Iris: Higher-Order Concurrent Separation Logic

Lecture 14: Extended Case Study: stacks with helping

Lars Birkedal

Aarhus University, Denmark

December 7, 2020

Overview

Earlier:

- Operational Semantics of $\lambda_{\mathrm{ref,conc}}$
  - e, $(h, e) \rightsquigarrow (h, e')$, and $(h, \mathcal{E}) \rightarrow (h', \mathcal{E}')$
- Basic Logic of Resources
  - $l \hookrightarrow v$, $P * Q$, $P \mathrel{-\!\!*} Q$, $\Gamma \mid P \vdash Q$
- Basic Separation Logic
  - $\{P\}\, e\, \{v.Q\}$: Prop, isList l xs, ADTs, foldr
- Later ($\triangleright$) and Persistent ($\square$) Modalities.
- Concurrency Intro, Invariants and Ghost State
- CAS, Spin Locks, Concurrent Counter Modules.
- Weakest preconditions and the fancy update modality

Today:

- Extended Case Study
- Key Points:
  - You can now verify fairly advanced programs!

Concurrent Stacks with Helping

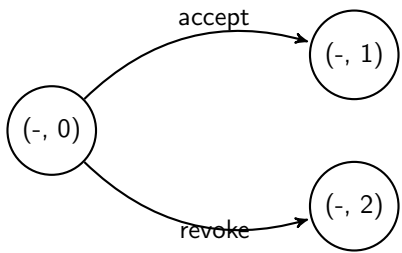
Goal for today:

- Implement, specify and verify a concurrent stack
- Implementation will use *helping*:
  - programming pattern where a *side-channel* is used to reduce contention on the data structure
  - suppose there are two threads, one which wishes to push (the *pusher*), and one which wishes to pop (the *popper*)
  - then they can communicate directly, on a side-channel, and *help* each other complete their respective operations, without touching the core data structure used for the stack
- The pusher will *offer* the value it wishes to push on a side-channel, and a concurrent popper may accept the offer.
- If no popper is around, then the offer may be revoked, and the value pushed onto the actual stack.
- Likewise, if the popper sees no offer, then it will try to pop from the actual stack.

Offers

- An offer can be *created* with an initial value.
- An offer can be accepted, marking the offer as taken and returning the underlying value.
- Once created, an offer can be revoked which will prevent anyone from accepting the offer and return the underlying value to the thread.

An offer is represented as a pair, consisting of the offered value and a reference to an int (0, 1, 2). STS:



Offer Implementation

$$
\begin{aligned}
\text{mk\_offer} &\triangleq \lambda v.(v, \text{ref}(0)) \\
\text{revoke\_offer} &\triangleq \lambda v.\ \text{let } u = \pi_1\ v \text{ in} \\
&\qquad\qquad \text{let } s = \pi_2\ v \text{ in} \\
&\qquad\qquad \text{if cas}(s, 0, 2) \text{ then Some } u \text{ else None} \\
\text{accept\_offer} &\triangleq \lambda v.\ \text{let } u = \pi_1\ v \text{ in} \\
&\qquad\qquad \text{let } s = \pi_2\ v \text{ in} \\
&\qquad\qquad \text{if cas}(s, 0, 1) \text{ then Some } u \text{ else None}
\end{aligned}
$$

Mailboxes for Offers

- The pattern of offering something, immediately revoking it, and returning the value if the revoke was successful is common: we encapsulate it in an abstraction called a *mailbox*.
- A mailbox is built around an underlying cell containing an offer. It provides two functions which, respectively, briefly put a new offer out and check for such an offer.

$$\text{mailbox} \triangleq \lambda\_.\ \text{let } r = \text{ref}(\text{None}) \text{ in}$$

$$\left(\begin{array}{l} \left(\begin{array}{ll} \lambda v. & \text{let off} = \text{mk\_offer } v \text{ in} \\ & r \leftarrow \text{Some off}; \text{revoke\_offer off} \end{array} \right), \\ \\ \left(\begin{array}{ll} \lambda\_. & \text{let offopt} = !\, r \text{ in} \\ & \text{match offopt with} \\ & \quad \text{None} \quad \Rightarrow \text{None} \\ & \mid \text{Some } x \Rightarrow \text{accept\_offer } x \\ & \text{end} \end{array} \right) \end{array} \right)$$

We will call the first part of the tuple the put method, and the second one the get method.

Stack Implementation

$$
\begin{aligned}
&\mathsf{stack} \triangleq \lambda\_. \\
&\qquad \mathsf{let}\ \mathsf{mb} = \mathsf{mailbox}()\ \mathsf{in} \\
&\qquad \mathsf{let}\ \mathsf{put} = \pi_1\ \mathsf{mb}\ \mathsf{in} \\
&\qquad \mathsf{let}\ \mathsf{get} = \pi_2\ \mathsf{mb}\ \mathsf{in} \\
&\qquad \mathsf{let}\ r = \mathsf{ref}(\mathsf{None})\ \mathsf{in}
\end{aligned}
$$

```
(rec pop() = match get() with
               None    ⇒ match ! r with
                           None      ⇒ None
                         | Some hd ⇒ let h = π1 hd in
                                     let t = π2 hd in
                                     if cas(r, Some hd, t)
                                      then Some h
                                      else pop()
                         end
             | Some x ⇒ Some x
             end,
```

$$
\begin{array}{l}
\mathsf{rec}\ \mathsf{push}() = \mathsf{match}\ \ \mathsf{put}()\ \mathsf{with} \\
\qquad\qquad\quad \mathsf{None} \quad\ \Rightarrow () \\
\qquad\qquad\quad \mid \mathsf{Some}\, n \Rightarrow \mathsf{let}\, r' = !\, r\ \mathsf{in} \\
\qquad\qquad\qquad\qquad\qquad\ \ \mathsf{let}\, r'' = \mathsf{Some}(n, r')\ \mathsf{in} \\
\qquad\qquad\qquad\qquad\qquad\ \ \mathsf{if}\, \mathsf{cas}(r, r', r'')\ \mathsf{then}\, () \\
\qquad\qquad\qquad\qquad\qquad\ \ \mathsf{else}\ \mathsf{push}() \\
\qquad\qquad\quad \mathsf{end})
\end{array}
$$

