Iris: Higher-Order Concurrent Separation Logic

Lecture 11: CAS and Spin Locks

Lars Birkedal

Aarhus University, Denmark

September 28, 2020

Overview

Earlier:

- Operational Semantics of $\lambda_{\text{ref,conc}}$
  - e, $(h, e) \rightsquigarrow (h, e')$, and $(h, \mathcal{E}) \rightarrow (h', \mathcal{E}')$
- Basic Logic of Resources
  - $l \hookrightarrow v$, $P * Q$, $P \twoheadrightarrow Q$, $\Gamma \mid P \vdash Q$
- Basic Separation Logic
  - $\{P\}\, e\, \{v.Q\}$: Prop, isList l xs, ADTs, foldr
- Later ($\triangleright$) and Persistent ($\square$) Modalities.
- Concurrency Intro and Invariants.
- Ghost State

Today:

- Specification and proof of lock module and client thereof
- Key Points:
  - Programming with and reasoning about uses of CAS.
  - Coarse and fine-grained concurrency.

Outline

- We consider a lock module.
- Note that our programming language $\lambda_{\mathrm{ref,conc}}$ does not include primitive locks – it is a feature of Iris that we can give expressive specifications to synchronization primitives programmed using cas.
- We then consider a client of the lock module, a concurrent bag implementation.
- Finally, we show how to verify an implementation of the lock module.

Lock Module Specification

$$
\begin{aligned}
&\exists\, \mathsf{isLock} : \mathit{Val} \to \mathsf{Prop} \to \mathsf{GhostName} \to \mathsf{Prop}.\\
&\exists\, \mathsf{locked} : \mathsf{GhostName} \to \mathsf{Prop}.\\
&\qquad \forall P, v, \gamma.\ \mathsf{isLock}(v, P, \gamma) \Rightarrow \Box\, \mathsf{isLock}(v, P, \gamma)\\
&\land \quad \forall \gamma.\ \mathsf{locked}(\gamma) * \mathsf{locked}(\gamma) \Rightarrow \mathsf{False}\\
&\land \quad \forall P.\ \{P\}\ \mathsf{newLock}()\ \{v.\exists \gamma.\ \mathsf{isLock}(v, P, \gamma)\}\\
&\land \quad \forall P, v, \gamma.\ \{\mathsf{isLock}(v, P, \gamma)\}\ \mathsf{acquire}\ v\ \{v.P * \mathsf{locked}(\gamma)\}\\
&\land \quad \forall P, v, \gamma.\ \{\mathsf{isLock}(v, P, \gamma) * P * \mathsf{locked}(\gamma)\}\ \mathsf{release}\ v\ \{\_.\mathsf{True}\}
\end{aligned}
$$

Remarks on lock spec

- isLock is persistent, hence duplicable, hence may can be shared among several threads, who will use the lock to coordinate access to shared memory.
- Quantification over P: the P predicates describes the resources the lock protects.
- Note the ownership transfer in acquire and release.
- The locked(γ) predicate (think of it as a token) is used to ensure that only the thread who has acquired the lock can release it – it is not duplicable since that would defeat its purpose.
- A *higher-order (3rd-order)* specification.
  - the order of the $\exists$ isLock and $\forall P$ quantifiers is not accidental: see iCap paper for an example, where this is crucial.
- Note that the specification only talks about resources (no mention of mutual exclusion and interleavings).

Client of Lock Module

- We now consider client of the lock module:
- A Concurrent (coarse-grained) bag.
- Note: we prove the client relative to the lock module spec, before considering an implementation of the lock module.
- MODULARITY !

Bag Specification

$$
\begin{array}{ll}
\multicolumn{2}{l}{\exists\, \mathrm{isBag} : (\mathit{Val} \to \mathrm{Prop}) \times \mathit{Val} \to \mathrm{Prop}.} \\
\multicolumn{2}{l}{\forall (\Phi : \mathit{Val} \to \mathrm{Prop}).} \\
 & \forall b.\ \mathrm{isBag}(\Phi, b) \Rightarrow \Box\, \mathrm{isBag}(\Phi, b) \\
\wedge & \{\mathrm{True}\}\ \mathrm{newBag}()\ \{b.\, \mathrm{isBag}(\Phi, b)\} \\
\wedge & \forall bu.\ \{\mathrm{isBag}(\Phi, b) * \Phi(u)\}\ \mathrm{insert}\, b\, u\ \{\_.\mathrm{True}\} \\
\wedge & \forall b.\ \{\mathrm{isBag}(\Phi, b)\}\ \mathrm{remove}\, b\ \{v.v = \mathrm{None} \vee \exists x.\ v = \mathrm{Some}\, x \wedge \Phi(x)\}
\end{array}
$$

- Note: for concurrent use (isBag is persistent).
- Hence we do not keep track of which elements the bag precisely contains.

