B\big)}{\vdash \{A\}\ c\ \{B\}}$$

PR-VS-Simp

$$\frac{A \Rrightarrow A' \qquad \vdash \{A'\}\ c\ \{B'\} \qquad B' \Rrightarrow B}{\vdash \{A\}\ c\ \{B\}}$$

PR-Exp

$$\frac{[\![e]\!] \in \mathit{Values}}{\vdash \{\mathsf{True}\}\ e\ \{\lambda r.\, r = [\![e]\!]\}}$$

PR-Exists

$$\frac{\forall a \in A.\ \vdash \{a\}\ c\ \{B\}}{\vdash \{\bigvee A\}\ c\ \{B\}}$$

PR-Fork

$$\frac{\vdash \{\mathsf{phase}(\tau.\mathtt{Forkee}) * \mathsf{obs}(O_f) * A\}\ c\ \{\mathsf{obs}(\emptyset)\}}{\vdash \{\mathsf{phase}(\tau) * \mathsf{obs}(O_m \uplus O_f) * A\}\ \textbf{fork}\ c\ \{\lambda r.\, \mathsf{phase}(\tau.\mathtt{Forker}) * \mathsf{obs}(O_m) * r = \mathsf{tt}\}}$$

(a) Basic proof rules.

PR-If

$$\frac{\begin{array}{c}\vdash \{A\}\ c_b\ \{\lambda b.\, C(b) \land (b = \mathsf{True} \lor b = \mathsf{False})\} \\ \vdash \{C(\mathsf{True})\}\ c_t\ \{B\} \qquad C(\mathsf{False}) \Rrightarrow B\end{array}}{\vdash \{A\}\ \textbf{if}\ c_b\ \textbf{then}\ c_t\ \{B\}}$$

PR-While

$$\frac{\forall \tau_{\text{it}}.\ \tau \sqsubseteq \tau_{\text{it}} \ \Rightarrow\ \vdash \{\mathsf{phase}(\tau_{\text{it}}) * I(\tau_{\text{it}})\}\ c_b \left\{ \lambda b. \begin{array}{l} \exists \tau'_{\text{it}}, \tau_{\text{anc}}.\ \tau_{\text{anc}} \sqsubseteq \tau'_{\text{it}} * \mathsf{phase}(\tau'_{\text{it}}) \\ * (b = \mathsf{True} \lor b = \mathsf{False}) \\ * (b \rightarrow \mathsf{itperm}(\tau_{\text{anc}}, \delta) * I(\tau'_{\text{it}})) \\ * (\neg b \rightarrow B(\tau'_{\text{it}})) \end{array} \right\}}{\vdash \{\mathsf{phase}(\tau) * I(\tau)\}\ \textbf{while}\ c_b\ \textbf{do skip}\ \{\exists \tau'.\ \tau \sqsubseteq \tau' * \mathsf{phase}(\tau') * B(\tau')\}}$$

PR-Let

$$\frac{\vdash \{A\}\ c\ \{\lambda r.\, C(r)\} \qquad \forall v.\ \vdash \{C(v)\}\ c'[v/x]\ \{B\}}{\vdash \{A\}\ \textbf{let}\ x := c\ \textbf{in}\ c'\ \{B\}}$$

(b) Control structures.

Figure 8: Proof rules (part 1).

PR-Acquire

$$\frac{m.\mathsf{lev} \prec_{\mathsf{L}} O}{\begin{array}{l} \quad \{\mathsf{obs}(O) * [f]\mathsf{mutex}(m, P)\} \\ \vdash \quad \textbf{acquire}\ m.\mathsf{loc} \\ \quad \{\lambda r.\ r = \mathsf{tt} * \mathsf{obs}(O \uplus \{\!\!\{m\}\!\!\}) * \mathsf{locked}(m, P, f) * P\} \end{array}}$$

PR-Release

$$\frac{\mathsf{obs}(O) * A \Rrightarrow \mathsf{obs}(O) * P * B}{\begin{array}{l} \quad \{\mathsf{obs}(O \uplus \{\!\!\{m\}\!\!\}) * \mathsf{locked}(m, P, f) * A\} \\ \vdash \quad \textbf{release}\ m.\mathsf{loc} \\ \quad \{\lambda r.\ r = \mathsf{tt} * \mathsf{obs}(O) * [f]\mathsf{mutex}(m, P) * B\} \end{array}}$$

PR-NewMutex

$$\vdash \{\mathsf{True}\}\ \textbf{new\_mutex}\ \{\lambda \ell.\ \mathsf{uninit}(\ell)\}$$

(a) Mutexes.

PR-Cons

$$\vdash \{\mathsf{True}\}\ \textbf{cons}(v)\ \{\lambda \ell.\ \ell \mapsto v\}$$

PR-ReadHeapLoc

$$\vdash \{[f]\ell \mapsto v\}\ [\ell]\ \{\lambda r.\ r = v * [f]\ell \mapsto v\}$$

PR-AssignToHeap

$$\vdash \{\ell \mapsto \_\}\ [\ell] := v\ \{\lambda r.\ r = \mathsf{tt} * \ell \mapsto v\}$$

(b) Heap access.

Figure 9: Proof rules (part 2).

PR-While-Simp

$$\frac{\tau_{\text{anc}} \sqsubseteq \tau \qquad \vdash \{\mathsf{phase}(\tau) * A\}\ c_b\ \{\lambda b.\ \mathsf{phase}(\tau) * (b \ \rightarrow\ \mathsf{itperm}(\tau_{\text{anc}}, \delta) * A) * (\neg b \ \rightarrow\ B)\}}{\vdash \{\mathsf{phase}(\tau) * A\}\ \textbf{while}\ c_b\ \textbf{do skip}\ \{\mathsf{phase}(\tau) * B\}}$$

Figure 10: Derived proof rule.

9 Annotated Semantics

Definition 9.1 (Annotated Resources)**.** *We define the set of annotated resources AnnoRes as follows:*

$$\begin{array}{rcl} r^{\mathsf{a}} \in \mathit{AnnoRes} & ::= & \ell \mapsto v \mid \mathrm{uninit}_{\mathsf{aRes}}(\ell) \mid \\ & & \mathrm{unlocked}_{\mathsf{aRes}}((\ell, L), a, H) \mid \mathrm{locked}_{\mathsf{aRes}}((\ell, L), a, f) \mid \\ & & \mathrm{signal}_{\mathsf{aRes}}((id, L), b) \end{array}$$

Definition 9.2 (Annotated Heaps)**.** *We define the set of annotated heaps as*

$$\mathit{Heaps}^{\mathsf{annot}} \quad := \quad \mathcal{P}_{\mathsf{fin}}(\mathit{AnnoRes}),$$

the function $\mathsf{locs}_{\mathsf{ah}} : \mathit{Heaps}^{\mathsf{annot}} \to \mathcal{P}_{\mathsf{fin}}(\mathcal{L}ocs)$ *mapping annotated heaps to the sets of allocated heap locations as*

$$\begin{array}{rcl} \mathsf{locs}_{\mathsf{ah}}(h^{\mathsf{a}}) & := & \{\ell \in \mathcal{L}ocs \mid \exists v \in \mathit{Values}.\ \exists L \in \mathcal{L}evs.\ \exists a \in \mathcal{A}. \\ & & \quad \exists H \in \mathit{Heaps}^{\mathsf{log}}.\ \exists f \in \mathcal{F}. \\ & & \quad\quad \ell \mapsto v \in h^{\mathsf{a}} \ \vee\ \mathrm{uninit}_{\mathsf{aRes}}(\ell) \in h^{\mathsf{a}} \ \vee \\ & & \quad\quad \mathrm{unlocked}_{\mathsf{aRes}}((\ell, L), a, H) \in h^{\mathsf{a}} \ \vee \\ & & \quad\quad \mathrm{locked}_{\mathsf{aRes}}((\ell, L), a, f) \in h^{\mathsf{a}}\} \end{array}$$

and the function $\mathsf{ids}_{\mathsf{ah}} : \mathit{Heaps}^{\mathsf{annot}} \to \mathcal{P}_{\mathsf{fin}}(\mathcal{ID})$ *mapping annotated heaps to sets of allocated signal IDs as*

$$\mathsf{ids}_{\mathsf{ah}}(h^{\mathsf{a}}) \quad := \quad \{id \in \mathcal{ID} \mid \exists L \in \mathcal{L}evs.\ \exists b \in \mathbb{B}.\ \mathrm{signal}_{\mathsf{aRes}}((id, L), b) \in h^{\mathsf{a}}\}.$$

We denote annotated heaps by h^{a}*.*

We call an annotated heap h^{a} finite *and write* $\mathsf{finite}_{\mathsf{ah}}(h^{\mathsf{a}})$ *if there exists no chunk* $\mathrm{unlocked}_{\mathsf{aRes}}((\ell, L), a, H) \in h^{\mathsf{a}}$ *for which* $\mathsf{finite}_{\mathsf{lh}}(H)$ *does not hold.*

Definition 9.3 (Annotated Single Thread Reduction Relation)**.** *We define a reduction relation* $\rightsquigarrow_{\mathsf{ast}}$ *for annotated threads according to the rules presented in Figures 11 and 12. A reduction step has the form*

$$h^{\mathsf{a}}, H, c \rightsquigarrow_{\mathsf{ast}} h^{\mathsf{a}\prime}, H', c', T^{\mathsf{a}}$$

for a set of annotated forked threads $T^{\mathsf{a}} \subset \mathit{Heaps}^{\mathsf{log}} \times \mathit{Cmds}$ *with* $|T^{\mathsf{a}}| \leq 1$*.*

It indicates that given annotated heap h^{a} *and a logical heap* H*, command* c *can be reduced to annotated heap* $h^{\mathsf{a}\prime}$*, logical heap* H' *and command* c'*. The either empty or singleton set* T^{a} *represents whether a new thread is forked in this step.*

For simplicity of notation we omit T^{a} *if it is clear from the context that no thread is forked and* $T^{\mathsf{a}} = \emptyset$*.*

Definition 9.4 (Annotated Thread Pools)**.** *We define the set of annotated thread pools* $\mathcal{TP}^{\mathsf{a}}$ *as the set of finite partial functions mapping thread IDs to annotated threads:*

$$\mathcal{TP}^{\mathsf{a}} \quad := \quad \Theta \rightharpoonup_{\mathsf{fin}} \mathit{Heaps}^{\mathsf{log}} \times (\mathit{Cmds} \cup \{\mathsf{term}\}).$$

AST-Red-EvalCtxt

$$\frac{h^{\mathsf{a}}, H, c \rightsquigarrow_{\mathsf{ast}} h^{\mathsf{a}\prime}, H', c', T}{h^{\mathsf{a}}, H, E[c] \rightsquigarrow_{\mathsf{ast}} h^{\mathsf{a}\prime}, H', E[c'], T}$$

AST-Red-Fork

$$\begin{array}{l} h^{\mathsf{a}}, H_m + \{\mathrm{phase}_{\mathsf{lRes}}(\tau), \mathrm{obs}_{\mathsf{lRes}}(O_m \uplus O_f)\} + H_f, \textbf{fork } c \rightsquigarrow_{\mathsf{ast}} \\ h^{\mathsf{a}}, H_m + \{\mathrm{phase}_{\mathsf{lRes}}(\tau.\mathtt{Forker}), \mathrm{obs}_{\mathsf{lRes}}(O_m)\}, \mathsf{tt}, \{(\{\mathrm{phase}_{\mathsf{lRes}}(\tau.\mathtt{Forkee}), \mathrm{obs}_{\mathsf{lRes}}(O_f)\} + H_f),\ c)\} \end{array}$$

(a) Basic constructs.

AST-Red-While

$$h^{\mathsf{a}}, H, \textbf{while } c \textbf{ do skip} \rightsquigarrow_{\mathsf{ast}} h^{\mathsf{a}}, H, \textbf{if } c \textbf{ then } (\textbf{consumeItPerm}; \textbf{while } c \textbf{ do skip})$$

AST-Red-IfTrue

$$h^{\mathsf{a}}, H, \textbf{if } \mathsf{True} \textbf{ then } c \rightsquigarrow_{\mathsf{ast}} h^{\mathsf{a}}, H, c$$

AST-Red-IfFalse

$$h^{\mathsf{a}}, H, \textbf{if } \mathsf{False} \textbf{ then } c \rightsquigarrow_{\mathsf{ast}} h^{\mathsf{a}}, H, \mathsf{tt}$$

AST-Red-Let

$$h^{\mathsf{a}}, H, \textbf{let } x := v \textbf{ in } c \rightsquigarrow_{\mathsf{ast}} h^{\mathsf{a}}, H, c[v/x]$$

(b) Control structures.

AST-Red-ConsumeItPerm

$$\frac{H(\mathrm{phase}_{\mathsf{lRes}}(\tau)) \geq 1 \qquad \tau_{\mathrm{anc}} \sqsubseteq \tau}{h^{\mathsf{a}}, H + \{\mathrm{itperm}_{\mathsf{lRes}}(\tau_{\mathrm{anc}}, \delta)\}, \textbf{consumeItPerm} \rightsquigarrow_{\mathsf{ast}} h^{\mathsf{a}}, H, \mathsf{tt}}$$

(c) Intermediate representation.

AST-Red-Cons

$$\frac{\ell \notin \mathsf{locs}_{\mathsf{ah}}(h^{\mathsf{a}})}{h^{\mathsf{a}}, H, \textbf{cons}(v) \rightsquigarrow_{\mathsf{ast}} h^{\mathsf{a}} \cup \{\ell \mapsto v\}, H + \{\ell \mapsto v\}, \ell}$$

AST-Red-ReadHeapLoc

$$\frac{\ell \mapsto v \in h^{\mathsf{a}}}{h^{\mathsf{a}}, H, [\ell] \rightsquigarrow_{\mathsf{ast}} h^{\mathsf{a}}, H, v}$$

AST-Red-Assign

$$h \sqcup \{\ell \mapsto v\}, H + \{\ell \mapsto v\}, [\ell] := v \rightsquigarrow_{\mathsf{ast}} h \sqcup \{\ell \mapsto v'\}, H + \{\ell \mapsto v'\}, \mathsf{tt}$$

(d) Heap access.

Figure 11: Annotated single thread reduction rules (part 1).

AST-Red-NewMutex

$$\frac{\ell \notin \mathsf{locs}_{\mathsf{ah}}(h^{\mathsf{a}})}{h^{\mathsf{a}}, H, \textbf{new\_mutex} \rightsquigarrow_{\mathsf{ast}} h^{\mathsf{a}} \cup \{\mathrm{uninit}_{\mathsf{aRes}}(\ell)\}, H + \{\mathrm{uninit}_{\mathsf{lRes}}(\ell)\}, \ell}$$

AST-Red-Acquire

$$\frac{f \in \mathcal{F} \qquad m.\mathsf{lev} \prec_{\mathsf{L}} O}{\begin{array}{l} h^{\mathsf{a}} \sqcup \{\mathrm{unlocked}_{\mathsf{aRes}}(m, a, H_P)\},\ H + \{\mathrm{obs}_{\mathsf{lRes}}(O)\} + f \cdot \{\mathrm{mutex}_{\mathsf{lRes}}(m, P)\}, \\ \textbf{acquire}\ m.\mathsf{loc} \\ \rightsquigarrow_{\mathsf{ast}}\ h^{\mathsf{a}} \sqcup \{\mathrm{locked}_{\mathsf{aRes}}(m, P, f)\},\ H + \{\mathrm{obs}_{\mathsf{lRes}}(O \uplus \{\!\{m\}\!\}), \mathrm{locked}_{\mathsf{lRes}}(m, P, f)\} + H_P, \\ \qquad \mathsf{tt} \end{array}}$$

AST-Red-Release

$$\frac{\begin{array}{c} H_P \vDash_{\mathsf{A}} P \qquad \mathsf{consistent}_{\mathsf{lh}}(H_P) \\ \exists O.\ H(\mathrm{obs}_{\mathsf{lRes}}(O)) \geq 1 \qquad \exists \tau.\ H(\mathrm{phase}_{\mathsf{lRes}}(\tau)) \geq 1 \end{array}}{\begin{array}{l} h^{\mathsf{a}} \sqcup \{\mathrm{locked}_{\mathsf{aRes}}(m, P, f)\},\ H + \{\mathrm{obs}_{\mathsf{lRes}}(O \uplus \{\!\{m\}\!\}), \mathrm{locked}_{\mathsf{lRes}}(m, P, f)\} + H_P, \\ \textbf{release}\ m.\mathsf{loc} \\ \rightsquigarrow_{\mathsf{ast}}\ h^{\mathsf{a}} \sqcup \{\mathrm{unlocked}_{\mathsf{aRes}}(m, P, H_P)\},\ H + \{\mathrm{obs}_{\mathsf{lRes}}(O)\} + f \cdot \{\mathrm{mutex}_{\mathsf{lRes}}(m, P)\}, \\ \qquad \mathsf{tt} \end{array}}$$

(a) Mutexes.

Figure 12: Annotated single thread reduction rules (part 2).

We denote annotated thread pools by P^{a} and the empty thread pool by $\emptyset_{\mathsf{atp}}$, i.e.,

$$\begin{array}{rcl} \emptyset_{\mathsf{atp}} & : & \Theta \rightharpoonup_{\mathsf{fin}} \mathit{Heaps}^{\mathsf{log}} \times (\mathit{Cmds} \cup \{\mathsf{term}\}), \\ \mathsf{dom}(\emptyset_{\mathsf{atp}}) & = & \emptyset. \end{array}$$

We define the modification operations $+_{\mathsf{atp}}$ and $-_{\mathsf{atp}}$ analogously to $+_{\mathsf{tp}}$ and $-_{\mathsf{tp}}$, respectively, cf. Definition 6.4.

For convenience of notation we define selector functions for annotated threads as

$$\begin{array}{rcl} (H, c).\mathsf{heap} & := & H, \\ (H, c).\mathsf{cmd} & := & c. \end{array}$$

Definition 9.5 (Ghost Reduction Relation)**.** *We define a thread pool reduction relation $\rightsquigarrow_{\mathsf{ghost}}$ according to the rules presented in Figures 13 and 14 to express ghost steps. A ghost reduction step has the form*

$$h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{ghost}} h^{\mathsf{a}\prime}, P^{\mathsf{a}\prime}.$$

We denote its reflexive transitive closure by $\rightsquigarrow^*_{\mathsf{ghost}}$.

