Iris: Higher-Order Concurrent Separation Logic

Lecture 3: Basic Separation Logic: Hoare Triples

Lars Birkedal

Aarhus University, Denmark

November 10, 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
  - $l \hookrightarrow v, P \ast Q, P \rightarrow Q, \Gamma \mid P \vdash Q$

Today:

- Basic Separation Logic: Hoare Triples
  - $\{P\} e \{v.Q\} : \text{Prop}$
Hoare Triples

\[ \Gamma \vdash P : \text{Prop} \quad \Gamma \vdash e : \text{Exp} \quad \Gamma \vdash \Phi : \text{Val} \to \text{Prop} \]

\[ \Gamma \vdash \{P\} e \{\Phi\} : \text{Prop} \]

Intuition

- \{P\} e \{\Phi\} holds if, when we run the program \(e\) in a heap \(h\) satisfying \(P\), then the computation does not get stuck and, moreover, if it terminates with a value \(v\) and a heap \(h'\), then \(h'\) satisfies \(\Phi(v)\).

- \(\Phi\) has two purposes: describes the value \(v\) (e.g., \(v = 3\)) and the resources after execution (e.g., \(x \leftarrow 15\)).

- Note that \(\Phi\) is a function – we often write \(\Phi\) as \(v. Q\) instead of \(\lambda v. Q\).

Examples

\[ \{l \leftarrow 5\} l \leftarrow !l + 1 \{v. v = () \land l \leftarrow 6\} \]

\[ \{\ell_1 \leftarrow v_1 \ast \ell_2 \leftarrow v_2\} \text{swap} \ell_1 \ell_2 \{v. v = () \land \ell_1 \leftarrow v_2 \ast \ell_2 \leftarrow v_1\}. \]
Hoare Triples

More intuition

- Precondition $P$ describes the resources necessary to run $e$ safely (recall “does not get stuck” in the intuitive reading above).
- In our operational semantics, memory errors, e.g., trying to dereference a location that has not been allocated, are modelled by the computation getting stuck.
- So if $e$ satisfies a Hoare triple, then its computation will not lead to any memory errors.
- Precondition $P$ describes the resources needed for $e$ to run safely: we sometimes say that $P$ includes the footprint of $e$.
- (Later on, not all resources needed to execute $e$ will need to be in the precondition — resources shared among different threads will be in invariants, and only resources owned by $e$’s thread will in the precondition.)
Frame Rule

\[
\text{HT-FRAME} \\
S \vdash \{P\} e \{v.Q\} \\
S \vdash \{P \ast R\} e \{v.Q \ast R\}
\]

- Intuitively sound because of the footprint reading of triples
- Note that the frame \(R\) is maintained unchanged from precondition to postcondition.
- We do not have to explicitly say that \(e\) does not modify other resources not in its precondition!
- Very important!
- We will use this rule all the time.
Frame Rule

Example

- Consider the specification for swap:

\[
\{\ell_1 \leftrightarrow v_1 \ast \ell_2 \leftrightarrow v_2\} \text{swap } \ell_1 \ell_2 \{v.v = () \land \ell_1 \leftrightarrow v_2 \ast \ell_2 \leftrightarrow v_1\}.
\]

- What if we want to apply this function somewhere, where we have more resources around? For instance \(\ell_3 \leftrightarrow 3\). Then we use the frame rule, with frame \(R = \ell_3 \leftrightarrow 3\), to derive

\[
\{\ell_1 \leftrightarrow v_1 \ast \ell_2 \leftrightarrow v_2 \ast \ell_3 \leftrightarrow 3\} \text{swap } \ell_1 \ell_2 \{v.v = () \land \ell_1 \leftrightarrow v_2 \ast \ell_2 \leftrightarrow v_1 \ast \ell_3 \leftrightarrow 3\}.
\]
Value Rule

**HT-ret**

\[
\begin{align*}
S & \vdash \{\text{True}\} \ w \ \{v. \ v = w\} \\
\text{w is a value}
\end{align*}
\]
Basic rules are given for values, e.g.,

\[
\text{HT-BINOP} \\
\begin{align*}
v_1 \text{ and } v_2 \text{ are values} \\
S \vdash \{ \text{True} \} v_1 \odot v_2 \{ v. v = v_1 \odot v_2 \}
\end{align*}
\]

Here the latter \( \odot \) is the mathematical operation corresponding to the syntactic operator.
To verify larger expressions we use the \(HT\)-BIND rule:

\[
\begin{array}{c}
\text{HT-BIND} \\
E \text{ is an eval. context} & S \vdash \{P\} e \{v. Q\} & S \vdash \forall v. \{Q\} E[v] \{w. R\} \\
& S \vdash \{P\} E[e] \{w. R\}
\end{array}
\]

- Exercise: Use \(HT\)-BIND to show \(\{\text{True}\} 3 + 4 + 5 \{v. v = 12\}\).
Persistent Propositions

- Intuition: persistent propositions are propositions that do not rely on resources, i.e., either they hold for all resources or none.

\[ P \land Q \vdash P \ast Q \quad \text{if } P \text{ is persistent.} \]

- Persistent propositions may be moved in and out of preconditions:

  - **HT-EQ**
    \[
    \begin{align*}
    S \land t =_{\tau} t' & \vdash \{P\} e \{v.Q\} \\
    S & \vdash \{P \land t =_{\tau} t'\} e \{v.Q\}
    \end{align*}
    \]

  - **HT-HT**
    \[
    \begin{align*}
    S \land \{P_1\} e_1 \{v.Q_1\} & \vdash \{P_2\} e_2 \{v.Q_2\} \\
    S & \vdash \{P_2 \land \{P_1\} e_1 \{v.Q_1\}\} e_2 \{v.Q_2\}
    \end{align*}
    \]

- For now it suffices to know that persistence is preserved by \(\forall\) and \(\land\) — we will see a general treatment later.
Example of $\text{HT-HT}$

$$
\{x \leftarrow 5\} \text{inc} x \{v.v = () \land x \leftarrow 6\} \vdash \{x \leftarrow 5\} \text{inc} x \{v.v = () \land x \leftarrow 6\}
$$

$$
\{x \leftarrow 5 \land \{x \leftarrow 5\} \text{inc} x \{v.v = () \land x \leftarrow 6\}\} \text{inc} x \{v.v = () \land x \leftarrow 6\}
$$
Consequence Rule

\[ \text{HT-CSQ} \]

\[
\begin{array}{c}
S \text{ persistent} \\
S \vdash P \Rightarrow P' \\
S \vdash \{P'\} e \{v. Q'\} \\
S \vdash \forall u. Q'[u/v] \Rightarrow Q[u/v] \\
S \vdash \{P\} e \{v. Q\}
\end{array}
\]

Remark: \( S \) is usually a conjunction of equalities and universally quantified Hoare triples, so is usually persistent.
Load Rule

\[ \text{HT-load} \]

\[ S \vdash \{ \ell \leftrightarrow u \} \! \ell \{ v.v = u \land \ell \leftrightarrow u \} \]

- Intuitively sound because . . .
Alloc Rule

\[ \text{HT-ALLOC} \]

\[ S \vdash \{ \text{True} \} \text{ref}(u) \{ v. \exists \ell. v = \ell \land \ell \leftrightarrow u \} \]

- Intuitively sound because ...
HT-STORE

\[ S \vdash \{ \ell \leftrightarrow - \} \ell \leftarrow w \{ v.v = () \land \ell \leftrightarrow w \} \]

- \( \ell \leftrightarrow - \) shorthand for \( \exists u. \ell \leftrightarrow u \)
- Intuitively sound because . . .
Rules for Conditionals

\[
\text{HT-If-True} \\
\{ P \ast v = \text{true} \} \quad e_2 \quad \{ u. Q \} \\
\{ P \ast v = \text{true} \} \quad \text{if } v \text{ then } e_2 \text{ else } e_3 \quad \{ u. Q \}
\]

\[
\text{HT-If-False} \\
\{ P \ast v = \text{false} \} \quad e_3 \quad \{ u. Q \} \\
\{ P \ast v = \text{false} \} \quad \text{if } v \text{ then } e_2 \text{ else } e_3 \quad \{ u. Q \}
\]

\[
\text{HT-If} \\
\{ P \ast v = \text{true} \} \quad e_1 \quad \{ u. Q \} \\
\{ P \ast v = \text{false} \} \quad e_3 \quad \{ u. Q \} \\
\{ P \} \quad \text{if } v \text{ then } e_2 \text{ else } e_3 \quad \{ u. Q \}
\]
Rules for Products and Sums

\[
\text{Proj} \quad S \vdash \{\text{True}\} \pi_i(v_1, v_2) \{v \cdot v = v_i\}
\]

\[
\text{Match} \quad S \vdash \{P\} e_i[u/x_i] \{v \cdot Q\}
\]

\[
S \vdash \{P\} \text{match inj}_i u \text{ with } x_1 \Rightarrow e_1 \mid x_2 \Rightarrow e_2 \text{ end } \{v \cdot Q\}
\]
Recursion Rule

\[ \text{HT-Rec} \]
\[ \Gamma, f : Val \mid S \land \forall y. \forall v. \{P\} f v \{u.Q\} \vdash \forall y. \forall v. \{P\} e[v/x] \{u.Q\} \]
\[ \Gamma \mid S \vdash \forall y. \forall v. \{P\} (\text{rec } f(x) = e) v \{u.Q\} \]

- Here \( y \) is a “logical” variable, which may be used in \( P \) and \( Q \) to relate pre and postconditions. Example:
  - \( \forall y : \mathbb{N}. \forall x. \{x = y\} \text{ double } x \{v. v = \text{val } 2 \times y\} \)
- When reasoning about the body, we get to assume that \( f \) satisfies the triple we are about to prove.
- Intuitively sound by induction on reduction steps.
Exercise (jointly, on the board)

- Specify and prove a functional implementation of factorial.
Factorial

Implementation

- \(\text{rec fac}(n) = \text{if } n = 0 \text{ then } 1 \text{ else } n \times \text{fac}(n - 1)\)

Specification

- \(\forall n : \mathbb{N}.\{\text{True}\} \text{ fac } n \{v. \ v = \text{Val } n!\}\)

Proof

- Use the recursion rule!