

Fine-grained concurrency verification without using observational equivalence

Bart Jacobs and Frank Piessens
Katholieke Universiteit Leuven,
Belgium

Structure of the Talk

- Programs; Executions; Example program
- Assertions; Truth; Proof Rules
- Proof of example: memory safety
- Proof of example: assert statement
- Modularized example
- Module spec
- Proof of modularized example

Programs

$c ::= \text{cons}(\bar{v}) \mid [\ell] \mid [\ell] := v \mid \text{let } x := c \text{ in } c(x) \mid \text{return } v$
 $\mid \text{atomic}(c) \mid \text{assert}(b) \mid \text{fork}(c)$

Executions

$$FullHeaps = \mathbb{Z}_0^+ \rightarrow_{\text{fin}} \mathbb{Z}$$

$$\kappa ::= \mathbf{done} \mid \mathbf{let } x := c \mathbf{ in } \kappa(x)$$

$$ThreadIds = \mathbb{Z}_0^+$$

$$Configs = FullHeaps \times (ThreadIds \rightarrow_{\text{fin}} \kappa)$$

Executions

S-LOOKUP

$$(t, \text{let } x := [\ell] \text{ in } \kappa(x)) \in T \quad (\ell, v) \in h$$

$$\langle h, T \rangle \xrightarrow{t:\ell, r} \langle h, T[t := \kappa(v)] \rangle$$

S-MUTATE

$$(t, [\ell] := v; \kappa) \in T \quad (\ell, v_0) \in h$$

$$\langle h, T \rangle \xrightarrow{t:\ell, r} \langle h[\ell := v], T[t := \kappa] \rangle$$

S-ATOMIC-LOOKUP

$$(t, \text{let } x := \text{atomic } [\ell] \text{ in } \kappa(x)) \in T \quad (\ell, v) \in h$$

$$\langle h, T \rangle \xrightarrow{t:\ell, a} \langle h, T[t := \kappa(v)] \rangle$$

S-ATOMIC-MUTATE

$$(t, \text{atomic}([\ell] := v); \kappa(x)) \in T \quad (\ell, v_0) \in h$$

$$\langle h, T \rangle \xrightarrow{t:\ell, a} \langle h[\ell := v], T[t := \kappa] \rangle$$

Example program

```
c := cons(0);  
fork (atomic ([c] := 1));  
atomic (x := [c]);  
atomic (y := [c]);  
assert (x ≤ y)
```

Assertions

$$\mathit{FracHeaps} = \mathbb{Z}_0^+ \multimap_{\text{fin}} ((0, 1] \times \mathbb{Z})$$

$$\mathit{FracGhostHeaps} = (\mathbb{Z}^+ \times \mathbb{Z}^+) \multimap_{\text{fin}} ((0, 1] \times \mathbb{Z})$$

$$\mathit{AtomicInvs} = \mathit{FracHeaps} \times \mathit{FracGhostHeaps} \rightarrow \mathbf{bool}$$

$$\mathit{AtomicPermSets} = \mathit{AtomicInvs} \rightarrow \mathbb{R}^+$$

Truth of an Assertion

$$h, g, A \models \mathbf{emp} \Leftrightarrow h = \emptyset \wedge g = \emptyset \wedge A = (\lambda I \bullet 0)$$

$$h, g, A \models \ell \xrightarrow{\pi} v \Leftrightarrow h = \{(\ell, (\pi, v))\} \wedge g = \emptyset \wedge A = (\lambda I \bullet 0)$$

$$h, g, A \models \ell_g \xrightarrow{\pi} v \Leftrightarrow h = \emptyset \wedge g = \{(\ell_g, (\pi, v))\} \wedge A = (\lambda I \bullet 0)$$

$$h, g, A \models \pi \boxed{I} \Leftrightarrow h = \emptyset \wedge g = \emptyset \wedge A = (\lambda I \bullet 0)[I := \pi]$$

$$s \models P * Q \Leftrightarrow \exists s_1, s_2 \bullet s = s_1 + s_2 \wedge (s_1 \models P) \wedge (s_2 \models Q)$$

$$s \models \phi \Leftrightarrow \phi \quad \text{for a pure formula } \phi$$

$$s \models P \odot Q \Leftrightarrow (s \models P) \odot (s \models Q) \quad \text{for } \odot \in \{\wedge, \vee, \Rightarrow\}$$