Definition 9.6 (Non-ghost Thread Pool Reduction Relation)**.** *We define a thread pool reduction relation $\rightsquigarrow_{\mathsf{real}}$ according to the rules presented in Figure 15*

GTP-Red-NewSignal

$$\frac{P^{\mathsf{a}}(\theta) = (H + \{\text{obs}_{\mathsf{lRes}}(O)\},\ c) \qquad id \notin \mathsf{ids}_{\mathsf{ah}}(h^{\mathsf{a}}) \quad H' = H + \{\text{signal}_{\mathsf{lRes}}((id, L), \mathsf{False}),\ \text{obs}_{\mathsf{lRes}}(O \uplus \{\!\{id, L\}\!\})\}}{h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{ghost}} h^{\mathsf{a}} \cup \{\text{signal}_{\mathsf{aRes}}((id, L), \mathsf{False})\},\ P^{\mathsf{a}}[\theta := (H', c)]}$$

GTP-Red-SetSignal

$$\frac{P^{\mathsf{a}}(\theta) = (H + \{\text{signal}_{\mathsf{lRes}}(s, \mathsf{False}),\ \text{obs}_{\mathsf{lRes}}(O \uplus \{\!\{s\}\!\})\},\ c) \quad H' = H + \{\text{signal}_{\mathsf{lRes}}(s, \mathsf{False}),\ \text{obs}_{\mathsf{lRes}}(O)\}}{h^{\mathsf{a}} \sqcup \{\text{signal}_{\mathsf{aRes}}(s, \mathsf{False})\}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{ghost}} h^{\mathsf{a}} \sqcup \{\text{signal}_{\mathsf{aRes}}(s, \mathsf{True})\},\ P^{\mathsf{a}}[\theta := (H', c)]}$$

GTP-Red-WaitPerm

$$\frac{\delta' <_{\mathsf{\Delta}} \delta \qquad P^{\mathsf{a}}(\theta) = (H + \{\text{itperm}_{\mathsf{lRes}}(\tau', \delta)\},\ c)}{h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{ghost}} h^{\mathsf{a}}, P^{\mathsf{a}}[\theta := (H + \{\text{wperm}_{\mathsf{lRes}}(\tau', id, \delta')\},\ c)]}$$

GTP-Red-Wait

$$\frac{\text{signal}_{\mathsf{aRes}}(s, \mathsf{False}) \in h^{\mathsf{a}} \qquad P^{\mathsf{a}}(\theta) = (H, c) \quad H(\text{phase}_{\mathsf{lRes}}(\tau)) \geq 1 \qquad H(\text{wperm}_{\mathsf{lRes}}(\tau_{\text{anc}}, s.\mathsf{id}, \delta)) \geq 1 \qquad H(\text{obs}_{\mathsf{lRes}}(O)) \geq 1 \quad \tau_{\text{anc}} \sqsubseteq \tau \qquad \forall o \in O.\, \mathsf{lev}(s) <_{\mathsf{L}} \mathsf{lev}(O)}{h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{ghost}} h^{\mathsf{a}}, P^{\mathsf{a}}[\theta := (H + \{\text{itperm}_{\mathsf{lRes}}(\tau, \delta)\},\ c)]}$$

GTP-Red-SpecItPerm

$$\frac{\tau_{\text{anc}} \sqsubseteq \tau \qquad P^{\mathsf{a}}(\theta) = (H + \{\mathsf{itperm}(\tau_{\text{anc}}, \delta)\},\ c)}{h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{ghost}} h^{\mathsf{a}}, P^{\mathsf{a}}[\theta := (H + \{\mathsf{itperm}(\tau, \delta)\},\ c)]}$$

GTP-Red-SpecWaitPerm

$$\frac{\tau_{\text{anc}} \sqsubseteq \tau \qquad P^{\mathsf{a}}(\theta) = (H + \{\mathsf{wperm}(\tau_{\text{anc}}, id, \delta)\},\ c)}{h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{ghost}} h^{\mathsf{a}}, P^{\mathsf{a}}[\theta := (H + \{\mathsf{wperm}(\tau, id, \delta)\},\ c)]}$$

GTP-Red-WeakItPerm

$$\frac{\delta' <_{\mathsf{\Delta}} \delta \qquad N \in \mathbb{N} \qquad P^{\mathsf{a}}(\theta) = (H + \{\text{itperm}_{\mathsf{lRes}}(\tau', \delta)\},\ c)}{h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{ghost}} h^{\mathsf{a}}, P^{\mathsf{a}}[\theta := (H + N \cdot \{\text{itperm}_{\mathsf{lRes}}(\tau', \delta')\},\ c)]}$$

GTP-Red-MutInit

$$\frac{P^{\mathsf{a}}(\theta) = (H + \{\text{uninit}_{\mathsf{lRes}}(\ell)\} + H_P,\ c) \qquad H_P \vDash_{\mathsf{A}} P \qquad \mathsf{consistent}_{\mathsf{lh}}(H_P) \quad \exists O.\ H(\text{obs}_{\mathsf{lRes}}(O)) \geq 1 \qquad \exists \tau.\ H(\text{phase}_{\mathsf{lRes}}(\tau)) \geq 1 \quad H' = H + \{\text{mutex}_{\mathsf{lRes}}((\ell, L), H_P)\}}{h^{\mathsf{a}} \sqcup \{\text{uninit}_{\mathsf{aRes}}(\ell)\}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{ghost}} h^{\mathsf{a}} \sqcup \{\text{unlocked}_{\mathsf{aRes}}((\ell, L), a, H_P)\},\ P^{\mathsf{a}}[\theta := (H', c)]}$$

Figure 13: Ghost thread pool reduction rules (part 1).

$$\textsc{GTP-Red-NewGhostCell}\quad \frac{\widehat{\ell} \notin \mathsf{getGLocs_{lh}}(H) \qquad P^{\mathsf{a}}(\theta) = (H, c)}{h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{ghost}} h^{\mathsf{a}}, P^{\mathsf{a}}[\theta := (H + \{\widehat{\ell} \mapsto \widehat{v}\},\, c)]}$$

$$\textsc{GTP-Red-MutateGhostCell}\quad \frac{\widehat{\ell} \notin \mathsf{getGLocs_{lh}}(H) \qquad P^{\mathsf{a}}(\theta) = (H + \{\widehat{\ell} \mapsto \widehat{v}\},\, c)}{h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{ghost}} h^{\mathsf{a}}, P^{\mathsf{a}}[\theta := (H + \{\widehat{\ell} \mapsto \widehat{v}'\},\, c)]}$$

Figure 14: Ghost thread pool reduction rules (part 2)

$$\textsc{RTP-Red-Lift}\quad \frac{\theta_f = \min(\Theta \setminus \mathsf{dom}(P^{\mathsf{a}})) \qquad P^{\mathsf{a}}(\theta) = (H, c) \qquad h^{\mathsf{a}}, H, c \rightsquigarrow_{\mathsf{ast}} h^{\mathsf{a}\prime}, H', c', T^{\mathsf{a}}}{h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{real}} h^{\mathsf{a}\prime}, P^{\mathsf{a}}[\theta := (H', c')] +_{\mathsf{atp}} T^{\mathsf{a}}}$$

$$\textsc{RTP-Red-Term}\quad \frac{P^{\mathsf{a}}(\theta) = (H, v) \qquad H.\mathsf{obs} = \emptyset}{h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{real}} h^{\mathsf{a}}, P^{\mathsf{a}} -_{\mathsf{atp}} \theta}$$

Figure 15: Non-ghost thread pool reduction rules.

to express real reduction steps. A reduction step has the form

$$h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{real}} h^{\mathsf{a}\prime}, P^{\mathsf{a}\prime}.$$

Definition 9.7 (Annotated Thread Pool Reduction Relation)**.** *We define the annotated thread pool reduction relation* $\rightsquigarrow_{\mathsf{atp}}$ *as*

$$\rightsquigarrow_{\mathsf{atp}} \quad := \quad \rightsquigarrow_{\mathsf{ghost}} \cup \rightsquigarrow_{\mathsf{real}}\ .$$

Definition 9.8 (Annotated Reduction Sequence)**.** *Let* $(h^{\mathsf{a}}{}_i)_{i\in\mathbb{N}}$ *and* $(P^{\mathsf{a}}{}_i)_{i\in\mathbb{N}}$ *be infinite sequences of annotated heaps and annotated thread pools, respectively. Let* $\mathsf{sig} : \mathbb{N} \rightharpoonup \mathcal{S}$ *be a partial function mapping indices to signals.*

We call $((h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}, \mathsf{sig})$ *an* annotated reduction sequence *if there exists a sequence of thread IDs* $(\theta_i)_{i\in\mathbb{N}}$ *such that the following holds for every* $i \in \mathbb{N}$*:*

- $h^{\mathsf{a}}_i, P^{\mathsf{a}}_i \overset{\theta_i}{\rightsquigarrow}_{\mathsf{atp}} h^{\mathsf{a}}_{i+1}, P^{\mathsf{a}}_{i+1}$
- *If this reduction step results from an application of* GTP-Red-Wait *to some signal* s*, then* $\mathsf{sig}(i) = s$ *holds and otherwise* $\mathsf{sig}(i) = \bot$*.*

In case the signal annotation sig *is clear from the context or not relevant, we omit it and write* $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}$ *instead of* $((h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}, \mathsf{sig})$*.*

We call $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)$ *an* annotated machine configuration*.*

Lemma 9.9 (Preservation of Finiteness). *Let* $(h_i^{\mathsf{a}}, P_i^{\mathsf{a}})_{i \in \mathbb{N}}$ *be an annotated reduction sequence with* $\mathsf{finite_{ah}}(h_0^{\mathsf{a}})$ *and* $\mathsf{finite_{lh}}(P_0^{\mathsf{a}}(\theta).\mathsf{heap})$ *for all* $\theta \in \mathsf{dom}(P_0^{\mathsf{a}})$.

Then, $\mathsf{finite_{lh}}(P_i^{\mathsf{a}}(\theta).\mathsf{heap})$ *holds for all* $i \in \mathbb{N}$ *and all* $\theta \in \mathsf{dom}(P_i^{\mathsf{a}})$.

Proof. Proof by induction on i. □

Lemma 9.10 (Preservation of Completeness). *Let* $(h_i^{\mathsf{a}}, P_i^{\mathsf{a}})_{i \in \mathbb{N}}$ *be an annotated reduction sequence with* $\mathsf{complete_{lh}}(P_0^{\mathsf{a}}(\theta).\mathsf{heap})$ *for all* $\theta \in \mathsf{dom}(P_0^{\mathsf{a}})$. *Furthermore, let there be no chunk* $\text{unlocked}_{\mathsf{aRes}}(m, P, H_P) \in h_0^{\mathsf{a}}$ *such that* $H_P(\text{phase}_{\mathsf{lRes}}(\tau)) > 0$ *or* $H_P(\text{obs}_{\mathsf{lRes}}(O)) > 0$ *holds for any* τ, O.

Then, $\mathsf{complete_{lh}}(P_i^{\mathsf{a}}(\theta).\mathsf{heap})$ *holds for every* $i \in \mathbb{N}$ *and every* $\theta \in \mathsf{dom}(P_i^{\mathsf{a}})$.

Proof. Proof by induction on i. □

Definition 9.11 (Fairness of Annotated Reduction Sequences). *We call an annotated reduction sequence* $(h_i^{\mathsf{a}}, P_i^{\mathsf{a}})_{i \in \mathbb{N}}$ fair *iff for all* $i \in \mathbb{N}$ *and* $\theta \in \mathsf{dom}(P_i^{\mathsf{a}})$ *with* $P_i^{\mathsf{a}}(\theta).\mathsf{cmd} \neq \mathsf{term}$ *there exists some* $k \geq i$ *with*

$$h_k^{\mathsf{a}}, P_k^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{real}} h_{k+1}^{\mathsf{a}}, P_{k+1}^{\mathsf{a}}.$$

Every thread of an annotated thread pool is annotated by a thread-local logical heap that expresses which resources are owned by this thread. In the following we define a function to extract the logical heap expressing which resources are owned by threads of a thread pool (i.e. the sum of all thread-local logical heaps).

Definition 9.12. *We define the function* $\mathsf{ownedResHeap_{atp}} : \mathcal{TP}^{\mathsf{a}} \to \mathit{Heaps}^{\mathsf{log}}$ *mapping annotated thread pools to logical heaps as follows:*

$$P^{\mathsf{a}} \quad \mapsto \sum_{\theta \,\in\, \mathsf{dom}(P^{\mathsf{a}})} P^{\mathsf{a}}(\theta).\mathsf{heap}$$

Annotated resources representing unlocked locks, i.e., $\text{unlocked}_{\mathsf{aRes}}(m, a, H_a)$, contain a logical heap H_a that expresses which resources are protected by this lock. In the following, we define a function that extracts a logical heap from an annotated heap h^{a} expressing which resources are protected by unlocked locks in h^{a}.

Definition 9.13. *We define the function* $\mathsf{protectedResHeap_{ah}} : \mathit{Heaps}^{\mathsf{annot}} \to \mathit{Heaps}^{\mathsf{log}}$ *mapping annotated heaps to logical heaps as follows:*

For any annotated heap h^{a} *let*

$$\begin{aligned} \mathit{LockInvs}(h^{\mathsf{a}}) \quad := \quad \{\!\!\{ H_P \in \mathit{Heaps}^{\mathsf{log}} \mid \exists m \in \mathcal{L}ocs \times \mathcal{L}evs.\ \exists P \in \mathcal{A}. \\ \text{unlocked}_{\mathsf{aRes}}(m, P, H_P) \in h^{\mathsf{a}} \}\!\!\} \end{aligned}$$

be the auxiliary set aggregating all logical heaps corresponding to lock invariants of unlocked locks stored in h^{a}. *We define* $\mathsf{protectedResHeap_{ah}}$ *as*

$$h^{\mathsf{a}} \quad \mapsto \sum_{H_P \,\in\, \mathit{LockInvs}(h^{\mathsf{a}})} H_P.$$

Definition 9.14 (Compatibility of Annotated and Logical Heaps)**.** *We inductively define a relation* $_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} \subset$ *Heaps*$^{\mathsf{annot}} \times$ *Heaps*$^{\mathsf{log}}$ *between annotated and logical heaps such that the following holds*

$$
\begin{array}{lll}
\emptyset & {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} & \emptyset_{\mathsf{log}}, \\
h^{\mathsf{a}} \cup \{\ell \mapsto v\} & {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} & H + \{\ell \mapsto v\}, \\
h^{\mathsf{a}} \cup \{\mathrm{uninit}_{\mathsf{aRes}}(\ell)\} & {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} & H + \{\mathrm{uninit}_{\mathsf{lRes}}(\ell)\}, \\
h^{\mathsf{a}} \cup \{\mathrm{unlocked}_{\mathsf{aRes}}(m, P, H_P)\} & {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} & H + \{\mathrm{mutex}_{\mathsf{lRes}}(m, P)\} + H_P, \\
h^{\mathsf{a}} \cup \{\mathrm{locked}_{\mathsf{aRes}}(m, P, f)\} & {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} & H + \{\mathrm{locked}_{\mathsf{lRes}}(m, P, f)\} \\
 & & \quad + (1 - f) \cdot \{\mathrm{mutex}_{\mathsf{lRes}}(m, P)\}, \\
h^{\mathsf{a}} \cup \{\mathrm{signal}_{\mathsf{aRes}}(s, b)\} & {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} & H + \{\mathrm{signal}_{\mathsf{lRes}}(s, b)\}, \\
h^{\mathsf{a}} & {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} & H + \{\mathrm{phase}_{\mathsf{lRes}}(\tau)\}, \\
h^{\mathsf{a}} & {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} & H + \{\mathrm{obs}_{\mathsf{lRes}}(O)\}, \\
h^{\mathsf{a}} & {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} & H + \{\mathrm{wperm}_{\mathsf{lRes}}(\tau, id, \delta)\}, \\
h^{\mathsf{a}} & {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} & H + \{\mathrm{itperm}_{\mathsf{lRes}}(\tau, \delta)\}, \\
h^{\mathsf{a}} & {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} & H + \{\widehat{\ell} \mapsto \widehat{v}\},
\end{array}
$$

where $h^{\mathsf{a}} \in$ *Heaps*$^{\mathsf{annot}}$ *and* $H \in$ *Heaps*$^{\mathsf{log}}$ *are annotated and logical heaps with* $\ell, m.\mathsf{loc} \notin \mathsf{locs}_{\mathsf{ah}}(h^{\mathsf{a}})$*,* $s.\mathsf{id} \notin \mathsf{ids}_{\mathsf{ah}}(h^{\mathsf{a}})$ *and* $h^{\mathsf{a}}\ {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}} H$*.*

We consider a machine configuration $(h^{\mathsf{a}}, P^{\mathsf{a}})$ to be *consistent* if it fulfils three criteria: (i) Every thread-local logical heap is consistent, i.e., for all used thread IDs θ, $P^{\mathsf{a}}(\theta).\mathsf{heap}$ only stores full phase, obligations, wait permission and iteration permission chunks. (ii) Every logical heap protected by an unlocked lock in h^{a} is consistent. (iii) h^{a} is compatible with the logical heap that represents (a) the resources owned by threads in P^{a} and (b) the resources protected by unlocked locks stored in h^{a}.

Definition 9.15 (Consistency of Annotated Machine Configurations)**.** *We call an annotated machine configuration* $(h^{\mathsf{a}}, P^{\mathsf{a}})$ consistent *and write* $\mathsf{consistent}_{\mathsf{conf}}(h^{\mathsf{a}}, P^{\mathsf{a}})$ *if all of the following hold:*

- $\mathsf{consistent}_{\mathsf{lh}}(P^{\mathsf{a}}(\theta).\mathsf{heap})$ *for all* $\theta \in \mathsf{dom}(P^{\mathsf{a}})$*,*
- $\forall m.\ \forall P.\ \forall H_P.\ \mathrm{unlocked}_{\mathsf{aRes}}(m, P, H_P) \in h^{\mathsf{a}} \ \rightarrow\ \mathsf{consistent}_{\mathsf{lh}}(H_P)$*,*
- $h^{\mathsf{a}}\ \ {}_{\mathsf{ah}}\!\!\sim_{\mathsf{lh}}\ \ \mathsf{ownedResHeap}_{\mathsf{atp}}(P^{\mathsf{a}}) + \mathsf{protectedResHeap}_{\mathsf{ah}}(h^{\mathsf{a}})$.

Lemma 9.16 (Preservation of Consistency)**.** *Let* $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i \in \mathbb{N}}$ *be an annotated reduction sequence with* $\mathsf{consistent}_{\mathsf{conf}}(h^{\mathsf{a}}_0, P^{\mathsf{a}}_0)$*. Then,* $\mathsf{consistent}_{\mathsf{conf}}(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)$ *holds for every* $i \in \mathbb{N}$*.*

Proof. Proof by induction on i. □

10 Hoare Triple Model Relation

Definition 10.1 (Command Annotation)**.** *We define the predicate* $\mathsf{annot}_{\mathsf{cmd}} \subset$ *Cmds* $\times$ *Cmds such that* $\mathsf{annot}_{\mathsf{cmd}}(c', c)$ *holds iff* c' *results from* c *by removing all occurrences of* **consumeItPerm***.*

Definition 10.2 (Thread Pool Annotation)**.** *We define a predicate* $\mathsf{annot_{tp}} \subset \mathcal{TP}^{\mathsf{a}} \times \mathcal{TP}$ *such that:*

$$\begin{gathered}\mathsf{annot_{tp}}(P^{\mathsf{a}}, P)\\ \Longleftrightarrow\\ \mathsf{dom}(P^{\mathsf{a}}) = \mathsf{dom}(P) \;\wedge\; \forall\theta \in \mathsf{dom}(P).\ \mathsf{annot_{cmd}}(P^{\mathsf{a}}(\theta).\mathsf{cmd}, P(\theta))\end{gathered}$$

Definition 10.3 (Compatibility of Annotated and Physical Heaps)**.** *We inductively define a relation* ${}_{\mathsf{ah}}{\sim}_{\mathsf{ph}} \subset \mathit{Heaps}^{\mathsf{annot}} \times \mathcal{R}^{\mathsf{phys}}$ *between annotated and physical heaps such that the following holds:*

$$\begin{array}{lcl}
\emptyset & {}_{\mathsf{ah}}{\sim}_{\mathsf{ph}} & \emptyset,\\
\ell \mapsto v \;\cup\; h^{\mathsf{a}} & {}_{\mathsf{ah}}{\sim}_{\mathsf{ph}} & \ell \mapsto v \;\cup\; h,\\
\mathrm{uninit}_{\mathsf{aRes}}(\ell) \;\cup\; h^{\mathsf{a}} & {}_{\mathsf{ah}}{\sim}_{\mathsf{ph}} & \mathrm{unlocked}_{\mathsf{pRes}}(\ell) \;\cup\; h,\\
\mathrm{unlocked}_{\mathsf{aRes}}((\ell, L), P, H_P) \;\cup\; h^{\mathsf{a}} & {}_{\mathsf{ah}}{\sim}_{\mathsf{ph}} & \mathrm{unlocked}_{\mathsf{pRes}}(\ell) \;\cup\; h,\\
\mathrm{locked}_{\mathsf{aRes}}((\ell, L), P, f) \;\cup\; h^{\mathsf{a}} & {}_{\mathsf{ah}}{\sim}_{\mathsf{ph}} & \mathrm{locked}_{\mathsf{pRes}}(\ell) \;\cup\; h,\\
\mathrm{signal}_{\mathsf{aRes}}(s, b) \;\cup\; h^{\mathsf{a}} & {}_{\mathsf{ah}}{\sim}_{\mathsf{ph}} & h
\end{array}$$

where $h^{\mathsf{a}} \in \mathit{Heaps}^{\mathsf{annot}}$ *and* $h \in \mathit{Heaps}^{\mathsf{phys}}$ *are annotated and physical heaps with* $h^{\mathsf{a}}\ {}_{\mathsf{ah}}{\sim}_{\mathsf{ph}}\ h$*.*

