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 $\lambda_{\text{ref,conc}}$**
  - $e, (h, e) \leadsto (h, e')$, and $(h, E) \rightarrow (h', E')$

- **Basic Logic of Resources**
  - $l \leftrightarrow v, P \ast Q, P \rightarrow Q, \Gamma | P \vdash Q$

- **Basic Separation Logic**
  - $\{P\} e \{v.Q\} : \text{Prop, isList } l \ xs, \text{ADTs, foldr}$

- **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.
We consider a lock module.

Note that our programming language $\lambda_{\text{ref,conc}}$ does not include primitive locks – it is a feature of Iris that we can give expressive specifications to synchronization primitives programmed using $\text{cas}$.

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

\( \exists \text{isLock} : \text{Val} \rightarrow \text{Prop} \rightarrow \text{GhostName} \rightarrow \text{Prop}. \)

\( \exists \text{locked} : \text{GhostName} \rightarrow \text{Prop}. \)

\( \forall P, v, \gamma. \ \text{isLock}(v, P, \gamma) \Rightarrow \Box \text{isLock}(v, P, \gamma) \)

\( \land \forall \gamma. \ \text{locked}(\gamma) * \text{locked}(\gamma) \Rightarrow \text{False} \)

\( \land \forall P. \ \{P\} \ \text{newLock}() \ \{v. \exists \gamma. \ \text{isLock}(v, P, \gamma)\} \)

\( \land \forall P, v, \gamma. \ \{\text{isLock}(v, P, \gamma)\} \ \text{acquire} \ v \ \{v. P * \text{locked}(\gamma)\} \)

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

∃ isBag : (Val → Prop) × Val → Prop.
∀ (Φ : Val → Prop).
  ∀ b. isBag(Φ, b) ⇒ □ isBag(Φ, b)
∧ {True} newBag() {b. isBag(Φ, b)}
∧ ∀ bu. {isBag(Φ, b) * Φ(u)} insert b u {...True}
∧ ∀ b. {isBag(Φ, b)} remove b {v. v = None ∨ ∃ x. v = Some x ∧ Φ(x)}

▶ 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:

\[
\{ \text{True} \}
\]

\[
\text{let } b = \text{newBag()} \text{ in }
\]

\[
\{ \text{isBag(even, } b \text{)} \}
\]

\[
\{ \text{isBag(even, } b \text{)} \ast \text{isBag(even, } b \text{)} \}
\]

\[
\{ \text{isBag(even, } b \text{)} \} \quad \{ \text{isBag(even, } b \text{)} \}
\]

\[
\text{insert}(b, 4) \ || \ \text{remove}(b)
\]

\[
\{ _.\text{True} \} \quad \{ _.\text{True} \}
\]

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

```ocaml
let newBag = \_. (ref(None), newLock())
let insert = \x. \v. let \ell = \pi_1 x in
    let lock = \pi_2 x in
    acquire lock;
    \ell ← Some(v, !\ell);
    release lock
let remove = \x. let \ell = \pi_1 x in
    let lock = \pi_2 x in
    acquire lock;
    let r = match !\ell with
        None  ⇒ None
        | Some p ⇒ \ell ← \pi_2 p; Some(\pi_1 p)
    end
    in  release lock; r
```
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:

\[
isBag(\Phi, b) = \exists \ell \nu \gamma. \ b = (\ell, \nu) \land \text{isLock}(\nu, \exists xs. \ell \hookrightarrow xs \ast \text{bagList}(\Phi, xs), \gamma)
\]

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

\[
\text{bagList}(\Phi, xs) = xs = \text{None} \lor \exists x. \exists r. \ xs = \text{Some}(x, r) \land \Phi(x) \ast \triangleright (\text{bagList}(\Phi, r)).
\]

- Let \( \Phi : Val \to \text{Prop} \) be arbitrary.

- Note that isBag(\( \Phi, b \)) is persistent, for any \( b \) (why?)

