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

\[
\text{mk\_offer} \triangleq \lambda v. (v, \text{ref}(0))
\]

\[
\text{revoke\_offer} \triangleq \lambda v. \begin{align*}
\text{let } u &= \pi_1 v \text{ in} \\
\text{let } s &= \pi_2 v \text{ in} \\
\text{if } \text{cas}(s, 0, 2) \text{ then Some } u \text{ else None}
\end{align*}
\]

\[
\text{accept\_offer} \triangleq \lambda v. \begin{align*}
\text{let } u &= \pi_1 v \text{ in} \\
\text{let } s &= \pi_2 v \text{ in} \\
\text{if } \text{cas}(s, 0, 1) \text{ then Some } u \text{ else None}
\end{align*}
\]
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.

\[
\text{mailbox} \triangleq \lambda_. \text{let } r = \text{ref}(\text{None}) \text{ in } \left( \begin{array}{c}
\lambda v. \text{let off} = \text{mk_offer } v \text{ in } \\
r \leftarrow \text{Some off}; \text{revoke_offer off }
\end{array} \right), \left( \begin{array}{c}
\lambda_. \text{let offopt} = ! r \text{ in } \\
\text{match offopt with } \\
\text{None } \Rightarrow \text{None } \\
| \text{Some } x \Rightarrow \text{accept_offer } x \\
\text{end}
\end{array} \right)
\]

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

\[
\text{stack} \triangleq \lambda \_.
\]

\[
\begin{align*}
\text{let } mb &= \text{mailbox()} \text{ in} \\
\text{let } put &= \pi_1 mb \text{ in} \\
\text{let } get &= \pi_2 mb \text{ in} \\
\text{let } r &= \text{ref}(\text{None}) \text{ in}
\end{align*}
\]
(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,
rec push() = match put() with
  None  ⇒ ()
| Some n ⇒ let r' = ! r in
     let r'' = Some(n, r') in
     if cas(r, r', r'') then ()
     else push()
end)
Stack Specification (bag-like spec)

\[
\forall \Phi. \{ \text{True} \} \text{ stack}() \left\{ \begin{array}{l}
p \exists \text{ pop push}.
\{ \text{True} \} \text{ pop}() \{ v. v = \text{None} \lor \exists v'. v = \text{Some } v' \land \Phi(v') \} \land \\
\forall v. \{ \Phi(v) \} \text{ push } v \{ u. u = () \land \text{True} \}
\end{array} \right.
\]
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:

\[ \text{stages}_\gamma(v, \ell) \triangleq (\Phi(v) \ast \ell \mapsto 0) \lor \ell \mapsto 1 \lor (\ell \mapsto 2 \ast \text{ex}(\emptyset))^{\gamma} \]

➤ Representation predicate for offers:

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

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

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

\[ \forall v. \{ \Phi(v) \} \text{ mk_offer}(v) \{ u. \exists \gamma. \text{ex}(())^\gamma \ast \text{is_offer}_\gamma(u) \} \]

- **revoke_offer** needs the token:

\[ \forall \gamma, v. \{ \text{is_offer}_\gamma(v) \ast \text{ex}(())^\gamma \} \text{ revoke_offer}(v) \{ u. u = \text{None} \lor \exists v'. u = \text{Some}(v') \ast \Phi(v') \} \]

- **accept_offer**

\[ \forall \gamma, v. \{ \text{is_offer}_\gamma(v) \} \text{ accept_offer}(v) \{ u. u = \text{None} \lor \exists v'. u = \text{Some}(v') \ast \Phi(v') \} \]
Verifying Mailboxes

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

\[
\begin{align*}
\{ \text{True} \} \mailbox() & \implies \exists \text{put, get}.
\begin{array}{c}
\forall v. \{ \Phi(v) \} \put(v) \{ w. w = \text{None} \lor \exists v'. w = \text{Some}(v') \land \Phi(v') \} \\
\{ \text{True} \} \get() \{ w. w = \text{None} \lor \exists v'. w = \text{Some}(v') \land \Phi(v') \}
\end{array}
\end{align*}
\]

(1)

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

\[
is\_mailbox(\ell) \triangleq \ell \leftrightarrow \text{None} \lor \exists v'. \ell \leftrightarrow \text{Some}(v') \land \text{is}\_offer_\gamma(v')
\]
Verifying Stacks

- Recall desired spec:

\[
\forall \Phi. \{\text{True}\} \text{ stack()} \begin{cases} 
  p. \exists \text{ pop push}. \{\text{True}\} \text{ pop()} \{v. v = \text{None} \lor \exists v'. v = \text{Some } v' \ast \Phi(v')\} \\
  \forall v. \{\Phi(v)\} \text{ push } v \{u. u = () \ast \text{True}\}
\end{cases}
\]

(2)

- Representation predicate:

\[
\text{is\_stack}(v) \triangleq v = \text{None} \lor \exists h, t. v = \text{Some}(h, t) \ast \Phi(h) \ast \triangleright \text{is\_stack}(t)
\]

\[
\text{stack\_inv}(v) \triangleq \exists v'. \ell \hookrightarrow v' \ast \text{is\_stack}(v')
\]