Iris: Higher-Order Concurrent Separation Logic

Lecture 8: Persistent Modality

Lars Birkedal

Aarhus University, Denmark

November 17, 2017
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
  - $l \hookrightarrow v, P * Q, P \rightarrow Q, \Gamma \mid P \vdash Q$
- Basic Separation Logic
  - $\{P\} e \{v.Q\} : \text{Prop, isList} \mid l \text{ xs, ADTs, foldr}$
- Later Modality: $\triangleright$

Today:

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

- 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: $P \vdash P \star P$.
- Duplicable predicates are important because we can always make a copy of them to give away to other threads.
Persistent Modality □

▶ Typing for □:

\[ \Gamma \vdash P : \text{Prop} \]
\[ \Gamma \vdash \square P : \text{Prop} \]

▶ Definition: we call a proposition \( P \) *persistent* if it satisfies \( P \vdash \square P \).
Persistent Modality □

- Typing for □:

\[
\Gamma \vdash P : \text{Prop} \\
\hline
\Gamma \vdash \Box P : \text{Prop}
\]

- Definition: we call a proposition \( P \) persistent if it satisfies \( P \vdash \Box P \).

- Persistent modality aka Always modality
Intuitive Semantics of □ Modality

- Intuitive semantics:

\[ \square P = \{ r \in \mathcal{R} \mid \exists s, r'. s \in P \land s = s \cdot s \land r = s \cdot r' \} \]

- You might think of this as “\( \square P \) 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 \( \mathcal{R} = \text{Heap} \), then
  - the only duplicable resource is the empty heap,
  - hence, \( \square P \) 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 \( P \).
The following laws are immediate from the intuitive reading of $\Box P$:

\[
\begin{align*}
\text{ALWAYS-MONO} & \quad P \vdash Q \\
& \quad \Box P \vdash \Box Q \\
\text{ALWAYS-E} & \quad \Box P \vdash P \\
\text{ALWAYS-IDEEMP} & \quad \Box P \vdash \Box \Box P
\end{align*}
\]

Using these we can derive:

\[
\begin{align*}
\text{ALWAYS-INTRO} & \quad \Box P \vdash Q \\
& \quad \Box P \vdash \Box Q
\end{align*}
\]
Laws for □

The following laws are immediate from the intuitive reading of □ P:

\[
\begin{align*}
\text{ALWAYS-MONO} & \quad P \vdash Q \\
& \quad \Box P \vdash \Box Q \\
\text{ALWAYS-E} & \quad \Box P \vdash P \\
\text{ALWAYS-IDEMP} & \quad \Box P \vdash \Box \Box P
\end{align*}
\]

Using these we can derive:

\[
\begin{align*}
\text{ALWAYS-INTRO} & \quad \Box P \vdash Q \\
& \quad \Box P \vdash \Box Q
\end{align*}
\]

▶ Assume □ P ⊢ Q. By ALWAYS-MONO, □ □ P ⊢ □ Q, and by ALWAYS-E, □ P ⊢ □ □ P, so done by transitivity.
Laws for □

□ commutes with many of the ordinary logical connectives (note: not ⇒):

True ⊢ □ True
□(P ∧ Q) ⊨ □ P ∧ □ Q
□(P ∨ Q) ⊨ □ P ∨ □ Q
□ □P ⊨ □ □P
∀x. □ P ⊨ □ ∀x. P
□ ∃x. P ⊨ ∃x. □ P

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 □

□ commutes with many of the ordinary logical connectives (note: not ⇒):

- \(\text{True} \vdash \square \text{True}\)
- \(\square (P \land Q) \not\vdash \square P \land \square Q\)
- \(\square (P \lor Q) \not\vdash \square P \lor \square Q\)
- \(\square \triangleright P \not\vdash \triangleright \square P\)
- \(\forall x. \square P \not\vdash \square \forall x. P\)
- \(\square \exists x. P \not\vdash \exists x. \square P\)

- 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 □

\[ \text{Always-sep} \]
\[ S \vdash □ P \land Q \]
\[ \vdash □ P \ast Q \]

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

Derivable rule:

\[
\begin{align*}
\text{ALWAYS-SEP-DERIVED} \\
S \vdash □ (P \land Q) \\
\hline \\
S \vdash □ (P \ast Q)
\end{align*}
\]

which is equivalent to the entailment

\[
□ (P \land Q) \vdash □ (P \ast Q).
\]
Proof of \( \Box (P \land Q) \vdash \Box (P \ast Q) \).

- We first show

\[
\begin{align*}
P & \vdash \Box P \\
\Box P & \vdash \Box P \land \Box P \\
\Box P \land \Box P & \vdash \Box P \ast \Box P.
\end{align*}
\]  

(1)

- Indeed, using \( P \vdash \Box P \) we have

\[
P \vdash \Box P \vdash \Box P \land \Box P \vdash \Box P \ast \Box P \vdash P \ast P
\]

using rule **Always-sep** in the third step.

