Iris: Higher-Order Concurrent Separation Logic

Lecture 11: CAS and Spin Locks

Lars Birkedal

Aarhus University, Denmark

November 28, 2017
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
  - $l \leftrightarrow v, P * Q, P \rightarrow* Q, \Gamma \vdash P \mid Q$
- Basic Separation Logic
  - $\{P\} e \{v.Q\} : \text{Prop, isList } l \ x s, \text{ADTs, foldr}$
- Later (⊿) and Persistent (□) 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 \texttt{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.
∃ \text{isLock} : \text{Val} \rightarrow \text{Prop} \rightarrow \text{GhostName} \rightarrow \text{Prop}.

∃ \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) \ast \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 \ast \text{locked}(\gamma)\}

\land \forall P, v, \gamma. \{\text{isLock}(v, P, \gamma) \ast P \ast \text{locked}(\gamma)\} \text{ release } v \{\ldots. \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).
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.
Trivial client:

```plaintext
{True}
let b = newBag() in
{isBag(even, b)}
{isBag(even, b) * isBag(even, b)}
{isBag(even, b)} {isBag(even, b)}
    insert(b, 4) || remove(b)
{_.True} {_.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 \leftarrow 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 \Rightarrow None
        | Some p \Rightarrow \ell \leftarrow \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 a linear variant of Iris, which can address these issues.
The isBag predicate is defined as follows:

\[
\text{isBag}(\Phi, b) = \exists \ell v \gamma. b = (\ell, v) \land \text{isLock}(v, \exists xs. \ell \leftrightarrow xs \star \text{bagList}(\Phi, xs), \gamma)
\]

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

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

Let \( \Phi : Val \rightarrow Prop \) 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 \ \{\nu. \nu = \text{None} \lor \exists x. \nu = \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, \text{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

acquire lock;

let \ r = \ \text{match} \ !\ell \ \text{with}

None \Rightarrow \text{None}
| \text{Some} \ p \Rightarrow \ell \leftarrow \pi_2 \ p; \text{Some}(\pi_1 \ p)

end

in \ \text{release} \ \text{lock}; \ r
Proof of remove

- Using \( HT-SEQ \) and spec for acquire – SFTS

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

where \( e' \) is the part of program \( e \) after acquire.

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

- The first case is

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

Left as exercise!

- In the second case, after structural rules SFTS:

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

- We use $H_{T\text{-LET-DET}}$. For the first premise we show

$$\{\text{locked}(γ) \land \ell \leftrightarrow \text{Some}(x, r) \ast \Phi(x) \ast \triangleright \text{bagList}(Φ, r)\}$$

match !ℓ with

  None  ⇒ None

| Some p ⇒ ℓ ← $\pi_2$ p; Some($\pi_1$ p)

end

$$\{u.u = \text{Some} x \land \ell \leftrightarrow r \ast \Phi(x) \ast \text{locked}(γ) \ast \text{bagList}(Φ, r)\}$$

(note the omission of $\triangleright$ on bagList in the postcondition)

- See notes.
Proof of remove

- For the second premise of the rule $\text{HT-LET-DET, SFTS}$

\[
\{ \ell \leftrightarrow r \ast \Phi(x) \ast \text{locked}(\gamma) \ast \text{bagList}(\Phi, r) \}\ \\
\text{release lock; Some } x \\
\{ u.\exists x. u = \text{Some } x \land \Phi(x) \}\]

- We use sequencing rule together with the release spec to give away the resources $\ell \leftrightarrow r$, 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:

  ```
  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
define 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:

\[
\text{HT-CAS} \\
\{ \triangleright \ell \leftrightarrow v \} \ \text{cas}(\ell, v_1, v_2) \{ u. (u = \text{true} \ast v = v_1 * \ell \leftrightarrow v_2) \lor (u = \text{false} \ast v \neq v_1 * \ell \leftrightarrow v) \}
\]

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 \( \{ \varepsilon, \perp, K \} \), with \( \varepsilon \cdot x = x \cdot \varepsilon = x \) and otherwise \( x \cdot y = \perp \).
- 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

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

- Intuition:
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 $\wedge, \exists$ 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: $\overline{K}^\gamma \cdot \overline{K}^\gamma \vdash \overline{K} \cdot \overline{K}^\gamma$ by OWN-OP which yields False by OWN-VALID.
- Now consider each operation.
Proof newLock

- TS

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

- By \texttt{Ht-beta}, SFTS

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

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

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

for some \( \gamma \).