Client of Bag Module

Trivial client:

$$
\begin{array}{l}
\{\text{True}\} \\
\textbf{let } b = \text{newBag}()\ \textbf{in} \\
\{\text{isBag}(\text{even}, b)\} \\
\{\text{isBag}(\text{even}, b) * \text{isBag}(\text{even}, b)\} \\
\begin{array}{lcl}
\{\text{isBag}(\text{even}, b)\} & & \{\text{isBag}(\text{even}, b)\} \\
\quad \text{insert}(b, 4) & \| & \quad \text{remove}(b) \\
\{\_.\text{True}\} & & \{\_.\text{True}\}
\end{array}
\end{array}
$$

See Hocap paper for a realistic client (a concurrent runner).

Bag Implementation

- We represent the bag by a pair consisting of
  - a reference to a (functional) list of values
  - a lock, used to protect access to the list of values

Bag Implementation

$$
\begin{aligned}
\textbf{let}\ \mathsf{newBag} &= \lambda\_.\ (\mathbf{ref}(\mathsf{None}), \mathsf{newLock}()) \\
\textbf{let}\ \mathsf{insert} &= \lambda x.\ \lambda v.\ \textbf{let}\ \ell = \pi_1\, x\ \textbf{in} \\
&\qquad \textbf{let}\ \mathsf{lock} = \pi_2\, x\ \textbf{in} \\
&\qquad \mathsf{acquire}\ \mathsf{lock}; \\
&\qquad \ell \leftarrow \mathsf{Some}(v, !\,\ell); \\
&\qquad \mathsf{release}\ \mathsf{lock} \\
\textbf{let}\ \mathsf{remove} &= \lambda x.\ \textbf{let}\ \ell = \pi_1\, x\ \textbf{in} \\
&\qquad \textbf{let}\ \mathsf{lock} = \pi_2\, x\ \textbf{in} \\
&\qquad \mathsf{acquire}\ \mathsf{lock}; \\
&\qquad \textbf{let}\ r = \textbf{match}\ !\,\ell\ \textbf{with} \\
&\qquad\qquad \mathsf{None} \Rightarrow \mathsf{None} \\
&\qquad\qquad \mid \mathsf{Some}\, p \Rightarrow \ell \leftarrow \pi_2\, p; \mathsf{Some}(\pi_1\, p) \\
&\qquad\qquad \textbf{end} \\
&\qquad \textbf{in}\ \ \mathsf{release}\ \mathsf{lock}; r
\end{aligned}
$$

Remark: limitations

- An implementation in which insert simply returns unit and in which remove simply returns None would also satisfy the specification.
- Note that such implementations would also be *safe*!
- But not as intended.
- Similar problem: if we forget to call release, then we can still verify the insert method.
- The problem is that Iris is *affine*, we can forget about resources.
- See Iron paper for a proposal of variant of Iris, which can address these issues.

Proof of Bag Spec

- The isBag predicate is defined as follows:

$$\mathsf{isBag}(\Phi, b) = \exists \ell v \gamma.\, b = (\ell, v) \wedge \mathsf{isLock}(v, \exists xs.\, \ell \hookrightarrow xs * \mathsf{bagList}(\Phi, xs), \gamma)$$

  where bagList is defined by guarded recursion as the unique predicate satisfying

$$\mathsf{bagList}(\Phi, xs) = xs = \mathsf{None} \vee \exists x.\, \exists r.\, xs = \mathsf{Some}(x, r) \wedge \Phi(x) * \triangleright(\mathsf{bagList}(\Phi, r)).$$

- Let $\Phi : \mathit{Val} \to \mathsf{Prop}$ be arbitrary.
- Note that $\mathsf{isBag}(\Phi, b)$ is persistent, for any b (why?)
- Showing the specs for newBag and insert is left as exercise.