Definition 10.4 (Safety)**.** *We define the safety predicate* $\mathsf{safe} \subseteq \mathit{Heaps}^{\mathsf{log}} \times \mathit{Cmds}$ *coinductively as the greatest solution (with respect to* $\subseteq$*) of the following equation:*

$$\begin{array}{l}
\hspace{14em}\mathsf{safe}(H, c)\\
\hspace{14.5em}\Longleftrightarrow\\
\quad\mathsf{complete_{lh}}(H) \;\rightarrow\\
\forall P, P'.\,\forall\theta \in \mathsf{dom}(P).\,\forall h, h'.\,\forall P^{\mathsf{a}}.\,\forall h^{\mathsf{a}}.\\
\quad \mathsf{consistent_{conf}}(h^{\mathsf{a}}, P^{\mathsf{a}}) \;\wedge\; h^{\mathsf{a}}\ {}_{\mathsf{ah}}{\sim}_{\mathsf{ph}}\ h \;\wedge\\
\quad P(\theta) = c \;\wedge\; P^{\mathsf{a}}(\theta) = (H, c) \;\wedge\; \mathsf{annot_{tp}}(P^{\mathsf{a}}, P) \;\wedge\; h, P \overset{\theta}{\rightsquigarrow}_{\mathsf{tp}} h', P' \;\rightarrow\\
\quad \exists P^{\mathsf{G}}, P^{\mathsf{a}\prime}.\ \exists h^{\mathsf{G}}, h^{\mathsf{a}\prime}.\\
\qquad h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}{}^{*}_{\mathsf{ghost}}\ h^{\mathsf{G}}, P^{\mathsf{G}} \;\wedge\; h^{\mathsf{G}}, P^{\mathsf{G}} \overset{\theta}{\rightsquigarrow}_{\mathsf{real}} h^{\mathsf{a}\prime}, P^{\mathsf{a}\prime} \;\wedge\; \mathsf{annot_{tp}}(P^{\mathsf{a}\prime}, P') \;\wedge\\
\qquad h^{\mathsf{a}\prime}\ {}_{\mathsf{ah}}{\sim}_{\mathsf{ph}}\ h' \;\wedge\\
\qquad \forall(H_f, c_f) \in \mathsf{range}(P^{\mathsf{a}\prime}) \setminus \mathsf{range}(P^{\mathsf{a}}).\ \mathsf{safe}(H_f, c_f).
\end{array}$$

Definition 10.5 (Hoare Triple Model Relation)**.** *We define the model relation for Hoare triples* $\vDash_{\mathsf{H}} \subset \mathcal{A} \times \mathit{Cmds} \times (\mathit{Values} \rightarrow \mathcal{A})$ *such that:*

$$\begin{gathered}
\vDash_{\mathsf{H}} \{A\}\ c\ \{\lambda r.\, B(r)\}\\
\Longleftrightarrow\\
\forall H_F.\ \forall E.\ (\forall v.\ \forall H_B.\ H_B \vDash_{\mathsf{A}} B(v) \;\rightarrow\; \mathsf{safe}(H_B + H_F,\ E[v]))\\
\rightarrow\quad \forall H_A.\ H_A \vDash_{\mathsf{A}} A \;\rightarrow\; \mathsf{safe}(H_A + H_F,\ E[c])
\end{gathered}$$

We can instantiate context E in above definition to **let** $x := \square$ **in** $\mathtt{tt}$, which yields the consequent $\mathsf{safe}(H_A + H_F, \textbf{let}\ x := c\ \textbf{in}\ \mathtt{tt})$. Note that this implies $\mathsf{safe}(H_A + H_F, c)$.

Lemma 10.6 (Hoare Triple Soundness)**.** *Let* $\vdash \{A\}\ c\ \{B\}$ *hold, then also* $\vDash_{\mathsf{H}} \{A\}\ c\ \{B\}$ *holds.*

Proof. Proof by induction on the derivation of $\vdash \{A\}\ c\ \{B\}$. $\square$

Theorem 10.7 (Soundness)**.** *Let*

$$\vdash \{\mathsf{phase}(\tau) * \mathsf{obs}(\emptyset) * \mathop{\huge *}^{i=1,\ldots,N} \mathsf{itperm}(\tau, \delta_i)\}\ c\ \{\mathsf{obs}(\emptyset)\}$$

hold. There exists no fair, infinite reduction sequence $(h_i, P_i)_{i\in\mathbb{N}}$ *with* $h_0 = \emptyset$ *and* $P_0 = \{(\theta_0, c)\}$ *for any choice of* θ_0*.*

11 Soundness

In this section, we prove the soundness theorem 10.7.

Lemma 11.1 (Construction of Annotated Reduction Sequences)**.** *Suppose we can prove* $\vDash_{\mathsf{H}} \{A\}\ c\ \{\mathsf{obs}(\emptyset)\}$*. Let* H_A *be a logical heap with* $H_A \vDash_{\mathsf{A}} A$ *and* $\mathsf{complete}_{\mathsf{lh}}(H_A)$ *and* h^{a}_0 *an annotated heap with* $h^{\mathsf{a}}_0\ {}_{\mathsf{ah}}{\sim}_{\mathsf{lh}}\ H_A$*. Let* $(h_i, P_i)_{i\in\mathbb{N}}$ *be a fair plain reduction sequence with* $h^{\mathsf{a}}_0\ {}_{\mathsf{ah}}{\sim}_{\mathsf{ph}}\ h_0$ *and* $P_0 = \{(\theta_0, c)\}$ *for some thread ID* θ_0 *and command* c*.*

Then, there exists a fair annotated reduction sequence $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}$ *with* $P^{\mathsf{a}} = \{(\theta_0, (H_A, c))\}$ *and* $\mathsf{consistent}_{\mathsf{conf}}(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)$ *for all* $i \in \mathbb{N}$*.*

Proof. We can construct the annotated reduction sequence inductively from the plain reduction sequence. $\square$

Definition 11.2 (Program Order Graph)**.** *Let* $((h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}, \mathsf{sig})$ *be an annotated reduction sequence. Let* N^r *be the set of names referring to reduction rules defining the relations* $\rightsquigarrow_{\mathsf{real}}$, $\rightsquigarrow_{\mathsf{ghost}}$ *and* $\rightsquigarrow_{\mathsf{ast}}$*. We define the set of annotated reduction rule names* N^{a} *where* GTP-Red-Wait *is annotated by signals as*

$$\begin{aligned} N^{\mathsf{a}} \ :=\ & (N^r \setminus \{\text{GTP-Red-Wait}\}) \\ & \cup\ (\{\text{GTP-Red-Wait}\} \times \mathcal{S}). \end{aligned}$$

We define the program order graph $G(((h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}, \mathsf{sig})) = (\mathbb{N}, E)$ *with root 0 where* $E \subset \mathbb{N} \times \Theta \times N^{\mathsf{a}} \times \mathbb{N}$*.*

A node $a \in \mathbb{N}$ *corresponds to the sequence's* a^{th} *reduction step, i.e.,* $h^{\mathsf{a}}_a, P^{\mathsf{a}}_a \overset{\theta}{\rightsquigarrow}_{\mathsf{atp}} h^{\mathsf{a}}_{a+1}, P^{\mathsf{a}}_{a+1}$ *for some* $\theta \in \mathsf{dom}(P^{\mathsf{a}}_a)$*. An edge from node* a *to node* b *expresses that the* b^{th} *reduction step continues the control flow of step* a*. For any* $\ell \in \mathbb{N}$*, let* θ_ℓ *denote the ID of the thread reduced in step* ℓ*. Furthermore, let* n^{a}_ℓ *denote the name of the reduction rule applied in the* ℓ^{th} *step, in the following sense:*

- *If* $h^{\mathsf{a}}_\ell, P^{\mathsf{a}}_\ell \overset{\theta}{\rightsquigarrow}_{\mathsf{atp}} h^{\mathsf{a}}_{\ell+1}, P^{\mathsf{a}}_{\ell+1}$ *results from an application of* RTP-Red-Lift *in combination with single-thread reduction rule* n^{st}*, then* $n^{\mathsf{a}}_\ell = n^{\mathsf{st}}$*.*

- *If $h^{\mathsf{a}}_\ell, P^{\mathsf{a}}_\ell \overset{\theta}{\rightsquigarrow}_{\mathsf{atp}} h^{\mathsf{a}}_{\ell+1}, P^{\mathsf{a}}_{\ell+1}$ results from an application of* GTP-Red-Wait, *then $n^{\mathsf{a}}_\ell = (\text{GTP-Red-Wait}, \mathsf{sig}(\ell))$.*

- *Otherwise, n^{a} denotes the applied (real or ghost) thread pool reduction rule.*

An edge $(a, \theta, n^{\mathsf{a}}, b) \in \mathbb{N} \times \Theta \times N^{\mathsf{a}} \times \mathbb{N}$ is contained in E if $n^{\mathsf{a}} = n^{\mathsf{a}}_a$ and one of the following conditions applies:

- *$\theta = \theta_a = \theta_b$ and $b = \min(\{k > a \mid h^{\mathsf{a}}_k, P^{\mathsf{a}}_k \overset{\theta_a}{\rightsquigarrow}_{\mathsf{atp}} h^{\mathsf{a}}_{k+1}, P^{\mathsf{a}}_{k+1}\})$.*
  In this case, the edge expresses that step b marks the first time that thread θ_a is rescheduled for reduction (after step a).

- *$\mathsf{dom}(P^{\mathsf{a}}_{a+1}) \setminus \mathsf{dom}(P^{\mathsf{a}}_a) = \{\theta\}$ and*
  $b = \min\{k \in \mathbb{N} \mid h^{\mathsf{a}}_k, P^{\mathsf{a}}_k \overset{\theta}{\rightsquigarrow}_{\mathsf{atp}} h^{\mathsf{a}}_{k+1}, P^{\mathsf{a}}_{k+1}\}$.
  In this case, θ identifies the thread forked in step a. The edge expresses that step b marks the first reduction of the forked thread.

In case the choice of reduction sequence $((h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}, \mathsf{sig})$ is clear from the context, we write G instead of $G(((h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}, \mathsf{sig}))$.

Observation 11.3. *Let $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}$ be an annotated reduction sequence with $|\mathsf{dom}(P^{\mathsf{a}}_0)| = 1$. The sequence's program order graph $G((h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}})$ is a binary tree.*

For any reduction sequence $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}$, the paths in its program order graph $G((h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}})$ represent the sequence's control flow paths. Hence, we are going to use program order graphs to analyse reduction sequences' control flows.

We refer to a program order graph's edges by the kind of reduction step they represent. For instance, we call edges of the form $(a, \theta, \text{ST-Red-While}, b)$ *loop edges* because they represent a loop backjump and edges of the form $(a, \theta, (\text{GTP-Red-Wait}, s), b)$ *wait edges*. Wait edges of this form represent applications of GTP-Red-Wait to signal s.

In the following, we prove that any path in a program order graph that does not involve a loop edge is finite. This follows from the fact that the size of the command reduced along this path decreases with each non-ghost non-loop step.

Lemma 11.4. *Let $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}$ be a fair annotated reduction sequence. Let $p = (V, E)$ be a path in $G((h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}})$. Let $L = \{e \in E \mid \pi_3(e) = \text{AST-Red-While}\}$ be the set of loop edges contained in p. Then, p is infinite if and only if L is infinite.*

Proof. If L is infinite, p is obviously infinite as well. So, suppose L is finite.

For any command, we consider its size to be the number of nodes contained in its abstract syntax tree. By structural induction over the set of commands, it follows that the size of a command $c = P^{\mathsf{a}}(\theta).\mathsf{cmd}$ decreases in every non-ghost reduction step $h^{\mathsf{a}}, P^{\mathsf{a}} \overset{\theta}{\rightsquigarrow}_{\mathsf{atp}} h^{\mathsf{a}\prime}, P^{\mathsf{a}\prime}$ that is not an application of RTP-Red-Lift in combination with AST-Red-While.

Since L is finite, there exists a node x such that the suffix $p_{\geq x}$ starting at node x does not contain any loop edges. By fairness of $(h_i^{\mathsf{a}}, P_i^{\mathsf{a}})_{i\in\mathbb{N}}$, every non-empty suffix of $p_{\geq x}$ contains an edge corresponding to a non-ghost reduction step. For any edge $e = (i, \theta, n, j)$ consider the command $c_e = P_i^{\mathsf{a}}(\theta).\mathsf{cmd}$ reduced in this edge. The size of these commands decreases along $p_{\geq x}$. So, $p_{\geq x}$ must be finite and thus p must be finite as well. □

Corollary 11.5. *Let $(h_i^{\mathsf{a}}, P_i^{\mathsf{a}})_{i\in\mathbb{N}}$ be a fair annotated reduction sequence. Let $p = (V, E)$ be a path in $G((h_i^{\mathsf{a}}, P_i^{\mathsf{a}})_{i\in\mathbb{N}})$. Let*

$$C \quad = \quad \{e \in E \mid \pi_3(e) = \text{AST-Red-ConsumeItPerm}\}$$

be the set of consume edges contained in p. Then, p is infinite if and only if C is infinite.

Proof. Follows from Lemma 11.4 by the fact that the set $\{e \in E \mid \pi_3(e) = \text{AST-Red-While}\}$ is infinite if and only if C is infinite. □

Definition 11.6. *Let $G = (V, E)$ be a subgraph of some program order graph. We define the function $\mathsf{waitEdges}_G : \mathcal{S} \to \mathcal{P}(E)$ mapping any signal s to the set of wait edges in G concerning s as:*

$$\mathsf{waitEdges}_G(s) \quad := \quad \{(a, \theta, (\text{GTP-Red-Wait}, s'), b) \in E \mid s' = s\}.$$

Furthermore, we define the set $\mathcal{S}_G \subset \mathcal{S}$ of signals being waited for in G and its subset $S_G^\infty \subseteq \mathcal{S}_G$ of signals waited-for infinitely often in G as follows:

$$\begin{array}{rcl} \mathcal{S}_G & := & \{s \in \mathcal{S} \mid \mathsf{waitEdges}_G(s) \neq \emptyset\}, \\ S_G^\infty & := & \{s^\infty \in \mathcal{S}_G \mid \mathsf{waitEdges}_G(s^\infty) \text{ infinite}\}. \end{array}$$

Definition 11.7. *Let $(h_i^{\mathsf{a}}, P_i^{\mathsf{a}})_{i\in\mathbb{N}}$ be a fair annotated reduction sequence and let $G = (V, E)$ be a subgraph of the sequence's program order graph. We define the function $\mathsf{itperms}_G : E \to Bags_{\mathsf{fin}}(\Lambda)$ mapping any edge e to the (potentially empty) finite bag of iteration permissions derived in the reduction step corresponding to e as follows:*

Let $(i, \theta, n, j) \in E$ be an edge.

- *If $n = (\text{GTP-Red-Wait}, s)$ for some signal $s \in \mathcal{S}$, then the i^{th} reduction step spawns a single iteration permission (τ, δ), i.e.,* $P_{i+1}^{\mathsf{a}} = P_i^{\mathsf{a}}[\theta := (P_i^{\mathsf{a}}(\theta).\mathsf{heap} + \{\text{itperm}_{\mathsf{IRes}}(\tau, \delta)\},\ P_i^{\mathsf{a}}(\theta).\mathsf{cmd})]$. *In this case, we define*

$$\mathsf{itperms}_G((i, \theta, (\text{GTP-Red-Wait}, s), j)) \quad := \quad \{\!\!\{(\tau, \delta)\}\!\!\}.$$

- *If $n = \text{GTP-Red-WeakItPerm}$, then the i^{th} reduction step consumes an iteration permission (τ', δ) and produces N permissions (τ', δ') of lower degree, i.e., $P_i^{\mathsf{a}}(\theta).\mathsf{heap} = H + \{\mathsf{itperm}(\tau', \delta)\}$ for some heap H and $P_{i+1}^{\mathsf{a}} = P_i^{\mathsf{a}}[\theta := (H',\ P_i^{\mathsf{a}}(\theta).\mathsf{cmd})]$ for*

$$H' \ = \ H + N \cdot \{\text{itperm}_{\mathsf{IRes}}(\tau', \delta')\}.$$

In this case, we define

$$\mathsf{itperms}_G((i,\theta,\text{GTP-Red-WeakItPerm},j)) \quad := \quad \{\!\!\{\underbrace{(\tau',\delta'),\ldots,(\tau',\delta')}_{N \text{ times}}\}\!\!\}.$$

- *Otherwise, we define*

$$\mathsf{itperms}_G((i,\theta,n,j)) \quad := \quad \emptyset.$$

Definition 11.8 (Signal Capacity). *Let $(h_i^{\mathsf{a}}, P_i^{\mathsf{a}})_{i\in\mathbb{N}}$ be a fair annotated reduction sequence and $G = (V, E)$ be a subgraph of the sequence's program order graph. We define the function $\mathsf{sigCap}_G : (\mathcal{S} \setminus S_G^\infty) \times \mathbb{N} \to Bags_{\mathsf{fin}}(\Lambda)$ mapping signals and indices to bags of iteration permissions as follows:*

$$\mathsf{sigCap}_G(s,i) \quad := \quad \biguplus_{\substack{(a,\theta,n,b)\,\in\,\mathsf{waitEdges}_G(s)\\ a\geq i}} \mathsf{itperms}_G((a,\theta,n,b)).$$

We call $\mathsf{sigCap}_G(s,i)$ the capacity *of signal s at index i.*

Note that the signal capacity above is indeed finite. For every G and every signal $s \in \mathcal{S} \setminus S_G^\infty$ the set of wait edges $\mathsf{waitEdges}_G(s)$ is finite. Hence, the big union above is a finite union over finite iteration permission bags.

Definition 11.9 (Partial Order on Permissions). *We define the partial order on iteration permissions $<_\Lambda \subset \Lambda \times \Lambda$ induced by $<_{\mathbf{\Delta}}$ such that*

$$(\tau_1,\delta_1) <_\Lambda (\tau_2,\delta_2) \quad\Longleftrightarrow\quad \delta_1 <_{\mathbf{\Delta}} \delta_2.$$

Lemma 11.10. *The partial order $<_\Lambda$ is well-founded.*

Proof. Follows directly from well-foundedness of $<_\Lambda$. $\square$

Definition 11.11 (Partial Order on Finite Bags). *Let X be a set and let $<_X \subset X \times X$ a partial order on X. We define the partial order $\prec_X \subset Bags_{\mathsf{fin}}(X) \times Bags_{\mathsf{fin}}(X)$ on finite bags over X as the Dershowitz-Manna ordering [Dershowitz and Manna(1979)] induced by $<_X$:*

$$\begin{aligned} A \prec_X B \quad\Longleftrightarrow\quad \exists C, D \in Bags_{\mathsf{fin}}(X).\ & \emptyset \neq C \subseteq B \\ & \wedge\ A = (B \setminus C) \uplus D \\ & \wedge\ \forall d \in D.\ \exists c \in C.\ d <_X c. \end{aligned}$$

We define $\preceq_X \subset Bags_{\mathsf{fin}}(X) \times Bags_{\mathsf{fin}}(X)$ such that

$$A \preceq_X B \quad\Longleftrightarrow\quad A = B\ \vee\ A \prec_X B$$

holds.

Corollary 11.12. *The partial order $\prec_\Lambda \subset Bags_{\mathsf{fin}}(\Lambda) \times Bags_{\mathsf{fin}}(\Lambda)$ is well-founded.*

Proof. Follows from [Dershowitz and Manna(1979)] and Lemma 11.10. □

In the following, we view paths in a program order graph as single-branched subgraphs. This allows us to apply above definitions on graphs to paths. In particular, this allows us to refer to the capacity of a signal s on a path p by referring to sigCap_p.

For the following definition, remember that a bag $B \in \mathit{Bags}(X)$ is a function $B : X \to \mathbb{N}$ while a logical heap $H \in \mathit{Heaps}^{\mathsf{log}}$ is a function $H : \mathcal{R}^{\mathsf{log}} \to \mathbb{Q}_{\geq 0}$. Also remember the signatures $\mathsf{ownedResHeap}_{\mathsf{atp}} : \mathcal{TP}^{\mathsf{a}} \to \mathit{Heaps}^{\mathsf{log}}$ and $\mathsf{protectedResHeap}_{\mathsf{ah}} : \mathit{Heaps}^{\mathsf{annot}} \to \mathit{Heaps}^{\mathsf{log}}$.