Ghost implication

$$\frac{\text{GI-IMPL} \quad P \Rightarrow Q}{P \Rightarrow Q}$$

$$\text{GI-GMUTATE} \quad l_g \mapsto v \Rightarrow l_g \mapsto v'$$

$$\text{GI-CREATEATOMIC} \quad I \Rightarrow \boxed{I}$$

$$\text{GI-DISPOSEATOMIC} \quad \boxed{I} \Rightarrow I$$

$$\frac{\text{GI-FRAME} \quad P \Rightarrow Q}{P * R \Rightarrow Q * R}$$

$$\frac{\text{GI-TRANS} \quad P \Rightarrow Q \quad Q \Rightarrow R}{P \Rightarrow R}$$

$$\frac{\text{C-GCONS} \quad \forall l \bullet \{(0, l) \mapsto v * P\} c \{Q\}}{\{P\} c \{Q\}}$$

Proof Rules

$$(\pi_1 + \pi_2) \boxed{S} \Rightarrow \pi_1 \boxed{S} * \pi_2 \boxed{S} \qquad \pi_1 \boxed{S} * \pi_2 \boxed{S} \Rightarrow (\pi_1 + \pi_2) \boxed{S}$$

$$\frac{\{S * P\} c \{S * Q\}}{\{\pi \boxed{S} * P\} \text{atomic}(c) \{\pi \boxed{S} * Q\}}$$

C-ATOMIC-NOOP

$$I * P \Rightarrow I * Q$$

$$\frac{}{\{\pi \boxed{I} * P\} \text{ATOMIC-NOOP} \{\pi \boxed{I} * Q\}}$$

C-FORK

$$\{P\} c \{\text{true}\}$$

$$\frac{}{\{P * R\} \text{fork}(c) \{R\}}$$

C-CONSEQ

$$P \Rightarrow P' \quad \{P'\} c \{Q\} \quad \forall v \bullet Q(v) \Rightarrow Q'(v)$$

$$\frac{}{\{P\} c \{Q'\}}$$

Proof of example program (1)

```
{emp}
c := cons(0);
{c ↦ 0}
{ $\exists v \bullet c \mapsto v$ }
fork (
  { $\frac{1}{2} \exists v \bullet c \mapsto v$ }
  atomic ([c] := 1)
  { $\frac{1}{2} \exists v \bullet c \mapsto v$ }
);
{ $\frac{1}{2} \exists v \bullet c \mapsto v$ }
atomic (x := [c]);
{ $\frac{1}{2} \exists v \bullet c \mapsto v$ }
atomic (y := [c]);
{ $\frac{1}{2} \exists v \bullet c \mapsto v$ }
// assert  $x \leq y$ 
```

```

{emp}
c := cons(0);
{c ↦ 0}
a := gcons(0);  (ghost cell allocation)
{c ↦ 0 * a ↦ 0}
{ $\exists v, b \bullet c \mapsto 0 * a \xrightarrow{1/2} b * b \leq v * v \leq 1$  *  $a \xrightarrow{1/2} 0$ }
fork (
  { $\frac{1}{2} \exists v, b \bullet c \mapsto v * a \xrightarrow{1/2} b * b \leq v * v \leq 1$ }
  atomic ([c] := 1)
  { $\frac{1}{2} \exists v, b \bullet c \mapsto v * a \xrightarrow{1/2} b * b \leq v * v \leq 1$ }
);
{ $\frac{1}{2} \exists v, b \bullet c \mapsto v * a \xrightarrow{1/2} b * b \leq v * v \leq 1$  *  $a \xrightarrow{1/2} 0$ }
atomic (
  x := [c]
  c ↦ v * a  $\xrightarrow{1/2}$  b * b ≤ v * v ≤ 1 * a  $\xrightarrow{1/2}$  0 ∧ x = v
  ⇔ merging of fractions
  c ↦ v * a ↦ 0 * 0 ≤ v * v ≤ 1 ∧ x = v
  ⇔ ghost cell update
  c ↦ v * a ↦ v * 0 ≤ v * v ≤ 1 ∧ x = v
  ⇔ splitting of fractions
  c ↦ v * a  $\xrightarrow{1/2}$  v * v ≤ v * v ≤ 1 * a  $\xrightarrow{1/2}$  v ∧ x = v
);
{ $\frac{1}{2} \exists v, b \bullet c \mapsto v * a \xrightarrow{1/2} b * b \leq v * v \leq 1$  *  $a \xrightarrow{1/2} x$ }
atomic (y := [c]);
{ $\frac{1}{2} \exists v, b \bullet c \mapsto v * a \xrightarrow{1/2} b * b \leq v * v \leq 1$  *  $a \xrightarrow{1/2} x \wedge x \leq y$ }
assert x ≤ y

