Iris: Higher-Order Concurrent Separation Logic

Lecture 8: Persistent Modality

Lars Birkedal

Aarhus University, Denmark

November 17, 2017

Overview

Earlier:

- Operational Semantics of $\lambda_{\mathrm{ref,conc}}$
  - e, $(h, e) \leadsto (h, e')$, and $(h, \mathcal{E}) \to (h', \mathcal{E}')$
- 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 Modality: $\triangleright$

Today:

- Persistent Modality: $\square$
- Key Points:
  - General treatment of persistent predicates.
  - Allows to recover ordinary non-sub-structural logic for reasoning about "knowledge" only (not resources).

Persistent Modality $\square$

- Earlier: explained that Hoare triples and equality predicates are *persistent* and hence may be moved in-and-out of preconditions.
- Today: systematic definition and treatment of persistent predicates.
- Informally, persistent predicates are those predicates that do not assert exclusive ownership over resources.
- Hence they only express "knowledge", not exclusive ownership.
- Hence persistent predicates P are duplicable: $P \dashv\vdash P * P$.
- Duplicable predicates are important because we can always make a copy of them to give away to other threads.
- (Why not just a "duplicable" modality? Not so easy. . . See http://cs.au.dk/ abizjak/documents/publications/box-modality-mfps.pdfBizjak and Birkedal: On Models of Higher-Order Separation Logic for an abstract detailed study.)

Persistent Modality $\square$

- Typing for $\square$:

$$\frac{\Gamma \vdash P : \mathrm{Prop}}{\Gamma \vdash \square\, P : \mathrm{Prop}}$$

- Definition: we call a proposition P *persistent* if it satisfies $P \vdash \square\, P$.

Persistent Modality $\square$

- Typing for $\square$:

$$\frac{\Gamma \vdash P : \mathrm{Prop}}{\Gamma \vdash \square\, P : \mathrm{Prop}}$$

- Definition: we call a proposition P *persistent* if it satisfies $P \vdash \square\, P$.
- Persistent modality aka Always modality

Intuitive Semantics of $\Box$ Modality

- Intuitive semantics:

$$\Box P = \{r \in \mathcal{R} \mid \exists s, r'.\, s \in P \land s = s \cdot s \land r = s \cdot r'\}$$

- You might think of this as "$\Box P$ is the upwards-closure of the set of duplicable resources in P" (recall that Iris propositions are upwards-closed wrt. resources, hence the upwards-closure).
- Example: If $\mathcal{R} = \mathit{Heap}$, then
  - the only duplicable resource is the empty heap,
  - hence, $\Box P$ is either the the set of all heaps (true), if the empty heap is in P, or the empty set (false), if the empty heap is not in P.

Laws for $\Box$

The following laws are immediate from the intuitive reading of $\Box P$:

$$\frac{P \vdash Q}{\Box P \vdash \Box Q}\ \textsc{always-mono} \qquad \frac{}{\Box P \vdash P}\ \textsc{always-E} \qquad \frac{}{\Box P \vdash \Box \Box P}\ \textsc{always-idemp}$$

Using these we can derive:

$$\frac{\Box P \vdash Q}{\Box P \vdash \Box Q}\ \textsc{always-intro}$$

Laws for $\square$

The following laws are immediate from the intuitive reading of $\square P$:

$$\frac{P \vdash Q}{\square P \vdash \square Q}\ \textsc{always-mono} \qquad \frac{}{\square P \vdash P}\ \textsc{always-E} \qquad \frac{}{\square P \vdash \square \square P}\ \textsc{always-idemp}$$

Using these we can derive:

$$\frac{\square P \vdash Q}{\square P \vdash \square Q}\ \textsc{always-intro}$$

- Assume $\square P \vdash Q$. By ALWAYS-MONO, $\square \square P \vdash \square Q$, and by ALWAYS-E, $\square P \vdash \square \square P$, so done by transitivity.

Laws for $\square$

$\square$ commutes with many of the ordinary logical connectives (note: not $\Rightarrow$):

$$
\begin{array}{rcl}
\mathsf{True} & \vdash & \square\,\mathsf{True} \\
\square(P \land Q) & \dashv\vdash & \square\, P \land \square\, Q \\
\square(P \lor Q) & \dashv\vdash & \square\, P \lor \square\, Q \\
\square \triangleright P & \dashv\vdash & \triangleright \square\, P \\
\forall x.\ \square\, P & \dashv\vdash & \square\, \forall x.\, P \\
\square\, \exists x.\, P & \dashv\vdash & \exists x.\ \square\, P
\end{array}
$$

Laws for $\Box$

$\Box$ commutes with many of the ordinary logical connectives (note: not $\Rightarrow$):

$$
\begin{array}{rcl}
\mathsf{True} & \vdash & \Box\,\mathsf{True} \\
\Box(P \wedge Q) & \dashv\vdash & \Box\, P \wedge \Box\, Q \\
\Box(P \vee Q) & \dashv\vdash & \Box\, P \vee \Box\, Q \\
\Box \triangleright P & \dashv\vdash & \triangleright \Box\, P \\
\forall x.\ \Box\, P & \dashv\vdash & \Box\,\forall x.\ P \\
\Box\,\exists x.\ P & \dashv\vdash & \exists x.\ \Box\, P
\end{array}
$$

- These facts are not supposed to be intuitively obvious; they rely on the precise semantics, which we do not cover in this course.