Definition 11.13. *We define the functions* $\mathsf{itperms}_{\mathsf{conf}} : \mathit{Heaps}^{\mathsf{annot}} \times \mathcal{TP}^{\mathsf{a}} \to \mathit{Bags}(\Lambda)$ *and* $\mathsf{wperms}_{\mathsf{conf}} : \mathit{Heaps}^{\mathsf{annot}} \times \mathcal{TP}^{\mathsf{a}} \to \mathit{Bags}(\Omega)$ *mapping annotated machine configurations to bags of iteration and wait permissions, respectively, as follows:*

$$\begin{aligned}
&\mathsf{itperms}_{\mathsf{conf}}(h^{\mathsf{a}}, P^{\mathsf{a}})(\tau, \delta)\\
&\quad := \quad \big\lfloor \big(\mathsf{ownedResHeap}_{\mathsf{atp}}(P^{\mathsf{a}}) + \mathsf{protectedResHeap}_{\mathsf{ah}}(h^{\mathsf{a}})\big)(\mathrm{itperm}_{\mathsf{lRes}}(\tau, \delta))\big\rfloor,\\[1em]
&\mathsf{wperms}_{\mathsf{conf}}(h^{\mathsf{a}}, P^{\mathsf{a}})(\tau, id, \delta)\\
&\quad := \quad \big\lfloor \big(\mathsf{ownedResHeap}_{\mathsf{atp}}(P^{\mathsf{a}}) + \mathsf{protectedResHeap}_{\mathsf{ah}}(h^{\mathsf{a}})\big)(\mathrm{wperm}_{\mathsf{lRes}}(\tau, id, \delta))\big\rfloor.
\end{aligned}$$

Note that for consistent annotated machine configurations $(h^{\mathsf{a}}, P^{\mathsf{a}})$ the above flooring is without any affect.

Corollary 11.14. *Let* $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i \in \mathbb{N}}$ *be an annotated reduction sequence such that* $\mathsf{finite}_{\mathsf{ah}}(h^{\mathsf{a}}_0)$ *and* $\mathsf{finite}_{\mathsf{lh}}(P^{\mathsf{a}}_0(\theta).\mathsf{heap})$ *hold for every* $\theta \in \mathsf{dom}(P^{\mathsf{a}}_0)$.

Then, $\mathsf{itperms}_{\mathsf{conf}}(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)$ *and* $\mathsf{wperms}_{\mathsf{conf}}(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)$ *are finite for every choice of* $i \in \mathbb{N}$.

Proof. Follows by preservation of finiteness, Lemma 9.9. □

Lemma 11.15. *Let* $G((h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i \in \mathbb{N}})$ *be a program order graph and let* $p = (V, E)$ *be a path in* G *with* $S^{\infty}_p = \emptyset$. *For every* $\theta \in \mathsf{dom}(P^{\mathsf{a}}_0)$ *let* $P^{\mathsf{a}}_0(\theta).\mathsf{heap}$ *be finite and complete. Further, let* h^{a}_0 *be finite and contain no chunks* $\mathrm{unlocked}_{\mathsf{aRes}}(m, P, H_P)$ *where* H_P *contains any phase or obligations chunk.*

Then, p *is finite.*

Proof. Assume p is infinite. We prove a contradiction by assigning a finite capacity to every node along the path. Let θ_i be the ID of the thread reduced in step i. For every $\theta \in \mathsf{dom}(P^{\mathsf{a}}_r)$ the logical heap $P^{\mathsf{a}}_0(\theta).\mathsf{heap}$ is complete and h^{a}_0 contains no chunks $\mathrm{unlocked}_{\mathsf{aRes}}(m, P, H_P)$ where H_P contains any phase or obligations chunk. By preservation of completeness, Lemma 9.10, $P^{\mathsf{a}}_i(\theta_i).\mathsf{heap}$ is also complete and hence it contains exactly one phase chunk $\mathrm{phase}_{\mathsf{lRes}}(\tau_i)$. That is, for every step i, the phase ID τ_i of the thread reduced in step i is uniquely defined.

Consider the function $\mathsf{nodeCap} : V \to \mathit{Bags}_{\mathsf{fin}}(\Lambda)$ defined as

$$\begin{array}{rcl} \mathsf{nodeCap}(i) & := & \{\!\!\{(\tau_{\mathrm{anc}}, \delta) \in \mathsf{itperms}_{\mathsf{conf}}(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i) \;\mid\; \tau_{\mathrm{anc}} \sqsubseteq \tau_i\}\!\!\} \\ & \uplus & \biguplus_{\substack{id \,\in\, \mathsf{waitIDs}(\tau_i) \\ (\tau_{\mathrm{anc}}, id, \delta) \,\in\, \mathsf{wperms}_{\mathsf{conf}}(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i) \\ L \,\in\, \mathcal{L}evs}} \mathsf{sigCap}_p((id, L), i). \end{array}$$

where $\mathsf{waitIDs}(\tau_i) := \{id \mid \exists \tau_{\mathrm{anc}}.\, (\tau_{\mathrm{anc}}, id, \_) \in \mathsf{wperms}_{\mathsf{conf}}(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i) \,\wedge\, \tau_{\mathrm{anc}} \sqsubseteq \tau_i\}$.

For every $i \in V$, the capacity of node i, i.e., $\mathsf{nodeCap}(i)$, is the union of two finite iteration permission bags: (i) Above $\{\!\!\{(\tau_{\mathrm{anc}}, \delta) \in \mathsf{itperms}_{\mathsf{conf}}(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i) \mid \tau_{\mathrm{anc}} \sqsubseteq \tau_i\}\!\!\}$ captures all iteration permissions contained in h^{a} and P^{a}_i that are qualified by an ancestor τ_{anc} of phase ID τ_i and are hence usable by the thread reduced in node i. This includes the permissions $(\tau_{\mathrm{anc}}, \delta)$ held by thread θ_i as well as such (temporarily) transferred to another thread via a lock invariant. (ii) Below $\biguplus \mathsf{sigCap}_p((id, L), i)$ captures all iteration permissions that will be created along the suffix of p that starts at node i by waiting for signals for which thread θ_i already holds a wait permission $(\tau_{\mathrm{anc}}, id, \delta)$ in step i.

Note that for every $i \in V$, the bag of iteration permissions returned by $\mathsf{nodeCap}(i)$ is indeed finite. The initial annotated heap and all initial thread-local logical heaps are finite. This allows us to apply Corollary 11.14, by which we get that $\mathsf{itperms}_{\mathsf{conf}}(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)$ and $\mathsf{wperms}_{\mathsf{conf}}(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)$ are finite.

Since signal IDs are unique, for every fixed choice of i and id, there is at most one level L, for which $\mathsf{sigCap}_p((id, L), i) \neq \emptyset$. By assumption, along p all signals are waited for only finitely often, i.e., $S^{\infty}_p = \emptyset$. Hence, also the big union $\biguplus \mathsf{sigCap}_p((id, L), i)$ is defined and finite.

Consider the sequence $(\mathsf{nodeCap}(i))_{i \in V}$. Since every element is a finite bag of permissions, we can order it by $\prec_\Lambda$. We are going to prove a contradiction by proving that the sequence is an infinitely descending chain.

Consider any edge $(i, \theta, n, j) \in E$. There are only three cases in which $\mathsf{nodeCap}(i) \neq \mathsf{nodeCap}(j)$ holds.

- $n =$ GTP-Red-WaitPerm:
  In this case, there are degrees δ, δ' with $\delta' <_{\mathsf{\Delta}} \delta$, a signal s and $N \in \mathbb{N}$ for which we get

  $$\mathsf{nodeCap}(j) = (\mathsf{nodeCap}(i) \setminus \{\!\!\{(\tau', \delta)\}\!\!\}) \uplus \{\!\!\{\underbrace{(\tau', \delta')}_{N \text{ times}}\}\!\!\}.$$

  That is, $\mathsf{nodeCap}(j) \prec_\Lambda \mathsf{nodeCap}(i)$.

- $n =$ GTP-Red-WeakItPerm: Same as above.

- $n =$ AST-Red-ConsumeItPerm:
  In this case, we know that $\mathsf{nodeCap}(j) = \mathsf{nodeCap}(i) \setminus \{\!\!\{(\tau_{\mathrm{anc}}, \delta)\}\!\!\} \prec_\Lambda \mathsf{nodeCap}(i)$ holds for some τ_{anc} and δ.

(Note that in case of $n = \text{GTP-Red-Wait}$, we have $\mathsf{nodeCap}(i) = \mathsf{nodeCap}(j)$ since

$$\{\!\![(\tau_{\text{anc}}, \delta) \in \mathsf{itperms}_{\mathsf{conf}}(h^{\mathsf{a}}_j, P^{\mathsf{a}}_j) \mid \tau_{\text{anc}} \sqsubseteq \tau_j]\!\!\} \\ = \\ \{\!\![(\tau_{\text{anc}}, \delta) \in \mathsf{itperms}_{\mathsf{conf}}(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i) \mid \tau_{\text{anc}} \sqsubseteq \tau_i]\!\!\} \uplus \{\!\![(\tau, \delta)]\!\!\}$$

and

$$\biguplus \mathsf{sigCap}_p((id, L), j) \quad = \quad \Big(\biguplus \mathsf{sigCap}_p((id, L), i)\Big) \setminus \{\!\![(\tau, \delta)]\!\!\}$$

for some δ.) So, $\mathsf{nodeCap}$ is monotonically decreasing.

By assumption p is infinite. According to Corollary 11.5 this implies that the path contains infinitely many consume edges, i.e., edges with a labelling $n = \text{AST-Red-ConsumeItPerm}$. Hence, the sequence $(\mathsf{nodeCap}(i))_{i \in V}$ forms an infinitely descending chain. However, according to Corollary 11.12, $\prec_\Lambda$ is well-founded. A contradiction. $\square$

Lemma 11.16. *Let $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i \in \mathbb{N}}$ be a fair annotated reduction sequence with $\mathsf{consistent}_{\mathsf{conf}}(h^{\mathsf{a}}_0, P^{\mathsf{a}}_0)$, $P^{\mathsf{a}}_0 = \{(\theta_0, (H_0, c))\}$, $\mathsf{complete}_{\mathsf{lh}}(H_0)$, $\mathsf{finite}_{\mathsf{lh}}(H_0)$ and with $\mathsf{finite}_{\mathsf{ah}}(h^{\mathsf{a}}_0)$. Let H_0 contain no signal or wait permission chunks. Further, let h^{a}_0 contain no chunks $\text{unlocked}_{\mathsf{aRes}}(m, P, H_P)$ where H_P contains any obligations, phase or signal chunks. Let G be the program order graph of $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i \in \mathbb{N}}$. Then, $S^\infty_G = \emptyset$.*

Proof. Suppose $S^\infty_G \neq \emptyset$. Since $\mathcal{L}evs$ is well-founded, the same holds for the set $\{\mathsf{lev}(s) \mid s \in S^\infty\}$. Hence, there is some $s_{\min} \in S^\infty$ for which no $z \in S^\infty$ with $\mathsf{lev}(z) <_{\mathsf{L}} \mathsf{lev}(s_{\min})$ exists.

Since neither the initial logical heap H_0 nor any unlocked lock invariant stored in h^{a}_0 does contain any signals, $s_{\min}$ must be created during the reduction sequence. The reduction step creating signal $s_{\min}$ is an application of GTP-Red-NewSignal, which simultaneously creates an obligation to set $s_{\min}$. By preservation of completeness, Lemma 9.10, every thread-local logical heap $P^{\mathsf{a}}_i(\theta).\mathsf{heap}$ annotating some thread θ in some step i is complete. According to reduction rule GTP-Red-Wait, every wait edge $(a, \theta, (\text{GTP-Red-Wait}, s_{\min}), b)$ implies together with completeness that in step a (i) thread θ does not hold any obligation for $s_{\min}$ (i.e. $P^{\mathsf{a}}_a(\theta).\mathsf{heap}.\mathsf{obs} = O$ for some bag of obligations O with $s_{\min} \notin O$) and (ii) $s_{\min}$ has not been set, yet (i.e. $\text{signal}_{\mathsf{aRes}}(s_{\min}, \mathsf{False}) \in h^{\mathsf{a}}_a$). Hence, in step a another thread $\theta_{\text{ob}} \neq \theta$ must hold the obligation for $s_{\min}$ (i.e. $P^{\mathsf{a}}_a(\theta_{\text{ob}}).\mathsf{heap}.\mathsf{obs} = O$ for some bag of obligations O with $s_{\min} \in O$). Since there are infinitely many wait edges concerning $s_{\min}$ in G, the signal is never set.

By fairness, for every wait edge as above, there must be a non-ghost reduction step $h^{\mathsf{a}}_k, P^{\mathsf{a}}_k \overset{\theta_{\text{ob}}}{\rightsquigarrow}_{\mathsf{atp}} h^{\mathsf{a}}_{k+1}, P^{\mathsf{a}}_{k+1}$ of the thread θ_{ob} holding the obligation for $s_{\min}$ with $k \geq a$. Hence, there exists an infinite path p_{ob} in G where each edge $(e, \theta_{\text{ob}}, n, f) \in \mathsf{edges}(p_{\text{ob}})$ concerns some thread θ_{ob} holding the obligation for $s_{\min}$. (Note that this thread ID does not have to be constant along the path, since the obligation can be passed on during fork steps.)

The path p_{ob} does not contain wait edges $(e, \theta_{\mathrm{ob}}, (\text{GTP-RED-WAIT}, s^\infty), f)$ for any $s^\infty \in S^\infty$, since reduction rule GTP-RED-WAIT would (together with completeness of $P^{\mathsf{a}}_e(\theta_{\mathrm{ob}}).\mathsf{heap}$) require s^∞ to be of a lower level than all held obligations. This restriction implies $\mathsf{lev}(s^\infty) <_{\mathsf{L}} \mathsf{lev}(s_{\min})$ and would hence contradict the minimality of $s_{\min}$. That is, $S^\infty_{p_{\mathrm{ob}}} = \emptyset$.

By preservation of finiteness, Lemma 9.9, we get that every logical heap associated with the root of p_{ob} is finite. This allows us to apply Lemma 11.15, by which we get that p_{ob} is finite. A contradiction. $\square$

Lemma 11.17. *Let*

$$\models_{\mathsf{H}} \left\{\mathsf{phase}(\tau_0) * \mathsf{obs}(\emptyset) * \mathop{\huge *}^{i=1,\ldots,N} \mathsf{itperm}(\tau_0, \delta_i)\right\}\ c\ \left\{\mathsf{obs}(\emptyset)\right\}$$

hold. There exists no fair, infinite annotated reduction sequence $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}$ *with* $h^{\mathsf{a}}_0 = \emptyset$, $P^{\mathsf{a}}_0 = \{(\theta_0, (H_0, c))\}$ *and*

$$H_0 \;=\; \{\mathrm{phase}_{\mathsf{lRes}}(\tau_0), \mathrm{obs}_{\mathsf{lRes}}(\emptyset), \mathrm{itperm}_{\mathsf{lRes}}(\tau_0, \delta_1), \ldots, \mathrm{itperm}_{\mathsf{lRes}}(\tau_0, \delta_N)\}.$$

Proof. Suppose a reduction sequence as described above exists. We are going to prove a contradiction by considering its infinite program order graph G.

According to Observation 11.3, G is a binary tree with an infinite set of vertices. By the Weak König's Lemma [Simpson(1999)] G has an infinite branch, i.e. an infinite path p starting at root 0.

The initial logical heap H_0 is complete and finite and the initial annotated machine configuration $(h^{\mathsf{a}}_0, P^{\mathsf{a}}_0)$ is consistent. By Lemma 11.16 we know that $S^\infty_G = \emptyset$. Since $S^\infty_p \subseteq S^\infty_G$, we get $S^\infty_p = \emptyset$. This allows us to apply Lemma 11.15, by which we get that p is finite, which is a contradiction. $\square$

Theorem 10.7 (Soundness)**.** *Let*

$$\vdash \left\{\mathsf{phase}(\tau) * \mathsf{obs}(\emptyset) * \mathop{\huge *}^{i=1,\ldots,N} \mathsf{itperm}(\tau, \delta_i)\right\}\ c\ \left\{\mathsf{obs}(\emptyset)\right\}$$

hold. There exists no fair, infinite reduction sequence $(h_i, P_i)_{i\in\mathbb{N}}$ *with* $h_0 = \emptyset$ *and* $P_0 = \{(\theta_0, c)\}$ *for any choice of* θ_0.

Proof. Suppose a reduction sequence as described above exists. Since we can prove $\vdash \left\{\mathsf{phase}(\tau) * \mathsf{obs}(\emptyset) * \mathop{\huge *}^{i=1,\ldots,N} \mathsf{itperm}(\tau, \delta_i)\right\}\ c\ \left\{\mathsf{obs}(\emptyset)\right\}$, we can also conclude $\models_{\mathsf{H}} \left\{\mathsf{phase}(\tau) * \mathsf{obs}(\emptyset) * \mathop{\huge *}^{i=1,\ldots,N} \mathsf{itperm}(\tau, \delta_i)\right\}\ c\ \left\{\mathsf{obs}(\emptyset)\right\}$ by Hoare triple soundness, Lemma 10.6. Consider the logical heap

$$H_0 = \{\mathrm{phase}_{\mathsf{lRes}}(\tau), \mathrm{obs}_{\mathsf{lRes}}(\emptyset), \mathrm{itperm}_{\mathsf{lRes}}(\tau, \delta_1), \ldots, \mathrm{itperm}_{\mathsf{lRes}}(\tau, \delta_N)\}$$

and the annotated heap $h^{\mathsf{a}}_0 = \emptyset$. It holds $H_0 \models_{\mathsf{A}} \mathsf{phase}(\tau) * \mathsf{obs}(\emptyset) * \mathop{\huge *}^{i=1,\ldots,N} \mathsf{itperm}(\tau, \delta_i)$, $h^{\mathsf{a}}_0 \;{}_{\mathsf{ah}}{\sim}_{\mathsf{lh}}\; H_0$ and $h^{\mathsf{a}}_0 \;{}_{\mathsf{ah}}{\sim}_{\mathsf{ph}}\; h_0$. This allows us to apply Lemma 11.1, by which we can construct a corresponding fair annotated reduction sequence $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}$ that starts with $h^{\mathsf{a}}_0 = \emptyset$ and $P^{\mathsf{a}}_0 = \{(\theta_0, (H_0, c))\}$. By Lemma 11.17 $(h^{\mathsf{a}}_i, P^{\mathsf{a}}_i)_{i\in\mathbb{N}}$ does not exist. A contradiction. $\square$

12 Verification Example

12.1 Minimal Example

Figures 16 and 17 sketch the verification of the example program presented in Figure 2. For this verification we let the set of values *Values* include natural numbers and choose $\mathcal{L}evs = \Delta = \mathbb{N}$.

12.2 Bounded FIFO

For this section, we let the set of values *Values* include natural numbers and finite sequences, aka lists, of natural numbers. Further, the set of operations *Ops* includes the canonical operations on natural numbers and lists, i.e., (i) $<$, $\leq$, $-$ and (ii) list concatenation $l_1 \cdot l_2$, prepending an element $e :: l$, getting the head and tail of a list $\mathbf{head}(l)$ (defined for non-empty l), $\mathbf{tail}(l)$ and getting the size of a list $\mathbf{size}(l)$. We denote the empty list by **nil**. We use the abbreviation $a\ R_1\ b\ R_2\ c$ for $R_1, R_2 \in \{<, \leq\}$ to denote $a\ R_1\ b\ *\ b\ R_2\ c$. Furthermore, we choose $\mathcal{L}evs = \Delta = \mathbb{N}$. Figure 19 presents an example program involving a bounded FIFO.

To simplify its verification, we refine the process of creating a new ghost signal, i.e., we split it in two steps: allocating a new signal ID and initializing a signal. To implement this, we replace view shift rule VS-NewSignal by the rules VS-AllocSigID and VS-SigInit presented in Figure 18. This way we can fix the IDs of all the signals we need throughout the proof at its beginning. This refinement does not affect the soundness of our verification approach. Figures 20 – 30 sketch the program's verification using fine-grained signals.