Proof of remove

- TS

$$\{\mathsf{isBag}(\Phi, b)\}\ \mathsf{remove}\, b\, \{v.v = \mathsf{None} \vee \exists x.\ v = \mathsf{Some}\, x \wedge \Phi(x)\}$$

- By def'n of $\mathsf{isBag}(\Phi, b)$, using $\textsc{Ht-exist}$, and $\textsc{Ht-always}$ together with $\textsc{Ht-Eq}$, SFTS

  $\{\mathsf{isLock}(\mathsf{lock}, \exists xs.\, \ell \hookrightarrow xs * \mathsf{bagList}(\Phi, xs), \gamma)\}\ \mathsf{remove}(\ell, \mathsf{lock})\, \{u.u = \mathsf{None} \vee \exists x.\ u = \mathsf{Some}\, x \wedge \Phi(x)\}$

  for some ℓ, lock and γ.
- By $\textsc{Ht-beta}$ and $\textsc{Ht-let-det}$, SFTS

$$\{\mathsf{isLock}(\mathsf{lock}, \exists xs.\, \ell \hookrightarrow xs * \mathsf{bagList}(\Phi, xs), \gamma)\}\ e\, \{u.u = \mathsf{None} \vee \exists x.\ u = \mathsf{Some}\, x \wedge \Phi(x)\}$$

  where e is the program

$$\begin{array}{l}
\mathsf{acquire}\,\mathsf{lock};\\
\mathbf{let}\ r = \begin{array}[t]{l}\mathbf{match}\ !\,\ell\ \mathbf{with}\\
\quad\ \mathsf{None} \quad \Rightarrow \mathsf{None}\\
\quad \mid \mathsf{Some}\, p \Rightarrow \ell \leftarrow \pi_2\, p; \mathsf{Some}(\pi_1\, p)\\
\mathbf{end}\end{array}\\
\mathbf{in}\ \ \mathsf{release}\,\mathsf{lock};\ r
\end{array}$$

Proof of remove

- Using $\textsc{Ht-seq}$ and spec for acquire – SFTS

$$\{\mathsf{locked}(\gamma) * \exists xs.\, \ell \hookrightarrow xs * \mathsf{bagList}(\Phi, xs)\}\; e' \;\{u.u = \mathsf{None} \vee \exists x.\, u = \mathsf{Some}\, x \wedge \Phi(x)\}$$

  where e' is the part of program e after acquire.

- Use that $\exists$ and $\vee$ distribute over $*$, $\textsc{Ht-exist}$, and def'n of $\mathsf{bagList}(\Phi, xs)$, with $\textsc{Ht-disj}$ we consider two cases.
- The first case is

$$\{\mathsf{locked}(\gamma) * \ell \hookrightarrow xs * xs = \mathsf{None}\}\; e' \;\{u.u = \mathsf{None} \vee \exists x.\, u = \mathsf{Some}\, x \wedge \Phi(x)\}$$

  Left as exercise!

- In the second case, after structural rules SFTS:

$$\{\mathsf{locked}(\gamma) * \ell \hookrightarrow \mathsf{Some}(x, r) * \Phi(x) * \triangleright \mathsf{bagList}(\Phi, r)\}\; e' \;\{u.\exists x.\, u = \mathsf{Some}\, x \wedge \Phi(x)\}.$$

Proof of remove

- We use HT-LET-DET. For the first premise we show

$$\{\mathsf{locked}(\gamma) * \ell \hookrightarrow \mathsf{Some}(x, r) * \Phi(x) * \triangleright \mathsf{bagList}(\Phi, r)\}$$

$$\begin{array}{l} \textbf{match } !\,\ell \textbf{ with} \\ \quad \mathsf{None} \quad \Rightarrow \mathsf{None} \\ \mid \mathsf{Some}\, p \Rightarrow \ell \leftarrow \pi_2\, p; \mathsf{Some}(\pi_1\, p) \\ \textbf{end} \end{array}$$

$$\{u.u = \mathsf{Some}\, x \wedge \ell \hookrightarrow r * \Phi(x) * \mathsf{locked}(\gamma) * \mathsf{bagList}(\Phi, r)\}$$

  (note the omission of $\triangleright$ on bagList in the postcondition)
- See notes.