Laws for $\square$

$$\frac{S \vdash \square P \wedge Q}{S \vdash \square P * Q}\ \textsc{Always-sep}$$

- Intuitively sound: suppose r is in $\square P \wedge Q$. Then $r \in \square P$ and $r \in Q$. By the former, $r = s \cdot r'$ for some $s \in P$ such that $s \cdot s = s$. Hence also $s \in \square P$. Moreover, $r = s \cdot r' = (s \cdot s) \cdot r' = s \cdot (s \cdot r') = s \cdot r$. Hence r is in $\square P * Q$, as required.

Laws for $\Box$

Derivable rule:

$$\frac{S \vdash \Box(P \wedge Q)}{S \vdash \Box(P * Q)}\ \textsc{Always-sep-derived}$$

which is equivalent to the entailment

$$\Box(P \wedge Q) \vdash \Box(P * Q).$$

Proof of $\Box(P \wedge Q) \vdash \Box(P * Q)$.

- We first show

$$\frac{P \vdash \Box P}{P \vdash P * P}. \tag{1}$$

- Indeed, using $P \vdash \Box P$ we have

$$P \vdash \Box P \vdash \Box P \wedge \Box P \vdash \Box P * \Box P \vdash P * P$$

  using rule Always-sep in the third step.
- Next, since $\Box$ commutes with conjunction and the above (1), we have

$$\Box P \wedge \Box Q \vdash \Box(P \wedge Q) \vdash \Box(P \wedge Q) * \Box(P \wedge Q)$$

- Now, using the fact that $P \wedge Q \vdash P$ and $P \wedge Q \vdash Q$ and monotonicity, we have

$$\Box(P \wedge Q) * \Box(P \wedge Q) \vdash \Box P * \Box Q.$$

Proof continued

- Hence we have proved

$$\Box P \wedge \Box Q \vdash \Box P * \Box Q. \tag{2}$$

- Finally, we get the desired by:

$$\Box(P \wedge Q) \vdash \Box\Box(P \wedge Q) \vdash \Box(\Box P \wedge \Box Q) \vdash \Box(\Box P * \Box Q) \vdash \Box(P * Q)$$

  where in the last step we use $\Box P \vdash P$ for any P and monotonicity of separating conjunction.

Laws for $\Box$

We have two kinds of primitive persistent propositions.

$$t =_\tau t' \dashv\vdash \Box(t =_\tau t') \qquad\qquad \{P\}\, e\, \{\Phi\} \dashv\vdash \Box\, \{P\}\, e\, \{\Phi\}$$

Finally, we have the following rule generalizing the in-out rules (Ht-Ht and Ht-Eq) we saw earlier) we saw earlier:

$$\textsc{Ht-always}\quad \frac{\Box\, Q \wedge S \vdash \{P\}\, e\, \{v.\, R\}}{S \vdash \{P \wedge \Box\, Q\}\, e\, \{v.\, R\}}$$

Homework Exercise

Using similar reasoning as in the proof above show the following derived rules.

1. $\square \square P \vdash \square P$
2. $\square(P \Rightarrow Q) \vdash \square P \Rightarrow \square Q$
3. $P \Rightarrow Q \vdash P \twoheadrightarrow Q$
4. $\square(P \twoheadrightarrow Q) \vdash \square(P \Rightarrow Q)$
5. $\square(P \twoheadrightarrow Q) \vdash \square P \twoheadrightarrow \square Q$
6. $(P \twoheadrightarrow (Q * \square R)) * P \vdash (P \twoheadrightarrow (Q * \square R)) * P * \square R$

Example

Recall the stack example from earlier:

$$\exists \mathrm{isStack} : \mathit{Val} \to \mathrm{list}\,\mathit{Val} \to (\mathit{Val} \to \mathsf{Prop}) \to \mathsf{Prop}.$$
$$\forall \Phi : \mathit{Val} \to \mathsf{Prop}.$$
$$\{\mathsf{True}\}\ \mathsf{mk\_stack}()\ \{s.\mathrm{isStack}(s, [], \Phi)\} \wedge$$
$$\forall s. \forall xs. \{\mathrm{isStack}(s, xs, \Phi) * \Phi(x)\}\ \mathsf{push}(x, s)\ \{v.v = () \wedge \mathrm{isStack}(s, x : xs, \Phi)\} \wedge$$
$$\forall s. \forall x, xs. \{\mathrm{isStack}(s, x : xs, \Phi)\}\ \mathsf{pop}(s)\ \{v.v = x \wedge \mathrm{isStack}(s, xs, \Phi) * \Phi(x)\}$$

The idea is that $\mathrm{isStack}(s, xs, \Phi)$ asserts that s is a stack whose values are xs and all of the values $x \in xs$ satisfy the given predicate Φ.

