Iris: Higher-Order Concurrent Separation Logic

Lecture 4: Basic Separation Logic: Proving Pointer Programs

Lars Birkedal

Aarhus University, Denmark

November 10, 2017

Overview

Earlier:

- Operational Semantics of $\lambda_{\mathrm{ref,conc}}$
  - e, $(h, e) \rightsquigarrow (h, e')$, 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: Hoare Triples
  - $\{P\}\, e \,\{v.Q\}$: Prop

Today:

- Basic Separation Logic: Proving Pointer Programs
- Key Points
  - Reasoning about mutable shared data structures by relating them to mathematical models
  - isList l xs

Derivable Rules

- Last time we saw rules for basic language constructs (expressions that may reduce in one step)
- And the Bind rule for reasoning about more complex expressions.
- It is possible to *derive* other rules that can be used to simplify reasoning.
- Example

$$\frac{\Gamma \mid S \vdash \{P[v/x]\}\, e[v/x]\, \{u.Q[v/x]\}}{\Gamma, x : Val \mid S \vdash \{x = v \land P\}\, e\, \{u.Q\}}\ \textsc{Ht-Pre-Eq}$$

- This rule is derivable, which means that if we assume the antecedent (the formula above the line), then we can use the logical rules (both basic logical entailments and rules for Hoare triples) to prove the conclusion (the formula below the line).
- Several such are mentioned in the exercises. Do those to get familiar with how the logic works.

Example: let expressions

- Let expressions $\mathsf{let}\, x = e \,\mathsf{in}\, e_2$ are definable in $\lambda_{\text{ref,conc}}$:

$$\begin{aligned} \mathsf{let}\, x = e \,\mathsf{in}\, e_2 &\equiv (\lambda x.\, e_2)\, e \\ &\equiv (\mathsf{rec}\, f(x) = e_2)\, e \end{aligned}$$

- Derivable rule

$$\frac{S \vdash \{P\}\, e_1\, \{x.x = v \wedge Q\} \qquad S \vdash \{Q[v/x]\}\, e_2\, [v/x]\ \{u.R\}}{S \vdash \{P\}\, \mathsf{let}\, x = e_1 \,\mathsf{in}\, e_2\, \{u.R\}}\ \textsc{Ht-let-det}$$

Sequencing

- Sequencing $e1; e_2$ is also definable (using let).
- Derivable rule

$$\textsc{Ht-seq}\quad \frac{S \vdash \{P\}\, e_1\, \{v.Q\} \qquad S \vdash \{\exists x.\, Q\}\, e_2\, \{u.R\}}{S \vdash \{P\}\, e_1; e_2\, \{u.R\}}$$

$$\frac{S \vdash \{P\}\, e_1\, \{\_.Q\} \qquad S \vdash \{Q\}\, e_2\, \{u.R\}}{S \vdash \{P\}\, e_1; e_2\, \{u.R\}}$$

where $\_.Q$ means that Q does not mention the return value.

Application and Deterministic Bind

$$\frac{\textsc{Ht-beta} \quad S \vdash \{P\}\, e\,[v/x]\, \{u.Q\}}{S \vdash \{P\}\, (\lambda x.e) v\, \{u.Q\}}$$

$$\frac{\textsc{Ht-bind-det} \quad E \text{ is an eval. context} \qquad S \vdash \{P\}\, e\, \{x.x = u \land Q\} \qquad S \vdash \{Q[u/x]\}\, E[u]\, \{w.\, R\}}{S \vdash \{P\}\, E[e]\, \{w.\, R\}}$$

Example: swap (finally!)

Implementation

$$\mathsf{swap} = \lambda x\ y.\mathsf{let}\ z = !\, x\ \mathsf{in}\ x \leftarrow !\, y; y \leftarrow z$$

Possible Specification

$$\{\ell_1 \hookrightarrow v_1 * \ell_2 \hookrightarrow v_2\}\ \mathsf{swap}\ \ell_1\, \ell_2\ \{v.v = () \land \ell_1 \hookrightarrow v_2 * \ell_2 \hookrightarrow v_1\}.$$

Proof:

- By Rule $\textsc{Ht-let-det}$ SFTS the following two triples

$$\{\ell_1 \hookrightarrow v_1 * \ell_2 \hookrightarrow v_2\}\ !\,\ell_1\ \{v.v = v_1 \land \ell_1 \hookrightarrow v_1 * \ell_2 \hookrightarrow v_2\}$$

$$\{\ell_1 \hookrightarrow v_1 * \ell_2 \hookrightarrow v_2\}\ \ell_1 \leftarrow !\ell_2; \ell_2 \leftarrow v_1\ \{v.v = () \land \ell_1 \hookrightarrow v_2 * \ell_2 \hookrightarrow v_1\}$$

Example: swap

Consider the first triple:

$$\{\ell_1 \hookrightarrow v_1 * \ell_2 \hookrightarrow v_2\} \, ! \, \ell_1 \, \{v. v = v_1 \wedge \ell_1 \hookrightarrow v_1 * \ell_2 \hookrightarrow v_2\}$$

- To show this, use Ht-load and Ht-frame.

Example: swap

Consider the second triple:

$$\{\ell_1 \hookrightarrow v_1 * \ell_2 \hookrightarrow v_2\}\, \ell_1 \leftarrow !\ell_2; \ell_2 \leftarrow v_1 \,\{v.v = () \wedge \ell_1 \hookrightarrow v_2 * \ell_2 \hookrightarrow v_1\}$$

- By sequencing rule $\textsc{Ht-seq}$, SFTS:
  - $\{\ell_1 \hookrightarrow v_1 * \ell_2 \hookrightarrow v_2\}\, \ell_1 \leftarrow !\,\ell_2 \,\{\_.\ell_1 \hookrightarrow v_2 * \ell_2 \hookrightarrow v_2\}$
  - $\{\ell_1 \hookrightarrow v_2 * \ell_2 \hookrightarrow v_2\}\, \ell_2 \leftarrow v_1 \,\{v.v = () \wedge \ell_1 \hookrightarrow v_2 * \ell_2 \hookrightarrow v_1\}$

  For the second, recall that $\ell_2 \hookrightarrow v_2$ implies $\ell_2 \hookrightarrow -$. Hence we can use $\textsc{Ht-csq}$ and $\textsc{Ht-store}$ followed by $\textsc{Ht-frame}$. For the first, use deterministic bind, and then proceed with $\textsc{Ht-store}$, etc.

Proof Outlines

In the literature, you will often see *proof outlines*, which sketch how proofs of programs are done. Tricky to make precise, so not used in the Iris Lecture Notes. Possible proof outline for above proof for swap:

$$
\begin{array}{l}
\{\ell_1 \hookrightarrow v_1 \;\ast\; \ell_2 \hookrightarrow v_2\} \\
\mathsf{swap}\,\ell_1\ell_2 \\
\{\ell_1 \hookrightarrow v_1 \;\ast\; \ell_2 \hookrightarrow v_2\} \\
\mathsf{let}\; z = !\,\ell_1 \;\mathsf{in} \\
\{\ell_1 \hookrightarrow v_1 \;\ast\; \ell_2 \hookrightarrow v_2 \ast z = v_1\} \\
\ell_1 \leftarrow !\,\ell_2; \\
\{\ell_1 \hookrightarrow v_2 \;\ast\; \ell_2 \hookrightarrow v_2 \ast z = v_1\} \\
\ell_2 \leftarrow z \\
\{\ell_1 \hookrightarrow v_2 \;\ast\; \ell_2 \hookrightarrow v_1 \ast z = v_1\} \\
\{\ell_1 \hookrightarrow v_2 \;\ast\; \ell_2 \hookrightarrow v_1\}
\end{array}
$$

Mutable Shared Data Structures: Linked Lists

- Key Point: We reason about mutable shared data structures by relating them to mathematical models.
- Today:
  - Mutable Shared Data Structure = linked lists
  - Mathematical Model = sequences (aka functional lists)
  - isList l xs relates value l to sequence xs
  - Defined by induction on xs

$$\begin{aligned}
&\mathsf{isList}\ l\ []\equiv l=\mathsf{inj}_1()\\
&\mathsf{isList}\ l\ (x:xs)\equiv \exists hd,l'.\, l=\mathsf{inj}_2(hd)*hd\hookrightarrow(x,l')*\mathsf{isList}\ l'\ xs
\end{aligned}$$

  - Exercise: Draw the linked list corresponding to mathematical sequence $[1,2,3]$.
  - Why do we use $*$ above ?

Example: inc on linked lists

Incrementing all values in a linked list of integers.