Proof of remove

- For the second premise of the rule $\textsc{Ht-let-det}$, SFTS

$$\begin{array}{l}\{\ell \hookrightarrow r * \Phi(x) * \mathsf{locked}(\gamma) * \mathsf{bagList}(\Phi, r)\} \\ \quad \mathsf{release}\,\mathsf{lock}; \mathsf{Some}\,x \\ \{u.\exists x.\ u = \mathsf{Some}\,x \wedge \Phi(x)\}\end{array}$$

- We use sequencing rule together with the release spec to give away the resources $\ell \hookrightarrow r$, $\mathsf{locked}(\gamma)$ and $\mathsf{bagList}(\Phi, r)$ back to the lock.
- We are left with proving

$$\begin{array}{l}\{\Phi(x)\} \\ \quad \mathsf{Some}\,x \\ \{u.\exists x.\ u = \mathsf{Some}\,x \wedge \Phi(x)\}\end{array}$$

which is immediate.

Spin lock implementation

- We now return to lock module and show that a spin lock implementation satisfies the lock module spec.
- The lock is implemented by a boolean flag:

$$\mathsf{let}\ \mathsf{newLock}() = \mathsf{ref}(\mathsf{false})$$

$$\mathsf{let}\ \mathsf{acquire}\ l = \mathsf{if}\ \mathsf{cas}(l, \mathsf{false}, \mathsf{true})\ \mathsf{then}\ ()\ \mathsf{else}\ \mathsf{acquire}\ l$$

$$\mathsf{let}\ \mathsf{release}\ l = l \leftarrow \mathsf{false}$$

Spin lock implementation

- We now return to lock module and show that a spin lock implementation satisfies the lock module spec.
- The lock is implemented by a boolean flag:

$$\textsf{let}\ \mathsf{newLock}() = \mathsf{ref}(\mathsf{false})$$
$$\textsf{let}\ \mathsf{acquire}\ l = \mathsf{if}\ \mathsf{cas}(l, \mathsf{false}, \mathsf{true})\ \mathsf{then}\ ()\ \mathsf{else}\ \mathsf{acquire}\ l$$
$$\textsf{let}\ \mathsf{release}\ l = l \leftarrow \mathsf{false}$$

- We now proceed to prove that this implementation meets the specification of the spin lock module. We do it in quite a lot of detail, since this example is very instructive! First, we need a proof rule for CAS.

Proof rule for CAS

Basic proof rule for CAS:

Ht-CAS

$$\frac{}{\{\triangleright \ell \hookrightarrow v\}\ \mathsf{cas}(\ell, v_1, v_2)\ \{u.(u = \mathsf{true} * v = v_1 * \ell \hookrightarrow v_2) \vee (u = \mathsf{false} * v \neq v_1 * \ell \hookrightarrow v)\}}$$

Often the following derived rules are easier to use.

Ht-CAS-succ

$$\frac{}{\{\triangleright \ell \hookrightarrow v_1\}\ \mathsf{cas}(\ell, v_1, v_2)\ \{u.u = \mathsf{true} * \ell \hookrightarrow v_2\}}$$

Ht-CAS-fail

$$\frac{}{\{\triangleright \ell \hookrightarrow v * \triangleright(v \neq v_1)\}\ \mathsf{cas}(\ell, v_1, v_2)\ \{u.u = \mathsf{false} * \ell \hookrightarrow v\}}$$

Proof of Spin Lock Spec

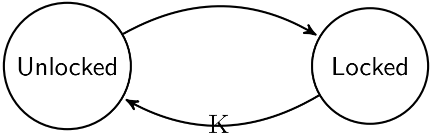
- Need to record whether lock is in a locked or unlocked state.
- Use resource algebra $\{\varepsilon, \bot, \mathrm{K}\}$, with $\varepsilon \cdot x = x \cdot \varepsilon = x$ and otherwise $x \cdot y = \bot$.
- isLock predicate:

$$\begin{aligned}\mathsf{isLock}(v, P, \gamma) &= \exists \ell \in \mathit{Loc}, \iota \in \mathsf{InvName}.\ v = \ell \wedge \boxed{I(\ell, P, \gamma)}^{\iota} \\ \mathsf{locked}(\gamma) &= \boxed{\mathrm{K}}^{\gamma}\end{aligned}$$

where the invariant is

$$I(\ell, P, \gamma) = \ell \hookrightarrow \mathsf{false} * \boxed{\mathrm{K}}^{\gamma} * P \vee \ell \hookrightarrow \mathsf{true}.$$

- Intuition:



Remarks on the Invariant

- Notice that the invariant involves a predicate variable P.
- Hence the predicate in the invariant is *not* timeless.
- This is an example where it is important the invariant opening rule includes a $\triangleright$ modality.
- Observe that the predicate variable P appears in the invariant, because we want to prove the very *general and modular* specification for the lock, which involves quantification over any predicate P.