```

Modularized example

$\text{createCell}(c) \stackrel{\Delta}{=} \text{skip}$
 $x := \text{get}(c) \stackrel{\Delta}{=} \text{atomic}(x := [c])$
 $\text{set}(c, x) \stackrel{\Delta}{=} \text{atomic}([c] := x)$
 $x := \text{compareAndSet}(c, x_0, x_1) \stackrel{\Delta}{=} \text{atomic}(x := [c]; \text{if } x = x_0 \text{ then } [c] := x_1)$
 $\text{disposeCell}(c) \stackrel{\Delta}{=} \text{skip}$

Figure 2. Implementation of the cell module

$c := \text{cons}(0);$	$x := \text{get}(c);$
$\text{createCell}(c);$	$y := \text{get}(c);$
$\text{fork}(\text{set}(c, 1));$	$\text{assert } x \leq y$

Figure 3. A client program of the cell module

Module spec

$$\begin{array}{c}
 \{c \mapsto v\} \text{ createCell}(c) \{ \text{cell}(c, v) \} \quad \frac{I * P \Leftrightarrow \exists v \bullet \text{cell}(c, v) * S(v) \quad \forall v \bullet \text{cell}(c, v) * S(v) \Rightarrow I * Q(v)}{\{ \pi \overline{I} \} * P \} x := \text{get}(c) \{ \pi \overline{I} \} * Q(x)} \\
 \\
 \frac{I * P \Leftrightarrow \text{cell}(c, -) * S \quad \text{cell}(c, v') * S \Rightarrow I * Q}{\{ \pi \overline{I} \} * P \} \text{ set}(c, v') \{ \pi \overline{I} \} * Q \} \\
 \\
 \frac{I * P \Leftrightarrow \exists v \bullet \text{cell}(c, v) * S(v) \quad \text{cell}(c, v_1) * S(v_0) \Rightarrow I * Q(v_0) \quad \forall v \bullet \text{cell}(c, v) * S(v) \wedge v \neq v_0 \Rightarrow I * Q(v)}{\{ \pi \overline{I} \} * P \} x := \text{compareAndSet}(c, v_0, v_1) \{ \pi \overline{I} \} * Q(x)} \\
 \\
 \overline{\{ \text{cell}(c, v) \} \text{ disposeCell}(c) \{ c \mapsto v \}}
 \end{array}$$