Stack Specification (bag-like spec)

$$\forall \Phi.\{\mathsf{True}\}\ \mathsf{stack}()\left\{p.\exists\, \mathsf{pop}\ \mathsf{push}\ .\ \begin{array}{l} p = (\mathsf{pop}, \mathsf{push}) * \\ \{\mathsf{True}\}\ \mathsf{pop}()\,\{v.v = \mathsf{None} \vee \exists v'.\ v = \mathsf{Some}\ v' * \Phi(v')\} * \\ \forall v.\ \{\Phi(v)\}\ \mathsf{push}\ v\,\{u.u = () * \mathsf{True}\} \end{array}\right\}$$

Outline of Specs and Proofs

Modularity:

- specs and proofs for
  - offers
  - mailboxes
  - stacks

Verifying Offers

- Encode the transition system using ghost state.
- Only the thread which has made an offer may revoke the offer, so need token to control that. Use the exclusive monoid on unit will as token.
- Transition system represented by:

$$\mathsf{stages}_\gamma(v, \ell) \triangleq (\Phi(v) * \ell \hookrightarrow 0) \lor \ell \hookrightarrow 1 \lor (\ell \hookrightarrow 2 * \boxed{\mathsf{ex}(())}^{\gamma})$$

- Representation predicate for offers:

$$\mathsf{is\_offer}_\gamma(v) \triangleq \exists v', \ell.\, v = (v', \ell) * \exists \iota.\, \boxed{\mathsf{stages}_\gamma(v', \ell)}^{\iota}$$

- (each ghost variable γ corresponds to an offer)

Specifying Offers

- mk_offer creates an offer and the right to revoke it:

$$\forall v.\, \{\Phi(v)\}\ \mathsf{mk\_offer}(v)\, \{u.\ \exists \gamma.\, \boxed{\mathsf{ex}(())}^{\gamma} * \mathsf{is\_offer}_{\gamma}(u)\}$$

- revoke_offer needs the token:

$$\forall \gamma, v.\, \{\mathsf{is\_offer}_{\gamma}(v) * \boxed{\mathsf{ex}(())}^{\gamma}\}\ \mathsf{revoke\_offer}(v)\, \{u.u = \mathsf{None} \vee \exists v'.\, u = \mathsf{Some}(v') * \Phi(v')\}$$

- accept_offer

$$\forall \gamma, v.\, \{\mathsf{is\_offer}_{\gamma}(v)\}\ \mathsf{accept\_offer}(v)\, \{u.u = \mathsf{None} \vee \exists v'.\, u = \mathsf{Some}(v') * \Phi(v')\}$$

Verifying Mailboxes

- Specifying put and get operations in the same style as before:

$$\{\mathsf{True}\}\ \mathsf{mailbox}()\left\{u.\ \begin{array}{l}\exists\, \mathsf{put}\, \mathsf{get}\,.\\ u = (\mathsf{put}, \mathsf{get}) *\\ \forall v.\, \{\Phi(v)\}\ \mathsf{put}(v)\, \{w.w = \mathsf{None} \vee \exists v'.\, w = \mathsf{Some}(v') * \Phi(v')\} *\\ \{\mathsf{True}\}\ \mathsf{get}()\, \{w.w = \mathsf{None} \vee \exists v'.\, w = \mathsf{Some}(v') * \Phi(v')\}\end{array}\right\} \tag{1}$$

- Representation predicate (invariant governing the shared mutable cell that contains potential offers):

$$\mathsf{is\_mailbox}(\ell) \triangleq \ell \hookrightarrow \mathsf{None} \vee \exists v' \gamma.\, \ell \hookrightarrow \mathsf{Some}(v') * \mathsf{is\_offer}_\gamma(v')$$

Verifying Stacks

- Recall desired spec:

$$\forall \Phi. \{\mathsf{True}\}\ \mathsf{stack}()\left\{p. \exists\, \mathsf{pop}\ \mathsf{push}\,.\ \begin{array}{l} p = (\mathsf{pop}, \mathsf{push}) * \\ \{\mathsf{True}\}\ \mathsf{pop}()\,\{v. v = \mathsf{None} \vee \exists v'.\ v = \mathsf{Some}\ v' * \Phi(v')\} \\ \forall v. \{\Phi(v)\}\ \mathsf{push}\ v\,\{u. u = () * \mathsf{True}\} \end{array}\right. \tag{2}$$

- Representation predicate:

$$\mathsf{is\_stack}(v) \triangleq v = \mathsf{None} \vee \exists h, t.\ v = \mathsf{Some}(h, t) * \Phi(h) * \triangleright\, \mathsf{is\_stack}(t)$$

$$\mathsf{stack\_inv}(v) \triangleq \exists v'.\ \ell \hookrightarrow v' * \mathsf{is\_stack}(v')$$