Proof of spin lock spec

- There are now five proof obligations, one for each of the conjuncts in the specification.
- The first says that isLock(v, P, γ) is persistent: clear because invariants and equality are persistent, and $\wedge$, $\exists$ preserves persistency.
- The second says that locked(γ) *is not* duplicable. This follows as $\mathrm{K} \cdot \mathrm{K} = \bot$ by definition of the resource algebra: $\boxed{\mathrm{K}}^{\gamma} * \boxed{\mathrm{K}}^{\gamma} \vdash \boxed{\mathrm{K} \cdot \mathrm{K}}^{\gamma}$ by $\textsc{Own-op}$ which yields False by $\textsc{Own-valid}$.
- Now consider each operation.

Proof of newLock

- TS

$$\{P\}\ \mathsf{newLock}()\ \{v.\exists\gamma.\ \mathsf{isLock}(v, P, \gamma)\}$$

- By Ht-beta, SFTS

$$\{P\}\ \mathsf{ref}(\mathsf{false})\ \{v.\exists\gamma.\ \mathsf{isLock}(v, P, \gamma)\}$$

- We allocate new ghost state by Ghost-alloc, use consequence Ht-exist. We are left with proving

$$\{\mathsf{locked}(\gamma) * P\}\ \mathsf{ref}(\mathsf{false})\ \{v.\, \mathsf{isLock}(v, P, \gamma)\}$$

  for some γ.
- Exercise!

Proof of acquire

- It is recursive, so we use derived rule for recursive functions, *i.e.*, we assume

$$\forall v, P, \gamma. \{\triangleright \mathsf{isLock}(v, P, \gamma)\}\ \mathsf{acquire}\, v\ \{v.P * \mathsf{locked}(\gamma)\} \tag{1}$$

  and then show

$$\{\mathsf{isLock}(v, P, \gamma)\}\ \mathsf{if}\, \mathsf{cas}(v, \mathsf{false}, \mathsf{true})\ \mathsf{then}\ ()\ \mathsf{else}\ \mathsf{acquire}(v)\ \{v.P * \mathsf{locked}(\gamma)\}.$$

- By isLock def'n, v is a location ℓ governed by an invariant, which we can move into the context as follows:

$$\boxed{I(\ell, P, \gamma)}^{\iota} \vdash \{\mathsf{True}\}\ \mathsf{if}\, \mathsf{cas}(\ell, \mathsf{false}, \mathsf{true})\ \mathsf{then}\ ()\ \mathsf{else}\ \mathsf{acquire}(\ell)\ \{v.P * \mathsf{locked}(\gamma)\}$$

- We next use HT-BIND – and start by showing

$$\boxed{I(\ell, P, \gamma)}^{\iota} \vdash \{\mathsf{True}\}\ \mathsf{cas}(\ell, \mathsf{false}, \mathsf{true})\ \{u.(u = \mathsf{true} * P * \mathsf{locked}(\gamma)) \lor (u = \mathsf{false})\}.$$

Proof of acquire

- As cas is atomic, we open the invariant to get at ℓ, using HT-INV-OPEN. So SFTS

$$\boxed{I(\ell, P, \gamma)}^{\iota} \vdash \{\triangleright I(\ell, P, \gamma)\}\,\mathsf{cas}(\ell, \mathsf{false}, \mathsf{true})\,\{u.((u = \mathsf{true} * P * \mathsf{locked}(\gamma)) \vee (u = \mathsf{false})) * \triangleright I(\ell, P, \gamma)\}.$$

- We proceed by cases on the invariant (using HT-DISJ). In the first case, TS

$$\boxed{I(\ell, P, \gamma)}^{\iota} \vdash$$

$$\{\triangleright(\ell \hookrightarrow \mathsf{false} * \mathsf{locked}\,\gamma * P)\}\,\mathsf{cas}(\ell, \mathsf{false}, \mathsf{true})\,\{u.(u = \mathsf{true} * P * \mathsf{locked}(\gamma) \vee (u = \mathsf{false})) * \triangleright I(\ell, P, \gamma)\}.$$