Figure 4. Specification of the cell module

```

{emp}
c := cons(0); createCell(c);
{cell(c, 0)}
a := gcons(0); (ghost cell allocation)
{cell(c, 0) * a ↦ 0}
{ $\exists v, b \bullet \text{cell}(c, v) * a \xrightarrow{1/2} b * b \leq v * v \leq 1 * a \xrightarrow{1/2} 0$ }
fork (
  { $\frac{1}{2} \exists v, b \bullet \text{cell}(c, v) * a \xrightarrow{1/2} b * b \leq v * v \leq 1$ }
  set(c, 1)
  { $\frac{1}{2} \exists v, b \bullet \text{cell}(c, v) * a \xrightarrow{1/2} b * b \leq v * v \leq 1$ }
);
{ $\frac{1}{2} \exists v, b \bullet \text{cell}(c, v) * a \xrightarrow{1/2} b * b \leq v * v \leq 1 * a \xrightarrow{1/2} 0$ }
cell(c, v) * a  $\xrightarrow{1/2}$  b * b ≤ v * v ≤ 1 * a  $\xrightarrow{1/2}$  0 ∧ x = v
⇔ merging of fractions
cell(c, v) * a ↦ 0 * 0 ≤ v * v ≤ 1 ∧ x = v
⇔ ghost cell update
cell(c, v) * a ↦ v * 0 ≤ v * v ≤ 1 ∧ x = v
⇔ splitting of fractions
cell(c, v) * a  $\xrightarrow{1/2}$  v * v ≤ v * v ≤ 1 * a  $\xrightarrow{1/2}$  v ∧ x = v
x := get(c);
{ $\frac{1}{2} \exists v, b \bullet \text{cell}(c, v) * a \xrightarrow{1/2} b * b \leq v * v \leq 1 * a \xrightarrow{1/2} x$ }
y := get(c);
{ $\frac{1}{2} \exists v, b \bullet \text{cell}(c, v) * a \xrightarrow{1/2} b * b \leq v * v \leq 1 * a \xrightarrow{1/2} x \wedge x \leq y$ }
assert x ≤ y

```

Queue example

```
q := create()  
enqueue(q, v)  
v := tryDequeue(q)  returns zero if queue is empty  
disposeQueue(q)
```

Queue spec

$$\overline{\{\text{emp}\} q := \text{create}() \{\text{queue}(q, \epsilon) * \text{consumer}(q)\}}$$

$$\frac{I * P \Leftrightarrow \exists \alpha \bullet \text{queue}(q, \alpha) * S(\alpha) \quad \forall \alpha \bullet \text{queue}(q, \alpha \cdot v) * S(\alpha) \Rightarrow I * Q}{\{\pi \boxed{I} * P\} \text{enqueue}(q, v) \{\pi \boxed{I} * Q\}}$$

$$\frac{I * P \Leftrightarrow \exists \alpha \bullet \text{queue}(q, \alpha) * S(\alpha) \quad \forall v, \alpha \bullet \text{queue}(q, \alpha) * S(v \cdot \alpha) \Rightarrow I * Q(v) \quad \text{queue}(q, \epsilon) * S(\epsilon) \Rightarrow I * Q(0)}{\{\text{consumer}(q) * \pi \boxed{I} * P\} x := \text{tryDequeue}(q) \{\text{consumer}(q) * \pi \boxed{I} * Q(x)\}}$$

$$\overline{\{\exists \alpha \bullet \text{queue}(q, \alpha) * \text{consumer}(q)\} \text{disposeQueue}(q) \{\text{emp}\}}$$

Soundness Proof

$$FullGhostHeaps = (\mathbb{Z}^+ \times \mathbb{Z}^+) \rightarrow_{\text{fin}} \mathbb{Z}$$

$$Multi(AtomicInvs) = AtomicInvs \rightarrow_{\text{fin}} \mathbb{Z}_0^+$$

$$\text{ids}(A) = \{(I, k) \mid \exists n \bullet (I, n) \in A \wedge k < n\}$$

modGhost

$$\text{gvalid}(\text{gcons}(v), Q) \triangleq \forall \ell \bullet (0, \ell) \notin \text{dom}(g) \Rightarrow Q(h, g[(0, \ell) := (1, v)], A)$$

$$\text{gvalid}([\ell_g] := v, Q) \triangleq \exists v_0 \bullet (\ell_g, (1, v_0)) \in g \wedge Q(h, g[\ell_g := (1, v)], A)$$

$$\text{gvalid}(\text{createAtomic}(I), Q) \triangleq \exists s_I, s' \bullet (h, g, A) = s_I + s' \wedge s_I \models I \wedge Q(s'[A := A[I := A(I) + 1]])$$

$$\text{gvalid}(\text{disposeAtomic}(I), Q) \triangleq A(I) \geq 1 \wedge \forall s_I, s' \bullet s' = (h, g, A) + s_I \wedge s_I \models I \Rightarrow Q(s'[A := A[I := A(I) - 1]])$$

$$\text{gsvalid}(\epsilon, Q) \triangleq Q(h, g, A)$$

$$\text{gsvalid}(c \cdot \bar{c}, Q) \triangleq \text{gvalid}(c, \text{gsvalid}(\bar{c}, Q))$$

$$\text{modGhost}(Q) \triangleq \exists \bar{c} \bullet \text{gsvalid}(\bar{c}, Q)$$