- Next, since \( \Box \) commutes with conjunction and the above (1), we have

\[
\Box P \land \Box Q \vdash \Box(P \land Q) \vdash \Box(P \land Q) \ast \Box(P \land Q)
\]

- Now, using the fact that \( P \land Q \vdash P \) and \( P \land Q \vdash Q \) and monotonicity, we have

\[
\Box(P \land Q) \ast \Box(P \land Q) \vdash \Box P \ast \Box Q.
\]
Proof continued

▶ Hence we have proved

$\Box P \land \Box Q \vdash \Box P \ast \Box Q.$ \hspace{1cm} (2)

▶ Finally, we get the desired by:

$\Box(P \land Q) \vdash \Box\Box(P \land Q) \vdash \Box(\Box P \land \Box Q) \vdash \Box(\Box P \ast \Box Q) \vdash \Box(P \ast Q)$

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

We have two kinds of primitive persistent propositions.

\[ t =_\tau t' \vdash \square (t =_\tau t') \quad \{ P \} e \{ \Phi \} \vdash \square \{ P \} e \{ \Phi \} \]

Finally, we have the following rule generalizing the in-out rules (\( \text{HT-HT} \) and \( \text{HT-EQ} \) we saw earlier) we saw earlier:

\[
\begin{align*}
\text{HT-ALWAYS} \\
\quad \square Q \land S \vdash \{ P \} e \{ v. R \} \\
\quad S \vdash \{ P \land \square Q \} e \{ v. R \}
\end{align*}
\]
Using similar reasoning as in the proof above show the following derived rules.

1. $\Box\Box P \vdash \Box P$
2. $\Box(P \Rightarrow Q) \vdash \Box P \Rightarrow \Box Q$
3. $P \Rightarrow Q \vdash P \Rightarrow Q$
4. $\Box(P \Rightarrow Q) \vdash \Box(P \Rightarrow Q)$
5. $\Box(P \Rightarrow Q) \vdash \Box P \Rightarrow \Box Q$
6. $(P \Rightarrow (Q \Rightarrow R)) \Rightarrow P \vdash (P \Rightarrow (Q \Rightarrow R)) \Rightarrow P \Rightarrow R$
Recall the stack example from earlier:

\[ \exists \text{isStack} : \text{Val} \to \text{list Val} \to (\text{Val} \to \text{Prop}) \to \text{Prop.} \]
\[ \forall \Phi : \text{Val} \to \text{Prop.} \]
\[ \{\text{True}\} \ \text{mk\_stack()} \ \{s.\text{isStack}(s, [], \Phi)\} \land \]
\[ \forall s.\forall xs.\{\text{isStack}(s, xs, \Phi) \ast \Phi(x)\} \ \text{push}(x, s) \ \{v.v = () \land \text{isStack}(s, x : xs, \Phi)\} \land \]
\[ \forall s.\forall x, xs.\{\text{isStack}(s, x : xs, \Phi)\} \ \text{pop}(s) \ \{v.v = x \land \text{isStack}(s, xs, \Phi) \ast \Phi(x)\} \]

The idea is that \text{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 \( \Phi \).
Example

Suppose we write in the fully modular ADT style:

\[
\{ \text{True} \} \\
\text{mk\_stack()} \\
\left\{ \\
\quad s. \ \exists \text{isStack}. \ \forall \Phi. \\
\quad \quad \text{isStack}(s, [], \Phi) \ast \\
\quad \quad \forall s. \forall xs. \{ \text{isStack}(s, xs, \Phi) \ast \Phi(x) \} \ \text{push}(x, s) \{ v. v = () \land \text{isStack}(s, x : xs, \Phi) \} \land \\
\quad \quad \forall s. \forall x, xs. \{ \text{isStack}(s, x : xs, \Phi) \} \ \text{pop}(s) \{ v. v = x \land \text{isStack}(s, xs, \Phi) \ast \Phi(x) \} \\
\right\}
\]
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

{True}

mk_stack() 

s. ∃isStack. ∀Φ.
    isStack(s, [], Φ) *
    ∀s.∀xs.{isStack(s, xs, Φ) * Φ(x)} push(x, s) {v.v = () ∧ isStack(s, x : xs, Φ)} ∧
    ∀s.∀x, xs.{isStack(s, x : xs, Φ)} pop(s) {v.v = x ∧ isStack(s, xs, Φ) * Φ(x)} ∧
    ∀s.∀xs.∀Ψ.
        ∀x.{Φ(x)} f(x) {v.v = () ∧ Ψ(x)}
        ⇒
        {isStack(s, xs, Φ)} iter(f, s) {v.v = () ∧ isStack(s, xs, Ψ)}
Example: iterator spec

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

  \[
  \text{let } s = \text{mk\_stack()} \text{ in } e
  \]

  then we use the $\text{HT-LET-DET}$ 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 $\square$ modality!
Example: iterator spec, v2

\{\text{True}\}

\text{mk\_stack()}

\begin{align*}
\text{s. } & \exists \text{isStack. } \forall \Phi. \\
& \text{isStack}(s, [], \Phi)^* \\
& \forall s. \forall xs. \{\text{isStack}(s, xs, \Phi) \land \Phi(x)\} \land \\
& \forall s. \forall x, xs. \{\text{isStack}(s, x : xs, \Phi)\} \land \\
& \forall s. \forall x, xs. \forall \Psi. \\
& \square \left( \forall x. \{\Phi(x)\} f(x) \{v. v = () \land \Psi(x)\} \Rightarrow \\ & \{\text{isStack}(s, xs, \Phi)\} \text{ iter}(f, s) \{v. v = () \land \text{isStack}(s, xs, \Psi)\} \right)
\end{align*}