Modern Separation Logic — Derek Dreyer

Lecture 3
June 30, 2026

1 A Motivating Example: WP Proofs

Firstly, let us introduce two notations:

$$\begin{aligned} \text{let } x := e_1 \text{ in } e_2 &\triangleq (\lambda x.\, e_2) e_1 \\ e_1; e_2 &\triangleq (\lambda \_.\, e_2) e_1 \end{aligned}$$

Then, consider the following definitions:

$$\begin{aligned} \texttt{foo} &\triangleq \lambda r_1.\, \lambda r_2.\, r_1 \leftarrow 23 \\ \texttt{bar} &\triangleq \text{let } x := \text{new } 0 \text{ in} \\ &\quad\ \text{let } y := \text{new } 42 \text{ in} \\ &\quad\ \texttt{foo}\ x\ y;\ !x + !y \end{aligned}$$

We want to prove two specifications:

1. $\forall r_1, r_2, w.\ \{r_1 \mapsto w\}\ \texttt{foo}\ r_1\ r_2\ \{\_.\, r_1 \mapsto 23\}$
2. $\{\mathsf{True}\}\ \texttt{bar}\ \{v.\, v = 65\}$

1.1 Heap Command Rules

As a reference, these are the rules of heap commands from the last slides of Lecture 2:

Compiled By:
Ricky Cheng, Kalab Assefa, Ziyang Wang

New:

$$\forall l.\, l \mapsto v \mathrel{-\!\!*} Q(l) \vdash \text{wp}\ \ (\text{new}\ v)\ \{w.\, Q(w)\}$$

Store:

$$l \mapsto u * (l \mapsto v \mathrel{-\!\!*} Q(\langle\rangle)) \vdash \text{wp}\ \ (l \leftarrow v)\ \{w.\, Q(w)\}$$

Load:

$$l \mapsto u * (l \mapsto u \mathrel{-\!\!*} Q(u)) \vdash \text{wp}\ \ (!l)\ \{w.\, Q(w)\}$$

1.2 Proof of `foo`

Let ℓ be the location bound to r_1 and v be the value bound to r_2. Our context initially contains the ownership $\ell \mapsto w$. We want to prove the specification:

$$\text{wp}\ \texttt{foo}\ \ell\ v\ \{\_.\, \ell \mapsto 23\}$$

First, we unfold the definition of `foo` and evaluate the pure computational steps (beta-reduction of the function application):

$$\text{wp}\ (\lambda r_1.\, \lambda r_2.\, r_1 \leftarrow 23)\ \ell\ v\ \{\_.\, \ell \mapsto 23\}$$

By the pure step rule, this reduces to our weakest precondition for the store operation:

$$\text{wp}\ (\ell \leftarrow 23)\ \{\_.\, \ell \mapsto 23\}$$

Now we apply the STORE rule. Our context currently holds $\ell \mapsto w$. The STORE rule consumes this points-to fact and updates the heap location, leaving our context with $\ell \mapsto 23$.

Because our updated context exactly matches our required postcondition (and the return value of a store is the unit value, which satisfies the wildcard _), the proof is complete.

1.3 Proof of `bar`

Let our goal condition be $Q(v) \triangleq v = 65$. After evaluating the allocations for x and y (let's call the allocated locations ℓ_1 and ℓ_2), our context contains the spatial resources $\ell_1 \mapsto 0$ and $\ell_2 \mapsto 42$.

Our goal is to prove:

$$\text{wp}\ (\texttt{foo}\ \ell_1\ \ell_2;\ !\ell_1 + !\ell_2)\ \{Q\}$$

Using the BIND rule on `foo`, we evaluate the function call first:

$$\text{wp } \texttt{foo}\ \ell_1\ \ell_2\ \{v.\,\text{wp } (v;\ !\ell_1{+}!\ell_2)\ \{Q\}\}$$

Next, we apply the WAND rule to utilize the specification of `foo`. We must provide the precondition of `foo` ($\ell_1 \mapsto 0$) and prove that its postcondition ($\ell_1 \mapsto 23$) implies our continuation:

$$\text{wp } \texttt{foo}\ \ell_1\ \ell_2\ \{\ell_1 \mapsto 23\} * (\ell_1 \mapsto 23 \mathrel{-\!\!*} \text{wp } ();\ !\ell_1{+}!\ell_2\ \{Q\})$$

The first conjunct is exactly our specification for `foo`. For the second conjunct, we assume $\ell_1 \mapsto 23$ and continue evaluating the rest of the expression:

$$\text{wp } ();\ !\ell_1{+}!\ell_2\ \{Q\}$$

With $\ell_1 \mapsto 23$ and $\ell_2 \mapsto 42$ in our context, loading the values yields:

$$\text{wp } (23 + 42)\ \{Q\} \implies Q(65)$$

Which holds trivially, completing the proof.