Lemma 1. *If $P \Rightarrow Q$, then $P \Rightarrow \text{modGhost}(Q)$.*

Validity

$$\text{valid}(\text{cons}(\bar{v}), Q) \triangleq \exists \bar{v}' \bullet \forall \ell, h', g' \bullet \text{fracAlloc } h, \ell, \bar{v}, h' \wedge \text{gAlloc } g, \ell, 0, \bar{v}', g' \Rightarrow Q(\ell)(h', g', A)$$

$$\text{valid}([\ell], Q) \triangleq \exists v \bullet (\ell, (1, v)) \in h \wedge Q(v)(h, g, A)$$

$$\text{valid}([\ell] := v, Q) \triangleq \exists v_0 \bullet (\ell, (1, v_0)) \in h \wedge Q(h[\ell := (1, v)], g, A)$$

$$\text{valid}(\text{let } x := c \text{ in } c'(x), Q) \triangleq \text{valid}(c, \text{modGhost}(\text{valid}(c'(x), Q)))$$

$$\text{valid}(\text{return } v, Q) \triangleq Q(v)(h, g, A)$$

$$\text{valid}(\text{atomic}(c), Q) \triangleq$$

$$\exists I, \pi \in \mathbb{R}_0^+ \bullet A(I) \geq \pi \wedge$$

$$\forall s_I, s' \bullet s_I \models I \wedge s' = s_I + (h, g, A[I := A(I) - \pi]) \Rightarrow$$

$$\text{modGhost}(\text{valid}(c, \text{modGhost}(\lambda h, g, A \bullet$$

$$\exists s_I, s' \bullet s_I \models I \wedge (h, g, A) = s_I + s' \wedge Q(s'[A := A[I := A(I) + \pi]]))$$

$$\text{valid}(\text{assert } b, Q) \triangleq b \wedge Q(h, g, A)$$

$$\text{valid}(\text{fork}(c), Q) \triangleq \exists s_F, s' \bullet (h, g, A) = s_F + s' \wedge \text{modGhost}(\text{valid}(c, (\lambda _ \bullet \text{true}))) (s_F) \wedge Q(s')$$

$$\text{valid}(\text{done}) \triangleq \text{true}$$

$$\text{valid}(\text{let } x := c \text{ in } \kappa(x)) \triangleq \text{valid}(c, \text{modGhost}(\text{valid}(\kappa(x))))$$

Validity versus triples

Lemma 2 (Correctness implies validity). *If $\{P\} c \{Q\}$, then $P * R \Rightarrow \text{modGhost}(\text{valid}(c, \text{modGhost}(Q * R)))$.*

Soundness Proof

A configuration $\langle h, T \rangle$ is *valid* if there exists a full ghost heap g , a multiset A of atomic invariants, a map s_T from the domain of T to bundles, and a map s_A from $\text{ids}(A)$ to bundles, such that

$$\forall \ell \notin \text{dom}(h), \ell' \bullet (\ell, \ell') \notin \text{dom}(g)$$

and

$$(h, g, A) = \sum_{t \in \text{dom}(T)} s_T(t) + \sum_{i \in \text{ids}(A)} s_A(i)$$

and

$$\forall (t, \kappa) \in T \bullet \text{valid}(\kappa)(s_T(t))$$

and

$$\forall (I, i) \in \text{ids}(A) \bullet s_A((I, i)) \models I$$

Soundness Proof

Lemma 4. *Execution steps preserve configuration validity.*

Lemma 5. *A valid configuration is not racy.*

Theorem 1 (Soundness). *If $\{\mathbf{true}\} c \{\mathbf{true}\}$, i.e. program c is correct, then c is data-race-free.*