Implementation

$$
\begin{aligned}
\mathsf{rec}\,\mathsf{inc}(l) = {} & \mathsf{match}\ l\ \mathsf{with} \\
& \quad \mathsf{inj}_1\, x_1 \Rightarrow () \\
& \mid \mathsf{inj}_2\, x_2 \Rightarrow \mathsf{let}\ h = \pi_1\ !\, x_2\ \mathsf{in} \\
& \qquad\qquad\quad \mathsf{let}\ t = \pi_2\ !\, x_2\ \mathsf{in} \\
& \qquad\qquad\quad x_2 \leftarrow (h+1, t); \\
& \qquad\qquad\quad \mathsf{inc}\ t \\
& \mathsf{end}
\end{aligned}
$$

Specification

$$\forall xs. \forall l. \{\mathsf{isList}\ l\ \ xs\}\ \mathsf{inc}\ l\ \{v. v = () \wedge \mathsf{isList}\ l\ \ (\mathsf{map}(1+)xs)\}$$

Example: inc on linked lists

Proof

- Proceed by the recursion rule.
- When considering the body of inc, proceed by case analysis of xs.
- Case $xs = []$: TS
  - $\forall l.\{\text{isList}\, l\, []\}$ match l with ... $\{v.v = () \land \text{isList}\, l (\text{map} (1+) [])\}$
- Case $xs = x : xs'$: TS
  - $\forall x, xs'.\forall l.\{\text{isList}\, l (x : xs')\}$ match l with... $\{v.v = () \land \text{isList}\, l (\text{map} (1+) (x : xs'))\}$
- In both cases, we proceed by the match rule (the isList predicate will tell us which branch is chosen).

Case $xs = []$

To show

- $\forall l.\{\text{isList } l []\}\ \texttt{match l with} \ldots \{v.v = () \wedge \text{isList } l (\text{map}(1+)[])\}$

Note: $\text{isList } l [] \equiv l = \texttt{inj}_1 ()$. Thus SFTS

$$\{\text{isList } l []\}\ l\ \{v.v = \texttt{inj}_1\,() * \text{isList } l []\}$$

which we do by $\textsc{Ht-frame}$ and $\textsc{Ht-Pre-Eq}$ followed by $\textsc{Ht-ret}$.

Case $xs = x : xs'$

- To show
  - $\forall x, xs'.\forall l.\{\text{isList}\, l(x : xs')\}\ \textsf{match}\ l\ \textsf{with} \ldots \{v.v = () \land \text{isList}\, l(\text{map}(1+)(x : xs'))\}$
- Note: $\text{isList}\, l(x : xs) \equiv \exists hd, l'.l = \textsf{inj}_2\, hd * hd \hookrightarrow (x, l') * \text{isList}\, l' xs'$. Thus we have

$$\{l = \textsf{inj}_2\, hd * hd \hookrightarrow (x, l') * \text{isList}\, l' xs'\}$$
$$l$$
$$\{r.r = \textsf{inj}_2\, hd * l = \textsf{inj}_2\, hd * hd \hookrightarrow (x, l') * \text{isList}\, l' xs'\}$$

  for some l' and hd, using the rule Ht-exist, the frame rule, the Ht-Pre-Eq rule, and the Ht-ret rule.
- Hence, the second branch will be taken and by the match rule it suffices to verify the body of the second branch.

Case $xs = x : xs'$, continued

- Using $\textsc{Ht-let-det}$ and $\textsc{Ht-Proj}$ repeatedly we quickly prove

$$\{l = \mathsf{inj}_2\, hd * hd \hookrightarrow (x, l') * \mathsf{isList}\, l' xs'\}$$

$$\begin{aligned}
&\mathsf{let}\ h = \pi_1\, !hd\ \mathsf{in}\\
&\mathsf{let}\ t = \pi_2\, !hd\ \mathsf{in}\\
&hd \leftarrow (h + 1, t)
\end{aligned}$$

$$\{l = \mathsf{inj}_2\, hd * hd \hookrightarrow (x + 1, l') * \mathsf{isList}\, l' xs' * t = l' * h = x\}$$

- Now, by $\textsc{Ht-seq}$ and $\textsc{Ht-csq}$, we are left with proving

$$\{l = \mathsf{inj}_2\, hd * hd \hookrightarrow (x + 1, l') * \mathsf{isList}\, t xs'\}$$

$$f\ t$$

$$\{r.r = () * \mathsf{isList}\, l(\mathsf{map}(+1)(x : xs'))\}$$

which follows from assumption (cf. recursion rule) and definition of the isList predicate.