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 $\lambda_{\text{ref,conc}}$
  - $e, (h, e) \leadsto (h, e')$, and $(h, \mathcal{E}) \rightarrow (h', \mathcal{E}')$
- Basic Logic of Resources
  - $l \hookrightarrow v, P \ast Q, P \ast Q, \Gamma \vdash P$
- Basic Separation Logic
  - $\{P\} e \{v.Q\} : \text{Prop, isList \ l \ xs, ADTs, foldr}$
- 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.

\[
\text{mk\_offer} = \text{fun v -> (v, ref 0)}
\]
\[
\text{revoke\_offer} =
\text{fun v ->}
\text{  if cas (snd v) 0 2 then Some (fst v) else None}
\]
\[
\text{accept\_offer} =
\text{fun v ->}
\text{  if cas (snd v) 0 1 then Some (fst v) else None}
\]
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

```ocaml
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:

\[
P(v) \implies \text{wp}_E \text{ push}(v) \{\text{True}\}
\]

\[
\text{wp}_E \text{ pop()} \quad \{v.v = \text{None} \lor \exists v'. v = \text{Some}(v') \implies P(v')\}
\]

- 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)

\[
\forall \Phi.
\quad (\forall f_1 f_2. \text{wp } f_1() \{v.v = \text{None} \lor \exists v'. v = \text{Some}(v') \implies P(v')\})
\quad \implies \forall v. P(v) \implies \text{wp } f_2(v) \{\text{True}\}
\quad \implies \Phi(f_1, f_2))
\quad \implies \text{wp stack()} \{\Phi\}
\]
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:

\[
\text{stages}_\gamma(v, \ell) \triangleq (P(v) \ast \ell \hookrightarrow 0) \lor \ell \hookrightarrow 1 \lor (\ell \hookrightarrow 2 \ast \text{ex}())
\]

- Representation predicate for offers:

\[
\text{is_offer}_\gamma(v) \triangleq \exists v', \ell. \, v = (v', \ell) \ast \ell. \, \text{stages}_\gamma(v', \ell)\]

- (each ghost variable \( \gamma \) corresponds to an offer)
Specifying Offers

- \texttt{mk\_offer} creates an offer and the right to revoke it:

\[ \forall v. \ P(v) \rightarrow wp \ mk\_offer(v) \{ v. \ \exists \gamma. \ \text{is\_offer\_}\gamma(v) \} \]

- \texttt{revoke\_offer} needs the token:

\[ \forall \gamma, v. \ \text{is\_offer\_}\gamma(v) \rightarrow wp \ \text{revoke\_offer}(v) \{ v. \ v = \text{None} \lor \exists v'. \ v = \text{Some}(v) \rightarrow P(v') \} \]

- \texttt{accept\_offer}

\[ \forall \gamma, v. \ \text{is\_offer\_}\gamma(v) \rightarrow wp \ \text{accept\_offer}(v) \{ v. \ v = \text{None} \lor \exists v'. \ v = \text{Some}(v) \rightarrow P(v') \} \]
Verifying Mailboxes

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

\[ \forall \Phi. \]
\[ (\forall f_1 f_2. (\forall v. P(v) \implies \text{wp } f_1(v) \{ v.v = \text{None} \lor \exists v'. v = \text{Some}(v') \ast P(v') \}) ) \]
\[ \implies \text{wp } f_2() \{ v.v = \text{None} \lor \exists v'. v = \text{Some}(v') \ast P(v') \} \]
\[ \implies \Phi(f_1, f_2) \]
\[ \implies \text{wp mailbox()} \{ \Phi \} \]

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

\[ \text{is_mailbox}(v) \triangleq \exists \ell. v = \ell \ast \ell \leftrightarrow \text{None} \lor \exists v' \gamma. l \leftrightarrow \text{Some}(v') \ast \text{is_offer}_\gamma(v') \]
### Verifying Stacks

- **Recall desired spec:**

\[
\forall \Phi. \\
(\forall f_1 f_2. \text{wp } f_1() \{ v. v = \text{None} \lor \exists v'. v = \text{Some}(v) \ast P(v) \}) \\
\rightarrow \forall v. P(v) \rightarrow \text{wp } f_2(v) \{ \text{True} \} \\
\rightarrow \Phi(f_1, f_2) \\
\rightarrow \text{wp } \text{stack}() \{ \Phi \}
\]

- **Representation predicate:**

\[
\text{is\_stack}(v) \triangleq \mu R. \ v = \text{None} \lor \exists h, t. v = \text{Some}(h, t) \ast P(h) \ast \triangleright R(t) \\
\text{stack\_inv}(v) \triangleq \exists \ell, v'. v = \ell \ast \ell \hookrightarrow v' \ast \text{is\_stack}(v')
\]