- Exercise!
Proof of acquire

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

  \[ \forall v, P, \gamma. \{ \triangleright \text{isLock}(v, P, \gamma) \} \ \text{acquire} \ v \ \{ v.P * \text{locked}(\gamma) \} \tag{1} \]

  and then show

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

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

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

- We next use \( \text{HT-BIND} \) – and start by showing

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

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

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

- 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
  \[ I(\ell, P, \gamma) \vdash \]
  \[ \{ [\mathcal{G} (\ell \leftrightarrow \text{true})] \text{cas}(\ell, \text{false, true}) \{ u.((u = \text{true} \ast P \ast \text{locked}(\gamma)) \lor (u = \text{false})) \ast \mathcal{G} I(\ell, P, \gamma) \}. \]

- 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 \(\text{HT-BIND}\), the evaluation of the \(\text{if}\), and thus \(\text{SFTS}\)

\[
\mathcal{I}(\ell, P, \gamma)^i \vdash \{ u = \text{true} \} \ast P \ast \text{locked}(\gamma) \lor \{ u = \text{false} \} \text{if } u \text{ then } () \text{ else } \text{acquire } \ell \{ \_ . P \ast \text{locked}(\gamma) \}
\]

- We consider the two cases in the precondition, using \(\text{HT-DISJ}\).

- We use \(\text{HT-IF-TRUE}\) and \(\text{HT-IF-FALSE}\) in the first and second case respectively, so \(\text{SFTS}\)

\[
\begin{align*}
\mathcal{I}(\ell, P, \gamma)^i & \vdash \{ P \ast \text{locked}(\gamma) \} () \{ \_ . P \ast \text{locked}(\gamma) \} \\
\mathcal{I}(\ell, P, \gamma)^i & \vdash \{ \text{True} \} \text{acquire } \ell \{ \_ . P \ast \text{locked}(\gamma) \}
\end{align*}
\]

- 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 **HT-BETA SFTS**

  \[
  \big\{\llbracket I(\ell, P, \gamma) \rrbracket^\ell \ast P \ast \text{locked}(\gamma)\big\} \ell \leftarrow \text{false} \{\text{..True}\}
  \]

- Invariants are persistent, hence we move it into context, and then use **HT-INV-OPEN. SFTS**

  \[
  \llbracket I(\ell, P, \gamma) \rrbracket^\ell \vdash \{\triangleright I(\ell, P, \gamma) \ast P \ast \text{locked}(\gamma)\} \ell \leftarrow \text{false} \{\text{..} \triangleright I(\ell, P, \gamma)\}
  \]


Proof of release

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

\[
\mathcal{I}(\ell, P, \gamma) \vdash \{ \triangleright (\ell \leftrightarrow \text{false} \ast \text{locked}(\gamma) \ast P) \ast P \ast \text{locked}(\gamma) \} \ell \leftarrow \text{false} \{ \ldots \triangleright I(\ell, P, \gamma) \}
\]

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

\[
\mathcal{I}(\ell, P, \gamma) \vdash \{ \triangleright (\ell \leftrightarrow \text{true}) \ast P \ast \text{locked}(\gamma) \} \ell \leftarrow \text{false} \{ \ldots \triangleright I(\ell, P, \gamma) \}
\]

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

\[
\mathcal{I}(\ell, P, \gamma) \triangleright \{ \triangleright (\ell \leftrightarrow \text{true}) \ast \triangleright (P \ast \text{locked}(\gamma)) \} \ell \leftarrow \text{false} \{ \ldots \triangleright (\ell \leftrightarrow \text{false}) \ast \triangleright (\text{locked}(\gamma) \ast P) \}
\]

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