References

[Dershowitz and Manna(1979)] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. In *ICALP*, 1979. doi:10.1007/3-540-09510-1_15.

[Simpson(1999)] S. Simpson. Subsystems of second order arithmetic. In *Perspectives in mathematical logic*, 1999. doi:10.1017/CBO9780511581007.

[Tarski(1955)] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955. doi:10.2307/2963937.

$\{\mathsf{phase}(()) * \mathsf{obs}(\emptyset) * \mathsf{itperm}((), 1)\}$	
let x :=	PR-Let & PR-VS-Simp & VS-SemImp
$\quad\{\mathsf{True} * \mathsf{phase}(()) * \mathsf{obs}(\emptyset) * \mathsf{itperm}((), 1)\}$	
$\quad$ **cons**(0)	PR-Cons & PR-Frame
$\quad\{\lambda \ell.\ \ell \mapsto 0 * \mathsf{phase}(()) * \mathsf{obs}(\emptyset) * \mathsf{itperm}((), 1)\}$	
in	
$\forall \ell_{\mathsf{x}}.$	ℓ_{x} represents value bound to x.
$\{\mathsf{phase}(()) * \mathsf{obs}(\emptyset) * \mathsf{itperm}((), 1) * \ell_{\mathsf{x}} \mapsto 0\}$	
let m :=	PR-Let & PR-VS-Simp & VS-SemImp
$\quad\{\mathsf{True} * \mathsf{phase}(()) * \mathsf{obs}(\emptyset) * \mathsf{itperm}((), 1) * \ell_{\mathsf{x}} \mapsto 0\}$	
$\quad$ **new_mutex**	PR-NewMutex & PR-Frame
$\quad\{\lambda \ell.\ \mathsf{uninit}(\ell) * \mathsf{phase}(()) * \mathsf{obs}(\emptyset) * \mathsf{itperm}((), 1) * \ell_{\mathsf{x}} \mapsto 0\}$	
in	
$\forall \ell_{\mathsf{m}}.$	ℓ_{m} represents value bound to m.
$\{\mathsf{phase}(()) * \mathsf{obs}(\emptyset) * \mathsf{itperm}((), 1) * \ell_{\mathsf{x}} \mapsto 0 * \mathsf{uninit}(\ell_{\mathsf{m}})\}$	PR-VS-Simp & VS-NewSignal & PR-Frame
$\left\{ \begin{array}{l} \exists \mathsf{id_s}.\ \mathsf{signal}((\mathsf{id_s}, 1), \mathsf{False}) * \mathsf{phase}(()) * \mathsf{obs}(\{\!\![(\mathsf{id_s}, 1)]\!\!\}) \\ * \mathsf{itperm}((), 1) * \ell_{\mathsf{x}} \mapsto 0 * \mathsf{uninit}(\ell_{\mathsf{m}}) \end{array} \right\}$	PR-Exists
$\forall \mathsf{id_s}.$	
$\left\{ \begin{array}{l} \exists \mathsf{id_s}.\ \mathsf{signal}((\mathsf{id_s}, 1), \mathsf{False}) * \mathsf{phase}(()) * \mathsf{obs}(\{\!\![(\mathsf{id_s}, 1)]\!\!\}) \\ * \mathsf{itperm}((), 1) * \ell_{\mathsf{x}} \mapsto 0 * \mathsf{uninit}(\ell_{\mathsf{m}}) \end{array} \right\}$	PR-VS-Simp & VS-SemImp
$\left\{ \begin{array}{l} \exists \mathsf{v_x}.\ \ell_{\mathsf{x}} \mapsto \mathsf{v_x} * \mathsf{signal}((\mathsf{id_s}, 1), \mathsf{v_x} \neq 0) \\ * \mathsf{obs}(\{\!\![(\mathsf{id_s}, 1)]\!\!\}) * \mathsf{itperm}((), 1) * \mathsf{uninit}(\ell_{\mathsf{m}}) \end{array} \right\}$	$\mathsf{P} := \exists \mathsf{v_x}.\ \ell_{\mathsf{x}} \mapsto \mathsf{v_x} * \mathsf{signal}((\mathsf{id_s}, 1), \mathsf{v_x} \neq 0)$
$\{\mathsf{P} * \mathsf{phase}(()) * \mathsf{obs}(\{\!\![(\mathsf{id_s}, 1)]\!\!\}) * \mathsf{itperm}((), 1) * \mathsf{uninit}(\ell_{\mathsf{m}})\}$	PR-VS-Simp & VS-MutInit & PR-Frame
$\{\mathsf{phase}(()) * \mathsf{obs}(\{\!\![(\mathsf{id_s}, 1)]\!\!\}) * \mathsf{itperm}((), 1) * \mathsf{mutex}((\ell_{\mathsf{m}}, 0), \mathsf{P})\}$	PR-VS-Simp & VS-SemImp
$\left\{ \begin{array}{l} \mathsf{phase}(()) * \mathsf{obs}(\{\!\![(\mathsf{id_s}, 1)]\!\!\}) * \mathsf{itperm}((), 1) \\ * [\frac{1}{2}]\mathsf{mutex}((\ell_{\mathsf{m}}, 0), \mathsf{P}) * [\frac{1}{2}]\mathsf{mutex}((\ell_{\mathsf{m}}, 0), \mathsf{P}) \end{array} \right\}$	
fork	PR-Fork & PR-Frame
$\quad\{\mathsf{phase}(\tau.\mathtt{Forkee}) * \mathsf{obs}(\emptyset) * \mathsf{itperm}((), 1) * [\frac{1}{2}]\mathsf{mutex}((\ell_{\mathsf{m}}, 0), \mathsf{P})\}$	
$\quad \ldots;$	Continued in Figure 17.
$\quad\{\mathsf{obs}(\emptyset)\}$	
$\{\mathsf{phase}(\tau.\mathtt{Forker}) * \mathsf{obs}(\{\!\![(\mathsf{id_s}, 1)]\!\!\}) * [\frac{1}{2}]\mathsf{mutex}((\ell_{\mathsf{m}}, 0), \mathsf{P})\}$	
acquire m;	PR-Acquire
$\{\mathsf{phase}(\tau.\mathtt{Forker}) * \mathsf{obs}(\{\!\![(\mathsf{id_s}, 1), (\ell_{\mathsf{m}}, 0)]\!\!\}) * \mathsf{locked}((\ell_{\mathsf{m}}, 0), \mathsf{P}, \frac{1}{2}) * \mathsf{P}\}$	PR-Exists
$\forall \mathsf{v_x}.$	
$\left\{ \begin{array}{l} \mathsf{phase}(\tau.\mathtt{Forker}) * \mathsf{obs}(\{\!\![(\mathsf{id_s}, 1), (\ell_{\mathsf{m}}, 0)]\!\!\}) * \mathsf{locked}((\ell_{\mathsf{m}}, 0), \mathsf{P}, \frac{1}{2}) \\ * \exists \mathsf{v_x}.\ \ell_{\mathsf{x}} \mapsto \mathsf{v_x} * \mathsf{signal}((\mathsf{id_s}, 1), \mathsf{v_x} \neq 0) \end{array} \right\}$	
$[\mathsf{x}] := 1;$	PR-AssignToHeap & PR-Frame
$\left\{ \begin{array}{l} \mathsf{phase}(\tau.\mathtt{Forker}) * \mathsf{obs}(\{\!\![(\mathsf{id_s}, 1), (\ell_{\mathsf{m}}, 0)]\!\!\}) * \mathsf{locked}((\ell_{\mathsf{m}}, 0), \mathsf{P}, \frac{1}{2}) \\ * \ell_{\mathsf{x}} \mapsto 1 * \mathsf{signal}((\mathsf{id_s}, 1), \mathsf{v_x} \neq 0) \end{array} \right\}$	PR-VS-Simp & VS-SetSignal
$\left\{ \begin{array}{l} \mathsf{phase}(\tau.\mathtt{Forker}) * \mathsf{obs}(\{\!\![(\mathsf{id_s}, 1), (\ell_{\mathsf{m}}, 0)]\!\!\}) * \mathsf{locked}((\ell_{\mathsf{m}}, 0), \mathsf{P}, \frac{1}{2}) \\ * \ell_{\mathsf{x}} \mapsto 1 * \mathsf{signal}((\mathsf{id_s}, 1), \mathsf{True}) \end{array} \right\}$	PR-VS-Simp & VS-SemImp
$\{\mathsf{phase}(\tau.\mathtt{Forker}) * \mathsf{obs}(\{\!\![(\ell_{\mathsf{m}}, 0)]\!\!\}) * \mathsf{locked}((\ell_{\mathsf{m}}, 0), \mathsf{P}, \frac{1}{2}) * \mathsf{P}\}$	
release m	PR-Release & PR-Frame
$\{\mathsf{phase}(\tau.\mathtt{Forker}) * \mathsf{obs}((\ell_{\mathsf{m}}, 0)\ \emptyset) * [\frac{1}{2}]\mathsf{mutex}((\ell_{\mathsf{m}}, 0), \mathsf{P}) * \mathsf{P}\}$	PR-VS-Simp & VS-SemImp
$\{\mathsf{obs}(\emptyset)\}$	

Figure 16: Verification sketch of main thread of example program presented in Figure 2. For readability we omit information about a command's return value if it is not relevant to the proof.

$\ldots$
$\forall \ell_x, id_s.$

$\ldots$

$\{phase(\tau.Forkee) * obs(\emptyset) * itperm((), 1) * [\frac{1}{2}]mutex((\ell_m, 0), P)\}$

$\{phase(\tau.Forkee) * obs(\emptyset) * itperm((), 1)\ wperm((), id_s, 0) * [\frac{1}{2}]mutex((\ell_m, 0), P)\}$

while

$\quad \{phase(\tau.Forkee) * obs(\emptyset) * wperm((), id_s, 0) * [\frac{1}{2}]mutex((\ell_m, 0), P)\}$

$\quad$ **acquire** m;

$\quad \{phase(\tau.Forkee) * obs(\{\!\![(\ell_m, 0)]\!\!\}) * wperm((), id_s, 0) * locked((\ell_m, 0), P, \frac{1}{2}) * P\}$

$\quad$ **let** y :=

$\quad\quad \left\{\begin{array}{l} phase(\tau.Forkee) * obs(\{\!\![(\ell_m, 0)]\!\!\}) * wperm((), id_s, 0) * locked((\ell_m, 0), P, \frac{1}{2}) \\ * \exists v_x.\ \ell_x \mapsto v_x * signal((id_s, 1), v_x \neq 0) \end{array}\right\}$

$\quad\quad\quad \forall v_x.$

$\quad\quad\quad \{\ell_x \mapsto v_x\}$

$\quad\quad\quad$ [x]

$\quad\quad\quad \{\lambda r.\ r = v_x * \ell_x \mapsto v_x\}$

$\quad\quad\quad \{\lambda r.\ \exists v_x.\ r = v_x * \ell_x \mapsto v_x\}$

$\quad\quad \left\{\begin{array}{l} \lambda r.phase(\tau.Forkee) * obs(\{\!\![(\ell_m, 0)]\!\!\}) * wperm((), id_s, 0) * locked((\ell_m, 0), P, \frac{1}{2}) \\ \quad * \exists v_x.\ \ell_x \mapsto v_x * signal((id_s, 1), v_x \neq 0) * r = v_x \end{array}\right\}$

$\quad$ **in**

$\quad \forall v_y.$

$\quad \left\{\begin{array}{l} phase(\tau.Forkee) * obs(\{\!\![(\ell_m, 0)]\!\!\}) * wperm((), id_s, 0) * locked((\ell_m, 0), P, \frac{1}{2}) \\ * \exists v_x.\ \ell_x \mapsto v_x * signal((id_s, 1), v_x \neq 0) * v_x = v_y \end{array}\right\}$

$\quad$ **release** m;

$\quad\quad \left\{\begin{array}{l} phase(\tau.Forkee) * obs(\{\!\![\ (\ell_m, 0)\ \ \emptyset\ ]\!\!\}) * wperm((), id_s, 0) \\ * locked((\ell_m, 0), P, \frac{1}{2}) * \exists v_x.\ \ell_x \mapsto v_x * signal((id_s, 1), v_x \neq 0) * v_x = v_y \end{array}\right\}$

$\quad\quad \forall v_x.$

$\quad\quad \left\{\begin{array}{l} phase(\tau.Forkee) * obs(\{\!\![\emptyset]\!\!\}) * wperm((), id_s, 0) \\ * \exists v_x.\ \ell_x \mapsto v_x * signal((id_s, 1), v_x \neq 0) * v_x = v_y \end{array}\right\}$

$\quad\quad \left\{\begin{array}{l} phase(\tau.Forkee) * obs(\{\!\![\emptyset]\!\!\}) * wperm((), id_s, 0) \\ * \ell_x \mapsto v_x * signal((id_s, 1), v_x \neq 0) * v_x = v_y * (v_x = 0 \leftrightarrow itperm(\tau.Forker, 0)) \end{array}\right\}$

$\quad\quad \left\{\begin{array}{l} phase(\tau.Forkee) * obs(\{\!\![\emptyset]\!\!\}) * wperm((), id_s, 0) * P \\ * (v_y = 0 \rightarrow itperm(\tau.Forker, 0)) \end{array}\right\}$

$\quad \left\{\begin{array}{l} phase(\tau.Forkee) * obs(\{\!\![\ (\ell_m, 0)\ \ \emptyset\ ]\!\!\}) * wperm((), id_s, 0) * [\frac{1}{2}]mutex((\ell_m, 0), P) \\ * P * (v_y = 0 \rightarrow itperm(\tau.Forker, 0)) \end{array}\right\}$

$\quad$ y = 0

$\quad \left\{\begin{array}{l} \lambda b.phase(\tau.Forkee) * obs(\emptyset) * wperm((), id_s, 0) * [\frac{1}{2}]mutex((\ell_m, 0), P) \\ \quad * (v_y = 0 \rightarrow itperm(\tau.Forker, 0)) * b = [\![v_y = 0]\!] \end{array}\right\}$

$\quad \left\{\begin{array}{l} \lambda b.phase(\tau.Forkee) * (\neg b \rightarrow obs(\emptyset)) \\ \quad * (b \rightarrow obs(\emptyset) * wperm((), id_s, 0) * [\frac{1}{2}]mutex((\ell_m, 0), P) * itperm(\tau.Forker, 0)) \end{array}\right\}$

do skip

$\{obs(\emptyset)\}$

Continuation of Figure 16.
ℓ_x, id_s universally quantified below
$P := \exists v_x.\ \ell_x \mapsto v_x$
$\quad * signal((id_s, 1), v_x \neq 0)$
PR-VS-SIMP & VS-WAITPERM
& PR-FRAME

PR-WHILE-SIMP

PR-ACQUIRE

PR-LET
PR-EXISTS & PR-FRAME
P
v_x quantified in local scope.

PR-VS-SIMP & VS-SEMIMP

P

v_y represents value bound to y.

PR-RELEASE

Release view shift
PR-EXISTS

v_x quantified in local scope.
PR-VS-SIMP & VS-WAIT
& PR-FRAME

PR-VS-SIMP & VS-SEMIMP

PR-EXP & PR-FRAME

PR-VS-SIMP & VS-SEMIMP

Figure 17: Verification sketch of busy-waiting thread of example program presented in Figure 2. For readability we omit information about a command's return value if it is not relevant to the proof.

VS-AllocSigID
$\mathsf{True} \Rrightarrow \exists id.\ \mathsf{uninitSig}(id)$

VS-SigInit
$\mathsf{obs}(O) * \mathsf{uninitSig}(id)$
$\Rrightarrow \mathsf{obs}(O \uplus \{\!\{(id, L)\}\!\}) * \mathsf{signal}((id, L), \mathsf{False})$

Figure 18: Fine-grained view shift rules for signal creation.

```
let fifo_10 := cons(nil) in
let m := new_mutex in
let c_p := cons(100) in
let c_c := cons(100) in
fork (
    while (
        acquire m;
        let f := [fifo_10] in
        if size(f) < 10 then (
            let c := [c_p] in
            [fifo_10] := f · (c :: nil);
            [c_p] := c − 1
        );
        release m;
        let c := [c_p] in
        c ≠ 0
    ) do skip;
);
while (
    acquire m;
    let f := [fifo_10] in
    if size(f) > 0 then (
        let c := [c_c] in
        [fifo_10] := tail(f);
        [c_c] := c − 1
    );
    release m;
    let c := [c_c] in
    c ≠ 0
) do skip
```

Figure 19: Example program with two threads communicating via a shared bounded FIFO with maximal size 10. Producer thread writes numbers 100, ..., 1 to shared FIFO and busy-waits until FIFO is not full and next element can be pushed. Consumer thread pops 100 numbers from FIFO and busy-waits for next number to arrive.

$\{\mathsf{phase}(()) * \mathsf{obs}(\emptyset) * \mathsf{itperm}((), 2)\}$

let $\mathsf{fifo}_{10} := \mathbf{cons}(\mathbf{nil})$ **in let** $\mathsf{m} := \mathbf{new\_mutex}$ **in**

$\forall \ell_{\mathsf{fifo}_{10}}, \ell_{\mathsf{m}}.$

PR-Let (2x) & PR-Cons & PR-NewMutex

$\{\mathsf{phase}(()) * \mathsf{obs}(\emptyset) * \mathsf{itperm}((), 2) * \ell_{\mathsf{fifo}_{10}} \mapsto \mathbf{nil} * \mathsf{uninit}(\ell_{\mathsf{m}})\}$

PR-VS-Simp & VS-AllocSigID & PR-Exists (200x)

$\forall id^1_{\text{pop}}, \ldots id^{100}_{\text{pop}}, id^1_{\text{push}}, \ldots, id^{100}_{\text{push}}.$

$\{ \bigstar_{i=1,\ldots,100} \mathsf{uninitSig}(id^i_{\text{pop}}) * \bigstar_{i=1,\ldots,100} \mathsf{uninitSig}(id^i_{\text{push}}) * \ldots\}$

$L^i_{\text{pop}} \;:=\; 102 - i, \quad L^i_{\text{push}} := 101 - i \quad \text{for} \quad 1 \leq i \leq 100$

(Later $L^{i+10}_{\text{pop}} < L^i_{\text{push}}$ and $L^i_{\text{push}} < L^i_{\text{pop}}$ must hold, cf. Figures 26 and 28.)

$s^i_{\text{push}} \;:=\; (id^i_{\text{push}}, L^i_{\text{push}}), \quad s^i_{\text{pop}} \;:=\; (id^i_{\text{pop}}, L^i_{\text{pop}}) \quad \text{for} \quad 1 \leq i \leq 100$

PR-VS-Simp & VS-SigInit

$\left\{ \begin{array}{l} \mathsf{uninitSig}(id^{100}_{\text{pop}}) * \mathsf{uninitSig}(id^{100}_{\text{push}}) \;\; \mathsf{signal}(s^{100}_{\text{pop}}, \mathsf{False}) * \mathsf{signal}(s^{100}_{\text{push}}, \mathsf{False}) \\ * \,\mathsf{obs}(\{\!\{ s^{100}_{\text{pop}}, s^{100}_{\text{push}} \}\!\}) * \ldots \end{array} \right\}$

PR-VS-Simp & VS-WeakPerm

$\{\mathsf{itperm}((), 2) \;\; \bigstar_{1,\ldots,400} \mathsf{itperm}((), 1) * \ldots\}$

PR-VS-Simp & VS-Wait

$\{\bigstar_{1,\ldots,200} \mathsf{itperm}((), 1) * \bigstar_{1,\ldots,200} \mathsf{itperm}((), 1) \;\; \bigstar_{i=1,\ldots,100} (\mathsf{wperm}((), id^i_{\text{pop}}, 0) * \mathsf{wperm}((), id^i_{\text{push}}, 0)) * \ldots\}$

Later each thread uses $\bigstar_{1,\ldots,100} \mathsf{itperm}((), 1)$ to justify productive iterations.

let $\mathsf{c_p} := \mathbf{cons}(100)$ **in let** $\mathsf{c_c} := \mathbf{cons}(100)$ **in**

$\forall \ell_{\mathsf{c_p}}, \ell_{\mathsf{c_c}}.$

PR-Let & PR-Cons (2x)

$\{\ell_{\mathsf{c_p}} \mapsto 100 * \ell_{\mathsf{c_c}} \mapsto 100 * \ldots\}$

PR-VS-Simp & VS-SemImp

$\left\{ \begin{array}{l} \mathsf{phase}(()) * \mathsf{obs}(\{\!\{ s^{100}_{\text{push}}, s^{100}_{\text{pop}} \}\!\}) * [\tfrac{1}{2}]\ell_{\mathsf{c_p}} \mapsto 100 * [\tfrac{1}{2}]\ell_{\mathsf{c_c}} \mapsto 100 * \mathsf{uninit}(\ell_{\mathsf{m}}) \\ * \bigstar_{i=1,\ldots,100} (\mathsf{wperm}((), id^i_{\text{pop}}, 0) * \mathsf{wperm}((), id^i_{\text{push}}, 0)) * \bigstar_{1,\ldots,200} \mathsf{itperm}((), 1) * P_{\mathsf{m}} \\ * \bigstar_{i=1,\ldots,99} (\mathsf{uninitSig}(id^i_{\text{push}}) * \mathsf{uninitSig}(id^i_{\text{pop}})) \end{array} \right\}$

For definition of lock invariant P_{m} cf. Figure 21.