- Showing the specs for newBag and insert is left as exercise.
Proof of remove

- **TS**

\[
\{ \text{isBag}(\Phi, b) \} \ \text{remove} \ b \ \{ v. v = \text{None} \lor \exists x. v = \text{Some} \ x \land \Phi(x) \}
\]

- By def'n of isBag(\Phi, b), using HT-EXIST, and HT-ALWAYS together with HT-EQ, SFTS

\[
\{ \text{isLock}(\text{lock}, \exists xs. \ell \leftrightarrow xs \ast \text{bagList}(\Phi, xs), \gamma) \} \ \text{remove}(\ell, \text{lock}) \ \{ u. u = \text{None} \lor \exists x. u = \text{Some} \ x \land \Phi(x) \}
\]

for some \( \ell \), lock and \( \gamma \).

- By HT-BETA and HT-LET-DET, SFTS

\[
\{ \text{isLock}(\text{lock}, \exists xs. \ell \leftrightarrow xs \ast \text{bagList}(\Phi, xs), \gamma) \} \ e \ \{ u. u = \text{None} \lor \exists x. u = \text{Some} \ x \land \Phi(x) \}
\]

where \( e \) is the program

\[
\begin{align*}
\text{acquire lock; } \\
\text{let } r = \text{match } \ell \text{ with } \\
\quad \text{None } \Rightarrow \text{None} \\
\quad | \text{Some } p \Rightarrow \ell \leftarrow \pi_2 p; \text{Some}(\pi_1 p) \\
\text{end} \\
\text{in } \text{release lock; } r
\end{align*}
\]
Proof of remove

- Using $\text{HT-SEQ}$ and spec for acquire – SFTS

\[
\{\text{locked}(\gamma) \ast \exists xs. \ell \leftrightarrow xs \ast \text{bagList}(\Phi, xs)\} e' \{u.u = \text{None} \lor \exists x. u = \text{Some} x \land \Phi(x)\}
\]

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

- Use that $\exists$ and $\lor$ distribute over $\ast$, $\text{HT-EXIST}$, and def'n of bagList$(\Phi, xs)$, with $\text{HT-DISJ}$ we consider two cases.

- The first case is

\[
\{\text{locked}(\gamma) \ast \ell \leftrightarrow xs \ast xs = \text{None}\} e' \{u.u = \text{None} \lor \exists x. u = \text{Some} x \land \Phi(x)\}
\]

Left as exercise!

- In the second case, after structural rules SFTS:

\[
\{\text{locked}(\gamma) \ast \ell \leftrightarrow \text{Some}(x, r) \ast \Phi(x) \ast \triangleright \text{bagList}(\Phi, r)\} e' \{u.\exists x. u = \text{Some} x \land \Phi(x)\}.
\]
Proof of remove

- We use \texttt{HT-LET-DET}. For the first premise we show

\[
\{\text{locked}(\gamma) \ast \ell \rightarrow \text{Some}(x, r) \ast \Phi(x) \ast \triangleright \text{bagList}(\Phi, r)\}
\]

\[
\text{match } !\ell \text{ with }
\]

\[
\text{None } \Rightarrow \text{None}
\]

\[
| \text{Some } p \Rightarrow \ell \leftarrow \pi_2 p; \text{Some}(\pi_1 p)
\]

\[
\text{end}
\]

\[
\{u.u = \text{Some } x \land \ell \rightarrow r \ast \Phi(x) \ast \text{locked}(\gamma) \ast \text{bagList}(\Phi, r)\}
\]

(note the omission of \triangleright on \text{bagList} in the postcondition)

- See notes.
Proof of remove

- For the second premise of the rule $H_{T\text{-LET}\text{-DET}}$, SFTS
  \[
  \{ \ell \hookrightarrow r \ast \Phi(x) \ast \text{locked(} \gamma \text{)} \ast \text{bagList}(\Phi, r) \} \\
  \text{release lock; Some } x \\
  \{ u. \exists x. u = \text{Some } x \land \Phi(x) \}\]

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

- We are left with proving
  \[
  \{ \Phi(x) \} \\
  \text{Some } x \\
  \{ u. \exists x. u = \text{Some } x \land \Phi(x) \}\]

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:

```ml
let newLock() = ref(false)
let acquire l = if cas(l, false, true) then () else acquire l
let release l = l ← 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:

  ```ocaml
  let newLock() = ref(false)
  let acquire l = if cas(l, false, true) then () else acquire l
  let release l = l ← 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:

$$\text{HT-CAS}$$

$$\{\triangleright \ell \leftrightarrow v\} \text{cas}(\ell, v_1, v_2) \{u.(u = \text{true} \ast v = v_1 \ast \ell \leftrightarrow v_2) \lor (u = \text{false} \ast v \neq v_1 \ast \ell \leftrightarrow v)\}$$