- We use HT-CAS-SUCC and HT-FRAME to get $u = \mathsf{true} * P * \mathsf{locked}(\gamma) * \ell \hookrightarrow \mathsf{true}$, which satisfies the disjunctions in the postcondition (also the one hidden in $I(\ell, P, \gamma)$).
- In the second case, TS

$$\boxed{I(\ell, P, \gamma)}^{\iota} \vdash$$

$$\{\triangleright(\ell \hookrightarrow \mathsf{true})\}\,\mathsf{cas}(\ell, \mathsf{false}, \mathsf{true})\,\{u.((u = \mathsf{true} * P * \mathsf{locked}(\gamma) \vee (u = \mathsf{false})) * \triangleright I(\ell, P, \gamma)\}.$$

- We use consequence and HT-CAS-FAIL, which yields postcondition $u = \mathsf{false} * \ell \hookrightarrow \mathsf{true}$.

Proof of acquire

- We now proceed with our use of HT-BIND, the evaluation of the if, and thus SFTS

$$\boxed{I(\ell, P, \gamma)}^{\iota} \vdash \{u = \mathsf{true} * P * \mathsf{locked}(\gamma) \vee u = \mathsf{false}\}\ \mathsf{if}\ u\ \mathsf{then}\ ()\ \mathsf{else}\ \mathsf{acquire}\ \ell\ \{\_.P * \mathsf{locked}(\gamma)\}$$

- We consider the two cases in the precondition, using HT-DISJ.
- We use HT-IF-TRUE and HT-IF-FALSE in the first and second case respectively, so SFTS

$$\boxed{I(\ell, P, \gamma)}^{\iota} \vdash \{P * \mathsf{locked}(\gamma)\}\ ()\ \{\_.P * \mathsf{locked}(\gamma)\}$$
$$\boxed{I(\ell, P, \gamma)}^{\iota} \vdash \{\mathsf{True}\}\ \mathsf{acquire}\ \ell\ \{\_.P * \mathsf{locked}(\gamma)\}$$

- The first follows by the rule for the unit expressions, the second by our induction hypothesis (1). Done!

Proof of release

- TS

$$\{\mathsf{isLock}(v, P, \gamma) * P * \mathsf{locked}(\gamma)\}\ \mathsf{release}\, v\, \{\_.\mathsf{True}\}$$

- By isLock(v, P, γ) def'n $v = \ell$ for some ℓ, and by $\textsc{Ht-beta}$ SFTS

$$\left\{\boxed{I(\ell, P, \gamma)}^{\iota} * P * \mathsf{locked}(\gamma)\right\} \ell \leftarrow \mathsf{false}\, \{\_.\mathsf{True}\}$$

- Invariants are persistent, hence we move it into context, and then use $\textsc{Ht-inv-open}$. SFTS

$$\boxed{I(\ell, P, \gamma)}^{\iota} \vdash \{\triangleright I(\ell, P, \gamma) * P * \mathsf{locked}(\gamma)\}\, \ell \leftarrow \mathsf{false}\, \{\_.\triangleright I(\ell, P, \gamma)\}$$

Proof of release

- We consider two cases, based on the disjunction in $I(\ell, P, \gamma)$ in the precondition.
- The first case is

$$\boxed{I(\ell, P, \gamma)}^{\iota} \vdash \{\triangleright(\ell \hookrightarrow \mathsf{false} * \mathsf{locked}(\gamma) * P) * P * \mathsf{locked}(\gamma)\}\, \ell \leftarrow \mathsf{false}\, \{\_.\, \triangleright I(\ell, P, \gamma)\}$$

  which is inconsistent as $\mathsf{locked}(\gamma) * \mathsf{locked}(\gamma) \vdash \mathsf{False}$. Hence done by Ht-later-false.
- In the second case we need to prove

$$\boxed{I(\ell, P, \gamma)}^{\iota} \vdash \{\triangleright(\ell \hookrightarrow \mathsf{true}) * P * \mathsf{locked}(\gamma)\}\, \ell \leftarrow \mathsf{false}\, \{\_.\, \triangleright I(\ell, P, \gamma)\}$$

- In the postcondition we show the first disjunct; by consequence SFTS

$$\boxed{I(\ell, P, \gamma)}^{\iota} \vdash \{\triangleright(\ell \hookrightarrow \mathsf{true}) * \triangleright(P * \mathsf{locked}(\gamma))\}\, \ell \leftarrow \mathsf{false}\, \{\_.\, \triangleright(\ell \hookrightarrow \mathsf{false}) * \triangleright(\mathsf{locked}(\gamma) * P)\}$$

  which holds by the frame rule and Ht-store.