PR-VS-Simp & VS-MutInit

$mut \;:=\; (\ell_{\mathsf{m}}, 0)$ (Later $\mathsf{lev}(mut) < L^i_{\text{push}}$ and $\mathsf{lev}(mut) < L^i_{\text{pop}}$ must hold for all $1 \leq i \leq 100$, cf. Figures 24 and 28.)

$\{\mathsf{uninit}(\ell_{\mathsf{m}}) * P_{\mathsf{m}} \;\; \mathsf{mutex}(mut, P_{\mathsf{m}}) * \ldots\}$

PR-VS-Simp & VS-SemImp

$\left\{ \begin{array}{l} \mathsf{phase}(()) * \mathsf{obs}(\{\!\{ s^{100}_{\text{push}}, s^{100}_{\text{pop}} \}\!\}) * [\tfrac{1}{2}]\ell_{\mathsf{c_p}} \mapsto 100 * [\tfrac{1}{2}]\ell_{\mathsf{c_c}} \mapsto 100 * \bigstar_{1,\ldots,200} \mathsf{itperm}((), 1) \\ * \bigstar_{i=1,\ldots,100} (\mathsf{wperm}((), id^i_{\text{pop}}, 0) * \mathsf{wperm}((), id^i_{\text{push}}, 0)) * \bigstar_{i=1,\ldots,99} (\mathsf{uninitSig}(id^i_{\text{push}}) * \mathsf{uninitSig}(id^i_{\text{pop}})) \\ * [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}}) * [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}}) \end{array} \right\}$

...

Continued in Figure 22.

Figure 20: Verification example bounded FIFO, initialisation. To lighten the notation, we do not show applications of the frame rule.

$$
\begin{aligned}
P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) \;:=\;& \exists v^{\mathsf{m}}_{\mathsf{c_p}}, v^{\mathsf{m}}_{\mathsf{c_c}}. \\
& [\tfrac{1}{2}]\ell_{\mathsf{c_p}} \mapsto v^{\mathsf{m}}_{\mathsf{c_p}} \;*\; [\tfrac{1}{2}]\ell_{\mathsf{c_c}} \mapsto v^{\mathsf{m}}_{\mathsf{c_c}} \;*\; 0 \leq v^{\mathsf{m}}_{\mathsf{c_p}} \leq 100 \;*\; 0 \leq v^{\mathsf{m}}_{\mathsf{c_c}} \leq 100 \\
& * \; \ell_{\mathsf{fifo}_{10}} \mapsto v^{\mathsf{m}}_{\mathsf{fifo}_{10}} \;*\; v^{\mathsf{m}}_{\mathsf{c_c}} = v^{\mathsf{m}}_{\mathsf{c_p}} + \mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) \;*\; 0 \leq \mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) \leq 10 \\
& * \; v^{\mathsf{m}}_{\mathsf{fifo}_{10}} = (v_{\mathsf{c_p}} + \mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}})) :: \ldots :: (v_{\mathsf{c_p}} + 1) :: \mathbf{nil} \\
& * \; \big(v^{\mathsf{m}}_{\mathsf{c_p}} > 0 \;\rightarrow\; \mathsf{signal}((id^{v^{\mathsf{m}}_{\mathsf{c_p}}}_{\mathrm{push}}, L^{v^{\mathsf{m}}_{\mathsf{c_p}}}_{\mathrm{push}}), \mathsf{False})\big) \\
& * \; \big(v^{\mathsf{m}}_{\mathsf{c_c}} > 0 \;\rightarrow\; \mathsf{signal}((id^{v^{\mathsf{m}}_{\mathsf{c_c}}}_{\mathrm{pop}}, L^{v^{\mathsf{m}}_{\mathsf{c_c}}}_{\mathrm{pop}}), \mathsf{False})\big) \\
P_{\mathsf{m}} \;:=\;& \exists v^{\mathsf{m}}_{\mathsf{fifo}_{10}}.\; P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}})
\end{aligned}
$$

Producer & consumer counters.
Bounded FIFO & its relationship to counters.
Signal set by producer.
Signal set by consumer.

Figure 21: Lock invariant

. . . 20 . . . — Continuation of Figure 20.

$$\left\{\begin{array}{l}
\mathsf{phase}(()) * \mathsf{obs}(\{\!\!\{s^{100}_{\mathrm{push}}, s^{100}_{\mathrm{pop}}\}\!\!\}) * [\tfrac{1}{2}]\ell_{\mathsf{c_p}} \mapsto 100 * [\tfrac{1}{2}]\ell_{\mathsf{c_c}} \mapsto 100 * \bigstar_{1,\ldots,200}\, \mathsf{itperm}((), 1) \\
* \bigstar_{i=1,\ldots,100} (\mathsf{wperm}((), id^i_{\mathrm{pop}}, 0) * \mathsf{wperm}((), id^i_{\mathrm{push}}, 0)) \\
* \bigstar_{i=1,\ldots,99} (\mathsf{uninitSig}(id^i_{\mathrm{push}}) * \mathsf{uninitSig}(id^i_{\mathrm{pop}})) * [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}}) * [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}})
\end{array}\right\}$$

fork (— PR-Fork

$$\left\{\begin{array}{l}
\mathsf{phase}((\mathtt{Forkee})) * \mathsf{obs}(\{\!\!\{s^{100}_{\mathrm{push}}\}\!\!\}) * [\tfrac{1}{2}]\ell_{\mathsf{c_p}} \mapsto 100 * \bigstar_{i=1,\ldots,100}\, \mathsf{itperm}((), 1) \\
* \bigstar_{i=1,\ldots,100} \mathsf{wperm}((), id^i_{\mathrm{pop}}, 0) * \bigstar_{i=1,\ldots,99} \mathsf{uninitSig}(s^i_{\mathrm{push}}) * [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}})
\end{array}\right\}$$

Resources transferred to producer thread.
PR-VS-Simp & VS-SemImp

$$\left\{\begin{array}{l}
\mathsf{phase}((\mathtt{Forkee})) * \mathsf{obs}(\{\!\!\{s^{100}_{\mathrm{push}}\}\!\!\}) * [\tfrac{1}{2}]\ell_{\mathsf{c_p}} \mapsto 100 * \bigstar_{i=1,\ldots,100}\, \mathsf{itperm}((), 1) \\
* \bigstar_{i=1,\ldots,100} \mathsf{wperm}((), id^i_{\mathrm{pop}}, 0) * \bigstar_{i=1,\ldots,99} \mathsf{uninitSig}(s^i_{\mathrm{push}}) * [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}}) \\
\exists v_{\mathsf{c_p}}, O_p.\; L_p(v_{\mathsf{c_p}}, O_p) * v_{\mathsf{c_p}} \neq 0
\end{array}\right\}$$

For definition of producer loop invariant $L_p(n, O)$ cf. Figure 23.

. . . 24 . . . — Producer loop on Figure 24.

$\{\mathsf{phase}((\mathtt{Forkee})) * \exists v_{\mathsf{c_p}}, O_p.\; L_p(v_{\mathsf{c_p}}, O_p) * v_{\mathsf{c_p}} \neq 0 \;\; \mathsf{obs}(\emptyset)\}$ — PR-VS-Simp & VS-SemImp

$\{\mathsf{phase}((\mathtt{Forkee})) * \mathsf{obs}(\emptyset)\}$

);

$$\left\{\begin{array}{l}
\mathsf{phase}(()) * \mathsf{obs}(\{\!\!\{s^{100}_{\mathrm{push}}, s^{100}_{\mathrm{pop}}\}\!\!\}) * [\tfrac{1}{2}]\ell_{\mathsf{c_p}} \mapsto 100 * [\tfrac{1}{2}]\ell_{\mathsf{c_c}} \mapsto 100 \\
* \bigstar_{1,\ldots,100}\, \mathsf{itperm}((), 1) * \bigstar_{1,\ldots,100}\, \mathsf{itperm}((), 1) \\
* \bigstar_{i=1,\ldots,100} \mathsf{wperm}((), id^i_{\mathrm{push}}, 0) * \bigstar_{i=1,\ldots,100} \mathsf{wperm}((), id^i_{\mathrm{pop}}, 0) \\
* \bigstar_{i=1,\ldots,99} \mathsf{uninitSig}(id^i_{\mathrm{pop}}) * \bigstar_{i=1,\ldots,99} \mathsf{uninitSig}(id^i_{\mathrm{push}}) * [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}}) * [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}})
\end{array}\right\}$$

Resources remaining with consumer thread.
PR-VS-Simp & VS-SemImp

$$\left\{\begin{array}{l}
\mathsf{phase}(()) * \mathsf{obs}(\{\!\!\{s^{100}_{\mathrm{pop}}\}\!\!\}) * [\tfrac{1}{2}]\ell_{\mathsf{c_c}} \mapsto 100 * \bigstar_{1,\ldots,100}\, \mathsf{itperm}((), 1) * \bigstar_{i=1,\ldots,100} \mathsf{wperm}((), id^i_{\mathrm{push}}, 0) \\
* \bigstar_{i=1,\ldots,99} \mathsf{uninitSig}(id^i_{\mathrm{pop}}) * [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}}) \;\; \exists v_{\mathsf{c_c}}, O_c.\; L_c(v_{\mathsf{c_c}}, O_c) * v_{\mathsf{c_c}} \neq 0
\end{array}\right\}$$

For definition of consumer loop invariant $L_c(n, O)$ cf. Figure 27.

. . . 28 . . . — Consumer loop on Fgure 28.

$\{\mathsf{phase}((\mathtt{Forker})) * \exists v_{\mathsf{c_c}}, O_c.\; L_c(v_{\mathsf{c_c}}, O_c) * v_{\mathsf{c_c}} \neq 0 \;\; \mathsf{obs}(\emptyset)\}$ — PR-VS-Simp & VS-SemImp

$\{\mathsf{phase}((\mathtt{Forker})) * \mathsf{obs}(\emptyset)\}$

Figure 22: Verification example bounded FIFO, forking.

$$
\begin{aligned}
L_p(n, O_p) \ :=\ & [\tfrac{1}{2}]\ell_{\mathsf{c_p}} \mapsto n \ * \ 0 \leq n \leq 100 \ * \ [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}}) \\
& * \ \mathsf{obs}(O_p) \ * \ (n > 0 \ \leftrightarrow \ O_p = \{\!\{s^n_{\text{push}}\}\!\}) \ * \ (n = 0 \ \leftrightarrow \ O_p = \emptyset) \\
& * \ \mathop{\Large\ast}\limits_{1,\ldots,n} \mathsf{itperm}((), 1) \\
& * \ \mathop{\Large\ast}\limits_{i=1,\ldots,100} \mathsf{wperm}((), id^i_{\text{pop}}, 0) \\
& * \ \mathop{\Large\ast}\limits_{i=1,\ldots,n-1} \mathsf{uninitSig}(id^i_{\text{push}})
\end{aligned}
$$

Loop invariant of producer.

Iteration permissions consumed by productive loop iterations, i.e., by iterations which decrease the producer counter $\mathsf{c_p}$.

Used to generate iteration permissions to justify unproductive loop iterations.

Remaining allocated signal IDs used to initialize new signal after next push.

$$
\begin{aligned}
L_p^{\text{locked}}(n, O_p) \ :=\ & [\tfrac{1}{2}]\ell_{\mathsf{c_p}} \mapsto n \ * \ 0 \leq n \leq 100 \\
& * \ [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}}) \ \ \mathsf{locked}(mut, P_{\mathsf{m}}, \tfrac{1}{2}) \ * \ \mathsf{obs}(O_p \ \uplus \{\!\{mut\}\!\}) \\
& * \ (n > 0 \ \leftrightarrow \ O_p = \{\!\{s^n_{\text{push}}\}\!\}) \ * \ (n = 0 \ \leftrightarrow \ O_p = \emptyset) \\
& * \ \mathop{\Large\ast}\limits_{1,\ldots,n} \mathsf{itperm}((), 1) \ * \ \mathop{\Large\ast}\limits_{i=1,\ldots,100} \mathsf{wperm}((), id^i_{\text{pop}}, 0) * \mathop{\Large\ast}\limits_{i=1,\ldots,n-1} \mathsf{uninitSig}(id^i_{\text{push}})
\end{aligned}
$$

Shorthand for invariant with acquired mutex.

$$
\begin{aligned}
L_p^{\substack{\text{no:mutex}\\ \text{no:obs}}}(n, O_p) \ :=\ & \exists id^n_{\text{push}}. \\
& [\tfrac{1}{2}]\ell_{\mathsf{c_p}} \mapsto n \ * \ 0 \leq n \leq 100 \ \ * \ [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}}) \ * \ \mathsf{obs}(O_p) \\
& * \ (n > 0 \ \leftrightarrow \ O_p = \{\!\{s^n_{\text{push}}\}\!\}) \ * \ (n = 0 \ \leftrightarrow \ O_p = \emptyset) \\
& * \ \mathop{\Large\ast}\limits_{1,\ldots,n} \mathsf{itperm}((), 1) \ * \ \mathop{\Large\ast}\limits_{i=1,\ldots,100} \mathsf{wperm}((), id^i_{\text{pop}}, 0) * \mathop{\Large\ast}\limits_{i=1,\ldots,n-1} \mathsf{uninitSig}(id^i_{\text{push}})
\end{aligned}
$$

Shorthand for invariant without mutex chunk and without obligations chunk.

Figure 23: Producer's loop invariant.

$\forall \ell_{\mathsf{fifo}_{10}}, \ell_{\mathsf{m}}, \ell_{\mathsf{c_p}}, \ell_{\mathsf{c_c}}.$

...

$\{\mathsf{phase}((\mathtt{Forkee})) * \exists v_{\mathsf{c_p}}, O_p.\ L_p(v_{\mathsf{c_p}}, O_p) * v_{\mathsf{c_p}} \neq 0\}$

while (For definition of producer loop invariant $L_p(n, O)$, lock invariant P_{m} and variations cf. Figures 23 and 21.

$\forall v_{\mathsf{c_p}}, O_p.$

$\{\mathsf{phase}((\mathtt{Forkee})) * \boxed{\exists v_{\mathsf{c_p}}, O_p.}\ L_p(v_{\mathsf{c_p}}, O_p) * v_{\mathsf{c_p}} \neq 0 * \boxed{\forall o \in O_p.\ \mathsf{lev}(o) = L^{v_{\mathsf{c_p}}}_{\mathrm{push}}}\}$

$\mathsf{lev}(mut) \ = \ 0 \ < \ 101 - v_{\mathsf{c_p}} \ = \ L^{v_{\mathsf{c_p}}}_{\mathrm{push}}$

acquire m;

$\{\mathsf{phase}((\mathtt{Forkee})) * \boxed{L_p(v_{\mathsf{c_p}}, O_p)}\ \boxed{L^{\mathrm{locked}}_p(v_{\mathsf{c_p}}, O_p) * P_{\mathsf{m}}} * v_{\mathsf{c_p}} \neq 0\ \boxed{* \forall o \in O_p.\ \mathsf{lev}(o) = \ldots}\}$

$\forall v^{\mathsf{m}}_{\mathsf{fifo}_{10}}.$

$\{\boxed{P_{\mathsf{m}}}\ \boxed{P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}})} * \ldots\}$

let $\mathsf{f} := [\mathsf{fifo}_{10}]$ **in**

$\{\mathsf{phase}((\mathtt{Forkee})) * L^{\mathrm{locked}}_p(v_{\mathsf{c_p}}, O_p) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * v_{\mathsf{c_p}} \neq 0\}$

if size$(\mathsf{f}) < 10$ **then** (

$\{\boxed{\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) < 10} * \mathsf{phase}((\mathtt{Forkee})) * L^{\mathrm{locked}}_p(v_{\mathsf{c_p}}, O_p) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * v_{\mathsf{c_p}} \neq 0\}$

...25... Production step presented on Figure 25.

$$\left\{\begin{array}{l} \boxed{\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) < 10} * \mathsf{phase}((\mathtt{Forkee}))\ \boxed{* L^{\mathrm{locked}}_p(v_{\mathsf{c_p}}, O_p) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) *} \\ \boxed{\exists v'_{\mathsf{c_p}}, O'_p.\ \mathsf{obs}(O'_p \uplus \{\!\{mut\}\!\}) * \mathsf{locked}(mut, P_{\mathsf{m}}, \tfrac{1}{2}) *} * v_{\mathsf{c_p}} \neq 0 \\ \boxed{* (\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) < 10 \ \rightarrow \ L^{\substack{\text{no:mutex} \\ \text{no:obs}}}_p(v'_{\mathsf{c_p}}, O'_p) * P_{\mathsf{m}} * \mathsf{itperm}((), 1))} \\ \boxed{* (\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 10 \ \rightarrow \ L^{\substack{\text{no:mutex} \\ \text{no:obs}}}_p(v_{\mathsf{c_p}}, O_p) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * O'_p = O_p)} \end{array}\right\}$$

) **else** (

$\{\boxed{\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 10} * \mathsf{phase}((\mathtt{Forkee})) * L^{\mathrm{locked}}_p(v_{\mathsf{c_p}}, O_p) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * v_{\mathsf{c_p}} \neq 0\}$

$$\left\{\begin{array}{l} \boxed{\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 10} * \mathsf{phase}((\mathtt{Forkee}))\ \boxed{* L^{\mathrm{locked}}_p(v_{\mathsf{c_p}}, O_p) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) *} \\ \boxed{v_{\mathsf{c_p}} \neq 0}\ \boxed{\exists v'_{\mathsf{c_p}}, O'_p.\ \mathsf{obs}(O'_p \uplus \{\!\{mut\}\!\}) * \mathsf{locked}(mut, P_{\mathsf{m}}, \tfrac{1}{2}) * \mathsf{PostIf}_p} \end{array}\right\}$$

);

$$\left\{\begin{array}{l} \mathsf{phase}((\mathtt{Forkee})) * \boxed{L^{\mathrm{locked}}_p(v_{\mathsf{c_p}}, O_p) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * v_{\mathsf{c_p}} \neq 0} \\ \boxed{\exists v'_{\mathsf{c_p}}, O'_p.\ \mathsf{obs}(O'_p \uplus \{\!\{mut\}\!\}) * \mathsf{locked}(mut, P_{\mathsf{m}}, \tfrac{1}{2}) * \mathsf{PostIf}_p} \end{array}\right\}$$

$\forall v'_{\mathsf{c_p}}, O'_p.$

release m; Wait step presented on Figure 26, i.e., view shift performed after releasing mut but before consuming P_{m}.

$\forall \tau^{\mathrm{anc}}_p, \delta_p.$

let $\mathsf{c} := [\mathsf{c_p}]$ **in** $\mathsf{c} \neq 0$

$$\left\{\begin{array}{l} \tau^{\mathrm{anc}}_p \sqsubseteq (\mathtt{Forkee}) * \mathsf{phase}((\mathtt{Forkee})) \\ * (v'_{\mathsf{c_p}} \neq 0 \ \rightarrow \ \exists v_{\mathsf{c_p}}, O_p.\ L_p(v_{\mathsf{c_p}}, O_p) * v_{\mathsf{c_p}} \neq 0 * \mathsf{itperm}(\tau^{\mathrm{anc}}_p, \delta_p)) * (v'_{\mathsf{c_p}} = 0 \ \rightarrow \ \mathsf{obs}(\emptyset)) \end{array}\right\}$$

) **do skip**

$\{\mathsf{phase}((\mathtt{Forkee})) * \boxed{\exists v_{\mathsf{c_p}}.\ L_p(v_{\mathsf{c_p}}) * v_{\mathsf{c_p}} \neq 0}\ \boxed{\mathsf{obs}(\emptyset)}\}$

...