Often the following derived rules are easier to use.

$$\text{HT-CAS-succ}$$

$$\{\triangleright \ell \leftrightarrow v_1\} \text{cas}(\ell, v_1, v_2) \{u.u = \text{true} \ast \ell \leftrightarrow v_2\}$$

$$\text{HT-CAS-fail}$$

$$\{\triangleright \ell \leftrightarrow v \ast \triangleright(v \neq v_1)\} \text{cas}(\ell, v_1, v_2) \{u.u = \text{false} \ast \ell \leftrightarrow v\}$$
Proof of Spin Lock Spec

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

\[
\text{isLock}(v, P, \gamma) = \exists \ell \in \text{Loc}, \iota \in \text{InvName}. \; v = \ell \land I(\ell, P, \gamma)^{\iota} \\
\text{locked}(\gamma) = K^{\gamma}
\]

where the invariant is

\[
I(\ell, P, \gamma) = \ell \rightarrow \text{false} \ast K^{\gamma} \ast P \lor \ell \rightarrow \text{true}.
\]

▶ 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 $\text{isLock}(v, P, \gamma)$ is persistent: clear because invariants and equality are persistent, and $\land$, $\exists$ preserves persistency.
- The second says that $\text{locked}(\gamma)$ is not duplicable. This follows as $K \cdot K = \bot$ by definition of the resource algebra: $\{K\}^\gamma \cdot \{K\}^\gamma \vdash \{K \cdot K\}^\gamma$ by OWN-OP which yields False by OWN-VALID.
- Now consider each operation.
Proof of newLock

- TS

\[ \{ P \} \text{newLock()} \{ v.\exists \gamma. \text{isLock}(v, P, \gamma) \} \]

- By HT-BETA, SFTS

\[ \{ P \} \text{ref(false)} \{ v.\exists \gamma. \text{isLock}(v, P, \gamma) \} \]

- We allocate new ghost state by GHOST-ALLOC, use consequence HT-EXIST. We are left with proving

\[ \{ \text{locked}(\gamma) \ast P \} \text{ref(false)} \{ v. \text{isLock}(v, P, \gamma) \} \]

for some \( \gamma \).

- Exercise!
Proof of acquire

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

\[ \forall v, P, \gamma. \{ \triangleright \text{isLock}(v, P, \gamma) \} \text{ acquire } v \{ v.P \ast \text{locked}(\gamma) \} \]  

(1)

and then show

\[ \{ \text{isLock}(v, P, \gamma) \} \text{ if cas}(v, \text{false}, \text{true}) \text{ then } () \text{ else } \text{acquire}(v) \{ v.P \ast \text{locked}(\gamma) \} \].

- By isLock def’n, \( v \) is a location \( \ell \) governed by an invariant, which we can move into the context as follows:

\[ I(\ell, P, \gamma) \vdash \{ \text{True} \} \text{ if cas}(\ell, \text{false}, \text{true}) \text{ then } () \text{ else } \text{acquire}(\ell) \{ v.P \ast \text{locked}(\gamma) \} \]

- We next use Ht-bind – and start by showing

\[ I(\ell, P, \gamma) \vdash \{ \text{True} \} \text{ cas}(\ell, \text{false}, \text{true}) \{ u.(u = \text{true} \ast P \ast \text{locked}(\gamma)) \lor (u = \text{false}) \}. \]
Proof of acquire

- As \text{cas} is atomic, we open the invariant to get at \( \ell \), using \text{HT-INV-OPEN}. So SFTS
  
  \[ I(\ell, P, \gamma) |\{\triangleright I(\ell, P, \gamma)\} \triangleright \text{cas}(\ell, \text{false}, \text{true}) \{u.(u = \text{true} \ast P \ast \text{locked(\gamma)}) \lor (u = \text{false}) \} \ast \triangleright I(\ell, P, \gamma) \].