Example

Suppose we write in the fully modular ADT style:

$$\{\mathsf{True}\}$$

$$\mathsf{mk\_stack}()$$

$$\left\{ \begin{array}{l} s.\ \exists \mathrm{isStack}.\, \forall \Phi. \\ \quad \mathrm{isStack}(s, [], \Phi) * \\ \quad \forall s. \forall xs. \{\mathrm{isStack}(s, xs, \Phi) * \Phi(x)\}\ \mathsf{push}(x, s)\ \{v. v = () \wedge \mathrm{isStack}(s, x : xs, \Phi)\} \wedge \\ \quad \forall s. \forall x, xs. \{\mathrm{isStack}(s, x : xs, \Phi)\}\ \mathsf{pop}(s)\ \{v. v = x \wedge \mathrm{isStack}(s, xs, \Phi) * \Phi(x)\} \end{array} \right\}$$

Example: adding an iterator

- Suppose we wish to add an iterator function to the module (similarly to what they have in OCaml, see http://caml.inria.fr/pub/docs/manual-ocaml/libref/Stack.html):
  - `val iter : ('a -> unit) -> 'a t -> unit`
  - `iter f s` applies `f` in turn to all elements of `s`, from the element at the top of the stack to the element at the bottom of the stack. The stack itself is unchanged.
- Then we wish to add a specification of the iterator to our stack specification, *e.g.*, as follows:

Example: iterator spec, v1

$$\{\mathsf{True}\}$$

$$\quad \mathsf{mk\_stack}()$$

$$\left\{\begin{array}{l}
s.\ \exists \mathrm{isStack}.\,\forall\Phi.\\
\quad \mathrm{isStack}(s, [], \Phi) *\\
\quad \forall s.\forall xs.\{\mathrm{isStack}(s, xs, \Phi) * \Phi(x)\}\ \mathsf{push}(x, s)\ \{v.v = () \wedge \mathrm{isStack}(s, x : xs, \Phi)\} \wedge\\
\quad \forall s.\forall x, xs.\{\mathrm{isStack}(s, x : xs, \Phi)\}\ \mathsf{pop}(s)\ \{v.v = x \wedge \mathrm{isStack}(s, xs, \Phi) * \Phi(x)\} \wedge\\
\quad\ \forall s.\forall xs.\forall\Psi.\\
\qquad \forall x.\{\Phi(x)\}\ f(x)\ \{v.v = () \wedge \Psi(x)\}\\
\qquad \Rightarrow\\
\qquad \{\mathrm{isStack}(s, xs, \Phi)\}\ \mathrm{iter}(f, s)\ \{v.v = () \wedge \mathrm{isStack}(s, xs, \Psi)\}
\end{array}\right\}$$

Example: iterator spec

- When we use the stack module, *e.g.*, like this

$$\text{let } s = \mathrm{mk\_stack}() \text{ in } e$$

  then we use the $\textsc{Ht-let-det}$ rule, and thus when we prove something for e, the precondition will essentially be the postcondition of the stack spec above.
- To reason about calls to the iterator, we will then need to move the spec for the iterator into the context.
- To do that, we need to know that the iterator spec (the formula above) is PERSISTENT.
- However, that is not necessarily the case, since persistent predicates are not closed under $\Rightarrow$.
- Hence we need to use the $\Box$ modality!

Example: iterator spec, v2

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\quad \mathsf{mk\_stack}() \\
\left\{
\begin{array}{l}
s.\ \exists \mathrm{isStack}.\, \forall \Phi. \\
\quad \mathrm{isStack}(s, [], \Phi) * \\
\quad \forall s. \forall xs. \{\mathrm{isStack}(s, xs, \Phi) * \Phi(x)\}\ \mathsf{push}(x, s)\ \{v. v = () \wedge \mathrm{isStack}(s, x : xs, \Phi)\} \wedge \\
\quad \forall s. \forall x, xs. \{\mathrm{isStack}(s, x : xs, \Phi)\}\ \mathsf{pop}(s)\ \{v. v = x \wedge \mathrm{isStack}(s, xs, \Phi) * \Phi(x)\} \wedge \\
\quad \Box \left(
\begin{array}{l}
\forall s. \forall xs. \forall \Psi. \\
\quad \forall x. \{\Phi(x)\}\ f(x)\ \{v. v = () \wedge \Psi(x)\} \\
\quad \Rightarrow \\
\quad \{\mathrm{isStack}(s, xs, \Phi)\}\ \mathrm{iter}(f, s)\ \{v. v = () \wedge \mathrm{isStack}(s, xs, \Psi)\}
\end{array}
\right)
\end{array}
\right\}
\end{array}
$$