Continuation of Figure 22.

PR-While-Simp & PR-Exists (2x) & PR-VS-Simp & VS-SemImp

$O_p = \{\!\{s^{v_{\mathsf{c_p}}}_{\mathrm{push}}\}\!\} \ \vee \ O_p = \emptyset$

Justification for application of:

PR-Acquire

PR-Exists

PR-Let & PR-ReadHeapLoc

PR-If

Define PostIf_p such that:

$$= \left\{\begin{array}{l} \mathsf{phase}((\mathtt{Forkee})) \\ * \exists v'_{\mathsf{c_p}}, O'_p.\ \mathsf{obs}(O'_p \uplus \{\!\{mut\}\!\}) \\ * \mathsf{locked}(mut, P_{\mathsf{m}}, \tfrac{1}{2}) * \mathsf{PostIf}_p \end{array}\right\}$$

PR-VS-Simp & VS-SemImp

PR-Exists (2x)

PR-Release & PR-Exists (2x)

PR-Let & PR-ReadHeapLoc & PR-Exp

Reestablished loop invariant.

Continued in Figure 22.

Figure 24: Verification example bounded FIFO, producer loop.

$\forall \ell_{\mathsf{fifo}_{10}}, \ell_{\mathsf{m}}, \ell_{\mathsf{c_p}}, \ell_{\mathsf{c_c}}, v_{\mathsf{c_p}}, O_p, v^{\mathsf{m}}_{\mathsf{fifo}_{10}}.$

...

Continuation of Figure 24.

For definition of P_{m}, $P'_{\mathsf{m}}(v)$, $L^{\text{locked}}_p(n, O)$ and $L_p{}^{\text{no:mutex}}_{\text{no:obs}}(n, O)$ cf. Figures 21 and 23.

$\{\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) < 10 * \mathsf{phase}((\mathtt{Forkee})) * L^{\text{locked}}_p(v_{\mathsf{c_p}}, O_p) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * v_{\mathsf{c_p}} \neq 0\}$ — PR-VS-Simp & VS-SemImp

$$\left\{\begin{array}{l} [\tfrac{1}{2}]\ell_{\mathsf{c_p}} \mapsto v_{\mathsf{c_p}} * [\tfrac{1}{2}]\ell_{\mathsf{c_p}} \mapsto v^{\mathsf{m}}_{\mathsf{c_p}} \quad \ell_{\mathsf{c_p}} \mapsto v_{\mathsf{c_p}} * v_{\mathsf{c_p}} = v^{\mathsf{m}}_{\mathsf{c_p}} \\ * \ (v_{\mathsf{c_p}} > 0 \ \rightarrow \ \mathsf{signal}(s^{v_{\mathsf{c_p}}}_{\text{push}}, \mathsf{False})) \quad \mathsf{signal}(s^{v_{\mathsf{c_p}}}_{\text{push}}, \mathsf{False}) \\ * \ (v_{\mathsf{c_p}} > 0 \ \leftrightarrow \ O_p = \{\!\{s^{v_{\mathsf{c_p}}}_{\text{push}}\}\!\}) * (v_{\mathsf{c_p}} = 0 \ \leftrightarrow \ O_p = \emptyset) \quad O_p = \{\!\{s^{v_{\mathsf{c_p}}}_{\text{push}}\}\!\} * \ldots \end{array}\right\}$$

let $\mathsf{c} := [\mathsf{c_p}]$ **in** — PR-Let & PR-ReadHeapLoc

$[\mathsf{fifo}_{10}] := \mathsf{f} \cdot (\mathsf{c} :: \mathbf{nil})$; $[\mathsf{c_p}] := \mathsf{c} - 1$ — PR-AssignToHeap (2x)

$\{\ell_{\mathsf{fifo}_{10}} \mapsto v^{\mathsf{m}}_{\mathsf{fifo}_{10}} \cdot (v_{\mathsf{c_p}} :: \mathbf{nil}) * \ell_{\mathsf{c_p}} \mapsto v_{\mathsf{c_p}} - 1 * \ldots\}$ — PR-VS-Simp & VS-SetSignal

$\{\mathsf{obs}(\{\!\{s^{v_{\mathsf{c_p}}}_{\text{push}}, \mathit{mut}\}\!\}) * \mathsf{signal}(s^{v_{\mathsf{c_p}}}_{\text{push}}, \mathsf{True}) * \ldots\}$ — PR-VS-Simp & VS-SemImp

$\{(v_{\mathsf{c_p}} - 1 = 0 \vee v_{\mathsf{c_p}} > 0) * \ldots\}$ — PR-VS-Simp & VS-Or

case: $v_{\mathsf{c_p}} - 1 = 0$ — Last iteration, nothing left to do. — PR-VS-Simp & VS-SemImp

$$\left\{\begin{array}{l} \mathsf{phase}((\mathtt{Forkee})) * \exists v'_{\mathsf{c_p}}, O'_p.\ \mathsf{obs}(O'_p \uplus \{\!\{\mathit{mut}\}\!\}) * \mathsf{locked}(\mathit{mut}, P_{\mathsf{m}}, \tfrac{1}{2}) * v_{\mathsf{c_p}} \neq 0 \\ * \left(\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) < 10 \ \rightarrow \ L_p{}^{\text{no:mutex}}_{\text{no:obs}}(v'_{\mathsf{c_p}}, O'_p) * P_{\mathsf{m}} * \mathsf{itperm}((), 1)\right) \\ * \left(\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 10 \ \rightarrow \ L_p{}^{\text{no:mutex}}_{\text{no:obs}}(v_{\mathsf{c_p}}, O_p) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * O'_p = O_p\right) \end{array}\right\}$$

$$= \left\{\begin{array}{l} \mathsf{phase}((\mathtt{Forkee})) \\ * \exists v'_{\mathsf{c_p}}, O'_p.\ \mathsf{obs}(O'_p \uplus \{\!\{\mathit{mut}\}\!\}) \\ * \mathsf{locked}(\mathit{mut}, P_{\mathsf{m}}, \tfrac{1}{2}) * \mathsf{PostIf}_p \end{array}\right\}$$

For definition of PostIf_p cf. Figure 24.

case: $v_{\mathsf{c_p}} - 1 > 0$ — Must create signal for next iteration. — PR-VS-Simp & VS-SigInit

$$\left\{\begin{array}{l} \mathsf{obs}(\{\!\{id^{v_{\mathsf{c_p}}-1}_{\text{push}}, \mathit{mut}\}\!\}) * \mathop{\Large\ast}_{i=1,\ldots,v_{\mathsf{c_p}}-2} \mathsf{uninitSig}(id^{i}_{\text{push}}) \\ * \ \mathsf{uninitSig}(id^{v_{\mathsf{c_p}}-1}_{\text{push}}) \quad \mathsf{signal}(s^{v_{\mathsf{c_p}}-1}_{\text{push}}, \mathsf{False}) * \ldots \end{array}\right\}$$

PR-VS-Simp & VS-SemImp

$\{\mathsf{phase}((\mathtt{Forkee})) * \exists v'_{\mathsf{c_p}}, O'_p.\ \mathsf{obs}(O'_p \uplus \{\!\{\mathit{mut}\}\!\}) * \mathsf{locked}(\mathit{mut}, P_{\mathsf{m}}, \tfrac{1}{2}) * \mathsf{PostIf}_p\}$

$\{\mathsf{phase}((\mathtt{Forkee})) * \exists v'_{\mathsf{c_p}}, O'_p.\ \mathsf{obs}(O'_p \uplus \{\!\{\mathit{mut}\}\!\}) * \mathsf{locked}(\mathit{mut}, P_{\mathsf{m}}, \tfrac{1}{2}) * \mathsf{PostIf}_p\}$

...

Continued in Figure 24.

Figure 25: Verification example bounded FIFO, producer thread's production step.

$\forall \ell_{\mathsf{fifo}_{10}}, \ell_{\mathsf{m}}, \ell_{\mathsf{c_p}}, \ell_{\mathsf{c_c}}, v_{\mathsf{c_p}}, O_p, v^{\mathsf{m}}_{\mathsf{fifo}_{10}}, v'_{\mathsf{c_p}}, O'_p.$

Continuation of Figure 24.

...

For definition of P_{m}, $P'_{\mathsf{m}}(v)$, $L_p^{\text{locked}}(n, O)$, $L_p^{\overset{\text{no:mutex}}{\text{no:obs}}}(n, O)$, PostIf_p cf. Figures 21, 23 and 24.

$\{\mathsf{phase}((\texttt{Forkee})) * \mathsf{obs}(O'_p \uplus \{\!\{mut\}\!\}) * \mathsf{locked}(mut, P_{\mathsf{m}}, \frac{1}{2}) * \mathsf{PostIf}_p\}$

release m — PR-Release allows view shift to happen after mutex mut was released but before lock invariant P_{m} is consumed.

$\{\mathsf{phase}((\texttt{Forkee})) * \mathsf{obs}(O'_p \; \boxed{\uplus \{\!\{mut\}\!\}}\;) \; \boxed{* \mathsf{locked}(mut, P_{\mathsf{m}}, \frac{1}{2})} * \mathsf{PostIf}_p\}$ — PR-VS-Simp & VS-SemImp

$\{(\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) < 10 \vee \mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 10) * \ldots\}$ — PR-VS-Simp & VS-Or

case: $\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) < 10$ — Production step already performed, nothing left to do.

$\{\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) < 10 * \mathsf{phase}((\texttt{Forkee})) * \mathsf{obs}(O'_p) * \mathsf{PostIf}_p\}$ — PR-VS-Simp & VS-SemImp

$$\left\{\begin{array}{l} \mathsf{obs}(O'_p) * P_{\mathsf{m}} * \exists \tau_p^{\text{anc}}, \delta_p.\ \tau_p^{\text{anc}} \sqsubseteq (\texttt{Forkee}) * \mathsf{phase}((\texttt{Forkee})) \\ * L_p^{\overset{\text{no:mutex}}{\text{no:obs}}}(v'_{\mathsf{c_p}}, O'_p) * \mathsf{itperm}(\tau_p^{\text{anc}}, \delta_p) \end{array}\right\}$$

Define $\mathsf{PostReleaseVS}_p$ such that: $= \{\mathsf{obs}(O'_p) * P_{\mathsf{m}} * \mathsf{PostReleaseVS}_p\}$

case: $\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 10$ — No production step performed. Must wait to generate permission.

$\{\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 10 * \mathsf{phase}((\texttt{Forkee})) * \mathsf{obs}(O'_p) * \mathsf{PostIf}_p\}$ — PR-VS-Simp & VS-SemImp

$$\left\{\begin{array}{l} \boxed{\mathsf{PostIf}_p} \; L_p^{\overset{\text{no:mutex}}{\text{no:obs}}}(v_{\mathsf{c_p}}, O_p) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * O'_p = O_p = \{\!\{s^{v_{\mathsf{c_p}}}_{\text{push}}\}\!\} \\ * \mathsf{obs}(\{\!\{s^{v_{\mathsf{c_p}}}_{\text{push}}\}\!\}) * v_{\mathsf{c_p}} \neq 0 * \mathsf{phase}((\texttt{Forkee})) \end{array}\right\}$$

PR-Exists (2x) & PR-VS-Simp & VS-SemImp

$\forall v^{\mathsf{m}}_{\mathsf{c_p}}, v^{\mathsf{m}}_{\mathsf{c_c}}.$

$$\left\{\begin{array}{l} \boxed{\exists v^{\mathsf{m}}_{\mathsf{c_p}}.} \; [\frac{1}{2}]\ell_{\mathsf{c_p}} \mapsto v_{\mathsf{c_p}} * [\frac{1}{2}]\ell_{\mathsf{c_p}} \mapsto v^{\mathsf{m}}_{\mathsf{c_p}} * v_{\mathsf{c_p}} = v^{\mathsf{m}}_{\mathsf{c_p}} * v^{\mathsf{m}}_{\mathsf{c_c}} = v_{\mathsf{c_p}} + 10 \\ \boxed{\exists v^{\mathsf{m}}_{\mathsf{c_c}}.\ (v^{\mathsf{m}}_{\mathsf{c_c}} > 0 \ \rightarrow\ \mathsf{signal}(s^{v^{\mathsf{m}}_{\mathsf{c_c}}}_{\text{pop}}, \mathsf{False}))} \; \mathsf{signal}(s^{v^{\mathsf{m}}_{\mathsf{c_c}}}_{\text{pop}}, \mathsf{False}) \\ * \mathop{\LARGE *}_{i=1,\ldots,100} \mathsf{wperm}((), id^{i}_{\text{pop}}, 0) * \mathsf{obs}(\{\!\{s^{v_{\mathsf{c_p}}}_{\text{push}}\}\!\}) * \mathsf{phase}((\texttt{Forkee})) * \ldots \end{array}\right\}$$

PR-VS-Simp & VS-Wait

$\mathsf{lev}(s^{v^{\mathsf{m}}_{\mathsf{c_c}}}_{\text{pop}}) \;=\; L^{v^{\mathsf{m}}_{\mathsf{c_c}}}_{\text{pop}} \;=\; L^{v_{\mathsf{c_p}}+10}_{\text{pop}} \;=\; 102 - (v_{\mathsf{c_p}} + 10)$
$< \; 101 - v_{\mathsf{c_p}} \;=\; L^{v_{\mathsf{c_p}}}_{\text{push}} \;=\; \mathsf{lev}(s^{v_{\mathsf{c_p}}}_{\text{push}})$

Justification for application of VS-Wait.

$\{\mathsf{itperm}((\texttt{Forkee}), 0) * \ldots\}$ — PR-VS-Simp & VS-SemImp

$\{\mathsf{obs}(O'_p) * P_{\mathsf{m}} * \mathsf{PostReleaseVS}_p\}$

$$\left\{\begin{array}{l} \boxed{\mathsf{phase}((\texttt{Forkee}))*} \; \mathsf{obs}(O'_p) \; \boxed{*\mathsf{PostIf}_p * (\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) < 10 \vee \mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 10)} \\ *P_{\mathsf{m}} * \mathsf{PostReleaseVS}_p \end{array}\right\}$$

Conclusion of VS-Or application.

$\{\mathsf{obs}(O'_p) \; \boxed{*P_{\mathsf{m}}} * \mathsf{PostReleaseVS}_p\}$ — Lock invariant P_{m} consumed by PR-Release. — PR-VS-Simp & VS-SemImp

$$\left\{\begin{array}{l} \exists \tau_p^{\text{anc}}, \delta_p.\ \tau_p^{\text{anc}} \sqsubseteq (\texttt{Forkee}) * \mathsf{phase}((\texttt{Forkee})) \\ * (v'_{\mathsf{c_p}} \neq 0 \ \rightarrow\ \exists v_{\mathsf{c_p}}, O_p.\ L_p(v_{\mathsf{c_p}}, O_p) * v_{\mathsf{c_p}} \neq 0 * \mathsf{itperm}(\tau_p^{\text{anc}}, \delta_p)) * (v'_{\mathsf{c_p}} = 0 \ \rightarrow\ \mathsf{obs}(\emptyset)) \end{array}\right\}$$

Reestablished loop invariant.

...

Continued in Figure 24.

Figure 26: Verification example bounded FIFO, producer's wait step.

$$
\begin{aligned}
L_c(n, O_c) \;:=\;& \\
& [\tfrac{1}{2}]\ell_{\mathsf{c_c}} \mapsto n \;*\; 0 \leq n \leq 100 \;*\; [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}}) \\
& *\; \mathsf{obs}(O_c) \;*\; (n > 0 \;\leftrightarrow\; O_c = \{\!\!\{ s^n_{\mathrm{pop}} \}\!\!\}) \;*\; (n = 0 \;\leftrightarrow\; O_c = \emptyset) \\
& *\; \mathop{\Huge *}_{1,\ldots,n} \mathsf{itperm}((), 1) \\
& *\; \mathop{\Huge *}_{i=1,\ldots,100} \mathsf{wperm}((), id^i_{\mathrm{push}}, 0) \\
& *\; \mathop{\Huge *}_{i=1,\ldots,n-1} \mathsf{uninitSig}(id^i_{\mathrm{pop}})
\end{aligned}
$$

Loop invariant of consumer.

Iteration permissions consumed by productive loop iterations, i.e., by iterations which decrease the consumer counter $\mathsf{c_c}$.

Used to generate iteration permissions to justify unproductive loop iterations.

Remaining allocated signal IDs used to initialize new signal after next pop.

$$
\begin{aligned}
L_c^{\mathrm{locked}}(n, O_p) \;:=\;& \\
& [\tfrac{1}{2}]\ell_{\mathsf{c_c}} \mapsto n \;*\; 0 \leq n \leq 100 \\
& *\; [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}}) \; \mathsf{locked}(mut, P_{\mathsf{m}}, \tfrac{1}{2}) \;*\; \mathsf{obs}(O_c \; \uplus \{\!\!\{ mut \}\!\!\}) \\
& *\; (n > 0 \;\leftrightarrow\; O_c = \{\!\!\{ s^n_{\mathrm{pop}} \}\!\!\}) \;*\; (n = 0 \;\leftrightarrow\; O_c = \emptyset) \\
& *\; \mathop{\Huge *}_{1,\ldots,n} \mathsf{itperm}((), 1) \;*\; \mathop{\Huge *}_{i=1,\ldots,100} \mathsf{wperm}((), id^i_{\mathrm{push}}, 0) \;*\; \mathop{\Huge *}_{i=1,\ldots,n-1} \mathsf{uninitSig}(id^i_{\mathrm{pop}})
\end{aligned}
$$

Shorthand for invariant with acquired mutex.

$$
\begin{aligned}
L_c{}^{\mathsf{no:mutex}}_{\mathsf{no:obs}}(n, O_p) \;:=\;& \exists id^n_{\mathrm{pop}}. \\
& [\tfrac{1}{2}]\ell_{\mathsf{c_c}} \mapsto n \;*\; 0 \leq n \leq 100 \; *\; [\tfrac{1}{2}]\mathsf{mutex}(mut, P_{\mathsf{m}}) \;*\; \mathsf{obs}(O_c) \\
& *\; (n > 0 \;\leftrightarrow\; O_c = \{\!\!\{ s^n_{\mathrm{pop}} \}\!\!\}) \;*\; (n = 0 \;\leftrightarrow\; O_c = \emptyset) \\
& *\; \mathop{\Huge *}_{1,\ldots,n} \mathsf{itperm}((), 1) \;*\; \mathop{\Huge *}_{i=1,\ldots,100} \mathsf{wperm}((), id^i_{\mathrm{push}}, 0) \;*\; \mathop{\Huge *}_{i=1,\ldots,n-1} \mathsf{uninitSig}(id^i_{\mathrm{pop}})
\end{aligned}
$$

Shorthand for invariant without mutex chunk and without obligations chunk.

Figure 27: Consumer's loop invariant.

$\forall \ell_{\mathsf{fifo}_{10}}, \ell_{\mathsf{m}}, \ell_{\mathsf{c_p}}, \ell_{\mathsf{c_c}}.$

...

$\{\mathsf{phase}((\mathtt{Forker})) * \exists v_{\mathsf{c_c}}, O_c.\ L_c(v_{\mathsf{c_c}}, O_c) * v_{\mathsf{c_c}} \neq 0\}$

while (

For definition of consumer loop invariant $L_p(n, O)$, lock invariant P_{m} and variations cf. Figures 27 and 21.