- We proceed by cases on the invariant (using \text{HT-DISJ}). In the first case, TS
  
  \[ I(\ell, P, \gamma) |\{\triangleright(\ell \leftrightarrow \text{false} \ast \text{locked(\gamma)} \ast P)\} \triangleright \text{cas}(\ell, \text{false}, \text{true}) \{u.(u = \text{true} \ast P \ast \text{locked(\gamma)}) \lor (u = \text{false}) \} \ast \triangleright I(\ell, P, \gamma) \].

- We use \text{HT-CAS-SUCC} and \text{HT-FRAME} to get
  
  \( u = \text{true} \ast P \ast \text{locked(\gamma)} \ast \ell \leftrightarrow \text{true} \), which satisfies the disjunctions in the postcondition (also the one hidden in \( I(\ell, P, \gamma) \)).

- In the second case, TS
  
  \[ I(\ell, P, \gamma) |\{\triangleright(\ell \leftrightarrow \text{true})\} \triangleright \text{cas}(\ell, \text{false}, \text{true}) \{u.(u = \text{true} \ast P \ast \text{locked(\gamma)}) \lor (u = \text{false}) \} \ast \triangleright I(\ell, P, \gamma) \].

- We use consequence and \text{HT-CAS-FAIL}, which yields postcondition
  
  \( u = \text{false} \ast \ell \leftrightarrow \text{true} \).
Proof of acquire

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

$$I(\ell, P, \gamma) \vdash \{u = \text{true} \ast P \ast \text{locked}(\gamma) \lor u = \text{false}\} \text{ if } u \text{ then } () \text{ else } \text{acquire } \ell \{\ldots P \ast \text{locked}(\gamma)\}$$

- 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

$$I(\ell, P, \gamma) \vdash \{P \ast \text{locked}(\gamma)\} () \{\ldots P \ast \text{locked}(\gamma)\}$$

$$I(\ell, P, \gamma) \vdash \{\text{True}\} \text{ acquire } \ell \{\ldots P \ast \text{locked}(\gamma)\}$$

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

- TS

\(\{ \text{isLock}(v, P, \gamma) \ast P \ast \text{locked}(\gamma) \}\ \text{release}\ v \ \{\text{True}\}\)

- By isLock\((v, P, \gamma)\) def'n \(v = \ell\) for some \(\ell\), and by \(H_T\)-\(\text{BETA}\) SFTS

\(\{I(\ell, P, \gamma)^\ell \ast P \ast \text{locked}(\gamma)\} \ell \leftarrow \text{false} \ \{\text{True}\}\)

- Invariants are persistent, hence we move it into context, and then use \(H_T\)-\(\text{INV-OPEN}\). SFTS

\(I(\ell, P, \gamma)^\ell \vdash \{\triangleright I(\ell, P, \gamma) \ast P \ast \text{locked}(\gamma)\} \ell \leftarrow \text{false} \ \{\triangleright I(\ell, P, \gamma)\}\)
Proof of release

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

- The first case is

\[
\begin{align*}
I(\ell, P, \gamma)^{\ell} \vdash \{ \triangleright (\ell \leftrightarrow \text{false} \ast \text{locked(}\gamma) \ast P) \ast P \ast \text{locked(}\gamma) \} \ell \leftarrow \text{false} \{ \ldots \triangleright I(\ell, P, \gamma) \}
\end{align*}
\]

which is inconsistent as \( \text{locked(}\gamma) \ast \text{locked(}\gamma) \vdash \text{False} \). Hence done by \( \text{HT-LATER-FALSE} \).

- In the second case we need to prove

\[
\begin{align*}
I(\ell, P, \gamma)^{\ell} \vdash \{ \triangleright (\ell \leftrightarrow \text{true}) \ast P \ast \text{locked(}\gamma) \} \ell \leftarrow \text{false} \{ \ldots \triangleright I(\ell, P, \gamma) \}
\end{align*}
\]

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

\[
\begin{align*}
I(\ell, P, \gamma)^{\ell} \vdash \{ \triangleright (\ell \leftrightarrow \text{true}) \ast \triangleright (P \ast \text{locked(}\gamma)) \} \ell \leftarrow \text{false} \{ \ldots \triangleright (\ell \leftrightarrow \text{false}) \ast \triangleright (\text{locked(}\gamma) \ast P) \}
\end{align*}
\]

which holds by the frame rule and \( \text{HT-STORE} \).