2 Shared State and Internalizing Hoare Triples

2.1 The `MutBit` Example

Consider an allocator for a shared mutable bit. It returns a pair of closures: one to flip the bit, and one to read it.

$$\begin{aligned}\texttt{MutBitImpl} \triangleq\ &\text{let } x := \text{new } 0 \text{ in}\\ &\langle \lambda\langle\rangle.\, x \leftarrow 1 - !x,\\ &\ \lambda\langle\rangle.\, \text{let } y := !x \text{ in}\\ &\qquad \text{if } y = 0 \text{ then false}\\ &\qquad \text{else if } y = 1 \text{ then true}\\ &\qquad \text{else assert false}\rangle\end{aligned}$$

We want to prove $\{\mathsf{True}\}\ \texttt{MutBitImpl}\ \{v.\,\text{MutBit}(v)\}$. Intuitively, to match the specification, we want to define $\text{MutBit}(v)$ as:

$$\text{MutBit}(v) := \{\mathsf{True}\}\ v.\text{flip}()\ \{w.\,w = ()\} * \{\mathsf{True}\}\ v.\text{get}()\ \{w.\,w \in \mathbb{B}\}$$

2.2 The Problem with Standard Hoare Triples

The specification MutBit poses two problems:

1. **Loss of State:** Standard Hoare triples are defined as entailments $P \vdash \text{wp } e\ \{v.\, Q(v)\}$, which is a meta-level property, not an Iris proposition. If we embed them as meta-level propositions, we lose all information about the current program state (e.g., the location x).
2. **Single-Use Ownership:** If we try to define $\{P\}e\{v.\, Q(v)\} := P \mathrel{-\!\!*} \text{wp } e\ \{v.\, Q(v)\}$ as an Iris proposition, it acts as a linear implication. We could keep our ownership of x, but we could only use the triple *once*! Any subsequent calls to 'flip' would fail because the resource would be consumed.

What we really want is for Hoare triples to be *duplicable*:

$$\{P\}\ e\ \{v.\, Q(v)\} \vdash \{P\}\ e\ \{v.\, Q(v)\} * \{P\}\ e\ \{v.\, Q(v)\}$$

3 Persistent Propositions

In Iris, we call propositions which can be duplicated (i.e., $P \vdash P * P$) *persistent*. We introduce a modality $\Box P$ (read: "P holds, and it holds persistently"), which "makes" propositions persistent.

With this modality, we can properly internalize Hoare triples into Iris:

$$\{P\}\ e\ \{v.\, Q(v)\} \triangleq \Box(P \mathrel{-\!\!*} \text{wp } e\ \{v.\, Q(v)\})$$

3.1 Rules for Persistent Propositions

The defining characteristics of $\Box P$ are that it can be eliminated to get the underlying proposition, and it can be freely duplicated:

$$\textsc{PersElim}\quad \frac{}{\Box P \vdash P} \qquad \textsc{PersDup}\quad \frac{}{\Box P \vdash (\Box P) * (\Box P)}$$

To *prove* and interact with persistent propositions, Iris provides the following structural rules:

$$\textsc{PersMono}\ \frac{P \vdash Q}{\Box P \vdash \Box Q} \qquad \textsc{PersPure}\ \frac{}{\phi \vdash \Box \phi} \qquad \textsc{PersAndSep}\ \frac{}{(\Box P) \wedge Q \vdash (\Box P) * Q} \qquad \textsc{PersIdemp}\ \frac{}{\Box P \vdash \Box\Box P}$$

$$\textsc{PersAll}\ \frac{}{\forall x : X.\, \Box P(x) \vdash \Box \forall x : X.\, P(x)} \qquad \textsc{PersExists}\ \frac{}{\Box \exists x : X.\, P(x) \vdash \exists x : X.\, \Box P(x)}$$

3.2 Example: Persistent Framing

Because persistent propositions can be duplicated and separate via PersAndSep (which effectively means $\Box P * \Box P \dashv\vdash \Box P$), they are incredibly useful for framing.

For example, if we know $\Box R * P \vdash P'$ and $\Box R * Q \vdash Q'$, we can prove:

$$\Box R * P * Q \vdash P' * Q'$$

Proof sketch: We use PersDup to duplicate $\Box R$ into $(\Box R) * (\Box R)$. We then rearrange the separating conjunctions to form $(\Box R * P) * (\Box R * Q)$, which directly entails $P' * Q'$ by our assumptions.