$\forall v_{\mathsf{c_c}}, O_c.$

$\{\mathsf{phase}((\mathtt{Forker})) * \exists v_{\mathsf{c_p}}, O_c.\ L_c(v_{\mathsf{c_c}}, O_c) * v_{\mathsf{c_c}} \neq 0 * \forall o \in O_c.\ \mathsf{lev}(o) = L_{\text{pop}}^{v_{\mathsf{c_c}}}\}$

$\mathsf{lev}(mut) \ = \ 0 \ < \ 102 - v_{\mathsf{c_c}} \ = \ L_{\text{pop}}^{v_{\mathsf{c_c}}}$

acquire m

$\{\mathsf{phase}((\mathtt{Forker})) * L_c(v_{\mathsf{c_c}}, O_c)\ \ L_c^{\text{locked}}(v_{\mathsf{c_c}}, O_c) * P_{\mathsf{m}} * v_{\mathsf{c_c}} \neq 0 * \forall o \in O_c.\ \mathsf{lev}(o) = \ldots\}$

$\forall v^{\mathsf{m}}_{\mathsf{fifo}_{10}}.$

$\{P_{\mathsf{m}}\ \ P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * \ldots\}$

let f :=[fifo$_{10}$] **in**

$\{\mathsf{phase}((\mathtt{Forker})) * L_c^{\text{locked}}(v_{\mathsf{c_c}}, O_c) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * v_{\mathsf{c_c}} \neq 0\}$

if size(f) > 0 **then** (

$\{\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) > 0 * \mathsf{phase}((\mathtt{Forker})) * L_c^{\text{locked}}(v_{\mathsf{c_c}}, O_c) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * v_{\mathsf{c_c}} \neq 0\}$

...29... Consumption step presented on Figure 29.

$$\left\{\begin{array}{l} \mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) > 0 * \mathsf{phase}((\mathtt{Forker})) * L_c^{\text{locked}}(v_{\mathsf{c_c}}, O_c) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * \\ \exists v'_{\mathsf{c_c}}, O'_c.\ \mathsf{obs}(O'_c \uplus \{\!\{mut\}\!\}) * \mathsf{locked}(mut, P_{\mathsf{m}}, \tfrac{1}{2}) * \ * v_{\mathsf{c_c}} \neq 0 \\ * \left(\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) > 0 \ \rightarrow \ L_c^{\substack{\text{no:mutex}\\ \text{no:obs}}}(v'_{\mathsf{c_c}}, O'_c) * P_{\mathsf{m}} * \mathsf{itperm}((), 1)\right) \\ * \left(\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 0 \ \rightarrow \ L_c^{\substack{\text{no:mutex}\\ \text{no:obs}}}(v_{\mathsf{c_c}}, O_c) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * O'_c = O_c\right) \end{array}\right\}$$

) **else** (

$\{\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 0 * \mathsf{phase}((\mathtt{Forker})) * L_c^{\text{locked}}(v_{\mathsf{c_c}}, O_c) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * v_{\mathsf{c_c}} \neq 0\}$

$$\left\{\begin{array}{l} \mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 0 * \mathsf{phase}((\mathtt{Forker})) * L_c^{\text{locked}}(v_{\mathsf{c_c}}, O_c) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * \\ v_{\mathsf{c_c}} \neq 0\ \ \exists v'_{\mathsf{c_c}}, O'_c.\ \mathsf{obs}(O'_c \uplus \{\!\{mut\}\!\}) * \mathsf{locked}(mut, P_{\mathsf{m}}, \tfrac{1}{2}) * \mathsf{PostIf}_c \end{array}\right\}$$

);

$$\left\{\begin{array}{l} \mathsf{phase}((\mathtt{Forker})) * L_c^{\text{locked}}(v_{\mathsf{c_c}}, O_c) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * v_{\mathsf{c_c}} \neq 0 \\ \exists v'_{\mathsf{c_c}}, O'_c.\ \mathsf{obs}(O'_c \uplus \{\!\{mut\}\!\}) * \mathsf{locked}(mut, P_{\mathsf{m}}, \tfrac{1}{2}) * \mathsf{PostIf}_c \end{array}\right\}$$

$\forall v'_{\mathsf{c_c}}, O'_c.$

release m;

Wait step presented on Figure 30, i.e., view shift performed after releasing mut but before consuming P_{m}.

$\forall \tau_c^{\text{anc}}, \delta_c.$

let c :=[c$_c$] **in** c $\neq 0$

$$\left\{\begin{array}{l} \tau_c^{\text{anc}} \sqsubseteq (\mathtt{Forker}) * \mathsf{phase}((\mathtt{Forker})) \\ * (v'_{\mathsf{c_c}} \neq 0 \ \rightarrow \ \exists v_{\mathsf{c_p}}, O_c.\ L_c(v_{\mathsf{c_c}}, O_c) * v_{\mathsf{c_c}} \neq 0 * \mathsf{itperm}(\tau_c^{\text{anc}}, \delta_c)) * (v'_{\mathsf{c_c}} = 0 \ \rightarrow \ \mathsf{obs}(\emptyset)) \end{array}\right\}$$

) **do skip**

$\{\mathsf{phase}((\mathtt{Forker})) * \exists v_{\mathsf{c_p}}.\ L_c(v_{\mathsf{c_p}}) * v_{\mathsf{c_c}} \neq 0\ \ \mathsf{obs}(\emptyset)\}$

...

Continuation of Figure 22.

PR-While-Simp & PR-Exists (2x) & PR-VS-Simp & VS-SemImp

$O_c = \{\!\{s_{\text{pop}}^{v_{\mathsf{c_c}}}\}\!\} \ \vee \ O_c = \emptyset$

Justification for application of:

PR-Acquire

PR-Exists

PR-Let & PR-ReadHeapLoc

PR-If

Define PostIf_c such that:

$$= \left\{\begin{array}{l} \mathsf{phase}((\mathtt{Forker})) \\ * \exists v'_{\mathsf{c_c}}, O'_c.\ \mathsf{obs}(O'_c \uplus \{\!\{mut\}\!\}) \\ * \mathsf{locked}(mut, P_{\mathsf{m}}, \tfrac{1}{2}) * \mathsf{PostIf}_c \end{array}\right\}$$

PR-VS-Simp & VS-SemImp

PR-Exists (2x)

PR-Release & PR-Exists (2x)

PR-Let & PR-ReadHeapLoc & PR-Exp

Reestablished loop invariant.

Continued in Figure 22.

Figure 28: Verification example bounded FIFO, consumer loop.

$\forall \ell_{\mathsf{fifo}_{10}}, \ell_{\mathsf{m}}, \ell_{\mathsf{c_p}}, \ell_{\mathsf{c_c}}, v_{\mathsf{c_c}}, O_c, v^{\mathsf{m}}_{\mathsf{fifo}_{10}}.$

$\ldots$

Continuation of Figure 28.

For definition of P_{m}, $P'_{\mathsf{m}}(v)$, $L_c^{\text{locked}}(n, O)$ and $L_c{}^{\substack{\text{no:mutex}\\ \text{no:obs}}}(n, O)$ cf. Figures 21 and 27.

$\{\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) > 0 * \mathsf{phase}((\mathtt{Forker})) * L_c^{\text{locked}}(v_{\mathsf{c_c}}, O_c) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * v_{\mathsf{c_c}} \neq 0\}$ — PR-Exists

$\forall v^{\mathsf{m}}_{\mathsf{c_c}}.$ — PR-VS-Simp & VS-SemImp

$$\left\{\begin{array}{l} [\tfrac{1}{2}]\ell_{\mathsf{c_c}} \mapsto v_{\mathsf{c_c}} * [\tfrac{1}{2}]\ell_{\mathsf{c_c}} \mapsto v^{\mathsf{m}}_{\mathsf{c_c}} \quad \ell_{\mathsf{c_c}} \mapsto v_{\mathsf{c_c}} * v_{\mathsf{c_c}} = v^{\mathsf{m}}_{\mathsf{c_c}} \\ * (v^{\mathsf{m}}_{\mathsf{c_c}} > 0 \ \rightarrow\ \mathsf{signal}(s^{v^{\mathsf{m}}_{\mathsf{c_c}}}_{\text{pop}}, \mathsf{False})) \quad \mathsf{signal}(s^{v_{\mathsf{c_c}}}_{\text{pop}}, \mathsf{False}) \\ * \ (v_{\mathsf{c_c}} > 0 \ \leftrightarrow\ O_c = \{\!\!\{ s^{v_{\mathsf{c_c}}}_{\text{pop}} \}\!\!\}) * (v_{\mathsf{c_c}} = 0 \ \leftrightarrow\ O_c = \emptyset) \quad O_c = \{\!\!\{ s^{v_{\mathsf{c_c}}}_{\text{pop}} \}\!\!\} * \ldots \end{array}\right\}$$

let $\mathsf{c} := [\mathsf{c_c}]$ **in** — PR-Let & PR-ReadHeapLoc

$[\mathsf{fifo}_{10}] := \mathbf{tail}(\mathsf{f});\ \ [\mathsf{c_c}] := \mathsf{c} - 1$ — PR-AssignToHeap (2x)

$\{\ell_{\mathsf{fifo}_{10}} \mapsto \mathbf{tail}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * \ell_{\mathsf{c_c}} \mapsto v_{\mathsf{c_c}} - 1 * \ldots\}$ — PR-VS-Simp & VS-SetSignal

$\{\mathsf{obs}(\{\!\!\{ s^{v_{\mathsf{c_c}}}_{\text{pop}}, \mathit{mut} \}\!\!\}) * \mathsf{signal}(s^{v_{\mathsf{c_c}}}_{\text{pop}}, \mathsf{True}) * \ldots\}$ — PR-VS-Simp & VS-SemImp

$\{(v_{\mathsf{c_c}} - 1 = 0 \vee v_{\mathsf{c_c}} > 0) * \ldots\}$ — PR-VS-Simp & VS-Or

case: $v_{\mathsf{c_c}} - 1 = 0$ Last iteration, nothing left to do. — PR-VS-Simp & VS-SemImp

$$\left\{\begin{array}{l} \mathsf{phase}((\mathtt{Forker})) * \exists v'_{\mathsf{c_p}}, O'_c.\ \mathsf{obs}(O'_c \uplus \{\!\!\{ \mathit{mut} \}\!\!\}) * \mathsf{locked}(\mathit{mut}, P_{\mathsf{m}}, \tfrac{1}{2}) * v_{\mathsf{c_c}} \neq 0 \\ * \big(\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) > 0 \ \rightarrow\ L_c{}^{\substack{\text{no:mutex}\\ \text{no:obs}}}(v'_{\mathsf{c_p}}, O'_c) * P_{\mathsf{m}} * \mathsf{itperm}((), 1)\big) \\ * \big(\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 0 \ \rightarrow\ L_c{}^{\substack{\text{no:mutex}\\ \text{no:obs}}}(v_{\mathsf{c_c}}, O_c) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * O'_c = O_c\big) \end{array}\right\}$$

$$= \left\{\begin{array}{l} \mathsf{phase}((\mathtt{Forker})) \\ * \exists v'_{\mathsf{c_c}}, O'_c.\ \mathsf{obs}(O'_c \uplus \{\!\!\{ \mathit{mut} \}\!\!\}) \\ * \mathsf{locked}(\mathit{mut}, P_{\mathsf{m}}, \tfrac{1}{2}) * \mathsf{PostIf}_c \end{array}\right\}$$

For definition of PostIf_c cf. Figure 28.

case: $v_{\mathsf{c_c}} - 1 > 0$ Must create signal for next iteration. — PR-VS-Simp & VS-SigInit

$$\left\{\begin{array}{l} \mathsf{obs}(\{\!\!\{ \mathit{id}^{v_{\mathsf{c_c}}-1}_{\text{pop}}, \mathit{mut} \}\!\!\}) * \mathop{\bigstar}_{i=1,\ldots,v_{\mathsf{c_c}}-1} \mathsf{uninitSig}(\mathit{id}^{i}_{\text{pop}}) \\ * \ \mathsf{uninitSig}(\mathit{id}^{v_{\mathsf{c_c}}-1}_{\text{pop}}) \quad \mathsf{signal}(s^{v_{\mathsf{c_c}}-1}_{\text{pop}}, \mathsf{False}) * \ldots \end{array}\right\}$$ — PR-VS-Simp & VS-SemImp

$\{\mathsf{phase}((\mathtt{Forker})) * \exists v'_{\mathsf{c_c}}, O'_c.\ \mathsf{obs}(O'_c \uplus \{\!\!\{ \mathit{mut} \}\!\!\}) * \mathsf{locked}(\mathit{mut}, P_{\mathsf{m}}, \tfrac{1}{2}) * \mathsf{PostIf}_c\}$

$\{\mathsf{phase}((\mathtt{Forker})) * \exists v'_{\mathsf{c_c}}, O'_c.\ \mathsf{obs}(O'_c \uplus \{\!\!\{ \mathit{mut} \}\!\!\}) * \mathsf{locked}(\mathit{mut}, P_{\mathsf{m}}, \tfrac{1}{2}) * \mathsf{PostIf}_c\}$

$\ldots$

Continued in Figure 28.

Figure 29: Verification example bounded FIFO, consumer thread's consumption step.

$\forall \ell_{\mathsf{fifo}_{10}}, \ell_{\mathsf{m}}, \ell_{\mathsf{c_c}}, \ell_{\mathsf{c_c}}, v_{\mathsf{c_c}}, O_c, v^{\mathsf{m}}_{\mathsf{fifo}_{10}}, v'_{\mathsf{c_c}}, O'_c.$

Continuation of Figure 28.

...

For definition of P_{m}, $P'_{\mathsf{m}}(v)$, $L_c^{\mathrm{locked}}(n, O)$, $L_c^{\overset{\mathsf{no:mutex}}{\mathsf{no:obs}}}(n, O)$, PostIf_c cf. Figures 21, 27 and 28.

$\{\mathsf{phase}((\mathtt{Forker})) * \mathsf{obs}(O'_c \uplus \{\!\{mut\}\!\}) * \mathsf{locked}(mut, P_{\mathsf{m}}, \frac{1}{2}) * \mathsf{PostIf}_c\}$

release m — PR-Release allows view shift to happen after mutex mut was released but before lock invariant P_{m} is consumed.

$\{\mathsf{phase}((\mathtt{Forker})) * \mathsf{obs}(O'_c \uplus \{\!\{mut\}\!\}) * \mathsf{locked}(mut, P_{\mathsf{m}}, \frac{1}{2}) * \mathsf{PostIf}_c\}$ — PR-VS-Simp & VS-SemImp

$\{(\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) > 0 \lor \mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 0) * \ldots\}$ — PR-VS-Simp & VS-Or

case: $\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) > 0$ — Consumption step already performed, nothing left to do.

$\{\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) > 0 * \mathsf{phase}((\mathtt{Forker})) * \mathsf{obs}(O'_c) * \mathsf{PostIf}_c\}$ — PR-VS-Simp & VS-SemImp

$\left\{\begin{array}{l}\mathsf{obs}(O'_c) * P_{\mathsf{m}} * \exists \tau_c^{\mathrm{anc}}, \delta_c.\ \tau_c^{\mathrm{anc}} \sqsubseteq (\mathtt{Forker}) * \mathsf{phase}((\mathtt{Forker})) \\ * L_c^{\overset{\mathsf{no:mutex}}{\mathsf{no:obs}}}(v'_{\mathsf{c_c}}, O'_c) * \mathsf{itperm}((), 1)\end{array}\right\}$

Define $\mathsf{PostReleaseVS}_c$ such that: $= \{\mathsf{obs}(O'_c) * P_{\mathsf{m}} * \mathsf{PostReleaseVS}_c\}$

case: $\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 0$ — No production step performed. Must wait to generate permission.

$\{\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 0 * \mathsf{phase}((\mathtt{Forker})) * \mathsf{obs}(O'_c) * \mathsf{PostIf}_c\}$ — PR-VS-Simp & VS-SemImp

$\left\{\begin{array}{l}\mathsf{PostIf}_c\ L_c^{\overset{\mathsf{no:mutex}}{\mathsf{no:obs}}}(v_{\mathsf{c_c}}, O_c) * P'_{\mathsf{m}}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) * O'_c = O_c = \{\!\{s_{\mathrm{pop}}^{v_{\mathsf{c_p}}}\}\!\} \\ * \mathsf{obs}(\{\!\{s_{\mathrm{pop}}^{v_{\mathsf{c_p}}}\}\!\}) * v_{\mathsf{c_c}} \neq 0 * \mathsf{phase}((\mathtt{Forker}))\end{array}\right\}$ — PR-Exists (2x) & PR-VS-Simp & VS-SemImp

$\forall v^{\mathsf{m}}_{\mathsf{c_c}}, v^{\mathsf{m}}_{\mathsf{c_p}}.$

$\left\{\begin{array}{l}\exists v^{\mathsf{m}}_{\mathsf{c_c}}, v^{\mathsf{m}}_{\mathsf{c_p}}.\ [\frac{1}{2}]\ell_{\mathsf{c_c}} \mapsto v_{\mathsf{c_c}} * [\frac{1}{2}]\ell_{\mathsf{c_c}} \mapsto v^{\mathsf{m}}_{\mathsf{c_c}} * v_{\mathsf{c_c}} = v^{\mathsf{m}}_{\mathsf{c_c}} * v_{\mathsf{c_c}} = v^{\mathsf{m}}_{\mathsf{c_p}} + 0 \\ (v^{\mathsf{m}}_{\mathsf{c_p}} > 0 \ \rightarrow\ \mathsf{signal}(s_{\mathrm{push}}^{v^{\mathsf{m}}_{\mathsf{c_p}}}, \mathsf{False}))\ \mathsf{signal}(s_{\mathrm{push}}^{v^{\mathsf{m}}_{\mathsf{c_p}}}, \mathsf{False}) \\ * \mathop{\huge *}_{i=1,\ldots,100} \mathsf{wperm}((), id^{i}_{\mathrm{push}}, 0) * \mathsf{obs}(\{\!\{s_{\mathrm{pop}}^{v_{\mathsf{c_c}}}\}\!\}) * \mathsf{phase}((\mathtt{Forker})) * \ldots\end{array}\right\}$ — PR-VS-Simp & VS-Wait

$\mathsf{lev}(s_{\mathrm{push}}^{v^{\mathsf{m}}_{\mathsf{c_p}}}) \ = \ L_{\mathrm{push}}^{v^{\mathsf{m}}_{\mathsf{c_p}}} \ = \ L_{\mathrm{push}}^{v_{\mathsf{c_c}}} \ = \ 101 - v_{\mathsf{c_c}} \ < \ 102 - v_{\mathsf{c_c}} \ = \ L_{\mathrm{pop}}^{v_{\mathsf{c_c}}} \ = \ \mathsf{lev}(s_{\mathrm{pop}}^{v_{\mathsf{c_c}}})$ — Justification for application of VS-Wait.

$\{\mathsf{itperm}((\mathtt{Forker}), 0) * \ldots\}$ — PR-VS-Simp & VS-SemImp

$\{\mathsf{obs}(O'_c) * P_{\mathsf{m}} * \mathsf{PostReleaseVS}_c\}$

$\left\{\begin{array}{l}\mathsf{phase}((\mathtt{Forker})) * \mathsf{obs}(O'_c) * \mathsf{PostIf}_c * (\mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) < 10 \lor \mathbf{size}(v^{\mathsf{m}}_{\mathsf{fifo}_{10}}) = 10) \\ * P_{\mathsf{m}} * \mathsf{PostReleaseVS}_c\end{array}\right\}$ — Conclusion of VS-Or application.

$\{\mathsf{obs}(O'_c) * P_{\mathsf{m}} * \mathsf{PostReleaseVS}_c\}$ — Lock invariant P_{m} consumed by PR-Release. — PR-VS-Simp & VS-SemImp

$\left\{\begin{array}{l}\exists \tau_c^{\mathrm{anc}}, \delta_c.\ \tau_c^{\mathrm{anc}} \sqsubseteq (\mathtt{Forker}) * \mathsf{phase}((\mathtt{Forker})) \\ * (v'_{\mathsf{c_c}} \neq 0 \ \rightarrow\ \exists v_{\mathsf{c_c}}, O_c.\ L_c(v_{\mathsf{c_c}}, O_c) * v_{\mathsf{c_c}} \neq 0 * \mathsf{itperm}(\tau_c^{\mathrm{anc}}, \delta_c)) * (v'_{\mathsf{c_c}} = 0 \ \rightarrow\ \mathsf{obs}(\emptyset))\end{array}\right\}$ — Reestablished loop invariant.

...

Continued in Figure 28.

Figure 30: Verification example bounded FIFO, consumer's wait step.