Iris: Higher-Order Concurrent Separation Logic

Lecture 14: Extended Case Study: stacks with helping

Lars Birkedal

Aarhus University, Denmark

December 5, 2017

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

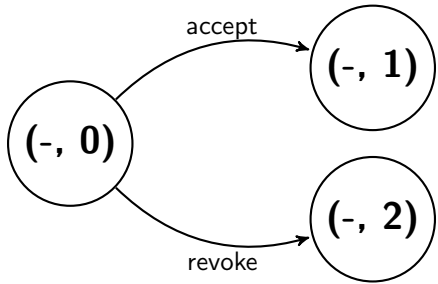
Goal for today:

- 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 around 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.

```
mk_offer = fun v -> (v, ref 0)
revoke_offer =
  fun v ->
    if cas (snd v) 0 2
    then Some (fst v)
    else None
accept_offer =
  fun v ->
    if cas (snd v) 0 1
    then Some (fst v)
    else None
```



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.

```
mailbox = fun () ->
  let r = ref None in
  (rec put v ->
     let off = mk_offer v in
     r := Some off;
     revoke_offer off,
   rec get n ->
     let offopt = !r in
     match offopt with
       None -> None
     | Some x -> accept_offer x
     end)
```

Stack Implementation

```
stack = fun () ->
  let mailbox = mailbox () in
  let put = fst mailbox in
  let get = snd mailbox in
  let r = ref None in
  (rec pop n ->
     match get () with
       None ->
       (match !r with
          None -> None
        | Some hd ⇒
          if cas r (Some hd) (snd hd)
          then Some (fst hd)
          else pop n
        end)
     | Some x -> Some x
     end,
```

Stack Implementation

```
rec push n ->
  match put n with
    None -> ()
  | Some n ->
    let r' = !r in
    let r'' = Some (n, r') in
    if cas r r' r''
    then ()
    else push n
  end)
```

Stack Specification

- Idea: bag-like spec:

$$P(v) \mathrel{-\!\!*} \mathsf{wp}_{\mathcal{E}}\, \mathsf{push}(v)\,\{\mathsf{True}\}$$
$$\mathsf{wp}_{\mathcal{E}}\, \mathsf{pop}()\quad \{v.v = \mathsf{None} \vee \exists v'.\, v = \mathsf{Some}(v') * P(v')\}$$

- Formally,
  - return push and pop functions, so need to use nested triples / weakest preconditions
  - we give the spec in the same style as proof rules for wps, with arbitrary postcondition (eases using the specification)

$$\begin{aligned}
&\forall \Phi.\\
&\quad (\forall f_1 f_2.\, \mathsf{wp}\ f_1()\,\{v.v = \mathsf{None} \vee \exists v'.\, v = \mathsf{Some}(v') * P(v')\}\\
&\qquad \mathrel{-\!\!*} \forall v.\, P(v) \mathrel{-\!\!*} \mathsf{wp}\ f_2(v)\,\{\mathsf{True}\}\\
&\qquad \mathrel{-\!\!*} \Phi(f_1, f_2))\\
&\quad \mathrel{-\!\!*} \mathsf{wp}\ \mathsf{stack}()\,\{\Phi\}
\end{aligned}$$

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:

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

- 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:

$$\forall v.\, P(v) \mathrel{-\!\!*} \mathsf{wp}\ \mathsf{mk\_offer}(v)\,\{v.\ \exists \gamma.\, \boxed{\mathsf{ex}(())}^{\gamma} * \mathsf{is\_offer}_{\gamma}(v)\}$$

- `revoke_offer` needs the token:

$$\forall \gamma, v.\, \mathsf{is\_offer}_{\gamma}(v) * \boxed{\mathsf{ex}(())}^{\gamma} \mathrel{-\!\!*} \mathsf{wp}\ \mathsf{revoke\_offer}(v)\,\{v.\quad v = \mathsf{None} \vee \exists v'.\, v = \mathsf{Some}(v) * P(v')\ \}$$

- `accept_offer`

$$\forall \gamma, v.\, \mathsf{is\_offer}_{\gamma}(v) \mathrel{-\!\!*} \mathsf{wp}\ \mathsf{accept\_offer}(v)\,\{v.\ v = \mathsf{None} \vee \exists v'.\, v = \mathsf{Some}(v) * P(v')\}$$

Verifying Mailboxes

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

$$
\begin{aligned}
&\forall \Phi. \\
&\quad (\forall f_1 f_2. (\forall v.\, P(v) \mathrel{-\!\!*} \mathsf{wp}\ f_1(v)\,\{v.v = \mathsf{None} \vee \exists v'.\, v = \mathsf{Some}(v') * P(v')\}) \\
&\qquad \mathrel{-\!\!*} \mathsf{wp}\ f_2()\,\{v.v = \mathsf{None} \vee \exists v'.\, v = \mathsf{Some}(v') * P(v')\} \\
&\qquad \mathrel{-\!\!*} \Phi(f_1, f_2)) \\
&\quad \mathrel{-\!\!*} \mathsf{wp}\ \mathsf{mailbox}()\,\{\Phi\}
\end{aligned}
\tag{1}
$$

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

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

Verifying Stacks

- Recall desired spec:

$$\begin{aligned}&\forall \Phi.\\ &\quad (\forall f_1 f_2.\ \mathsf{wp}\ f_1()\,\{v.v = \mathsf{None} \lor \exists v'.\ v = \mathsf{Some}(v) * P(v)\}\\ &\qquad -\!\!*\forall v.\ P(v) -\!\!* \mathsf{wp}\ f_2(v)\,\{\mathsf{True}\}\\ &\qquad -\!\!*\Phi(f_1, f_2))\\ &\quad -\!\!*\ \mathsf{wp}\ \mathsf{stack}()\,\{\Phi\}\end{aligned}$$

- Representation predicate:

$$\mathsf{is\_stack}(v) \triangleq \mu R.\ v = \mathsf{None} \lor \exists h, t.\ v = \mathsf{Some}(h, t) * P(h) * \triangleright R(t)$$

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