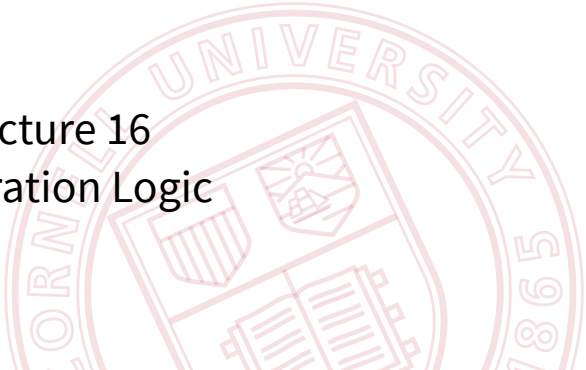
CS 4110
Programming Languages & Logics



Lecture 16
Separation Logic

IMP with Heaps

Let's extend IMP with standard commands for manipulating data on the heap.

Note: for today's lecture I'll write e instead of a for arithmetic expressions.

$$
\begin{array}{rcl}
c & ::= & \ldots \\
& \mid & x := \texttt{new}(e) \\
& \mid & \texttt{free}(e) \\
& \mid & x := *e \\
& \mid & *e_1 := e_2
\end{array}
$$

State = Store + Heap

A *state* s now has two parts: (σ, h) where:

- $\sigma : \text{Var} \rightharpoonup \mathbb{Z}$ is the *store* as before, and
- $h : \text{Addr} \rightharpoonup \mathbb{Z}$ is the *heap*, where an Addr is the type of pointers (and left abstract, though in an implementation would just be an integer).

The big-step semantics for commands becomes:

$$\langle \sigma, h, c \rangle \Downarrow (\sigma', h')$$

Note: the big-step semantics for expressions remains,

$$\langle \sigma, e \rangle \Downarrow v$$

as evaluating expressions does not require access to the heap.

Big-Step Semantics: Basic Commands

$$\frac{}{\langle \sigma, h, \textbf{skip} \rangle \Downarrow (\sigma, h)} \textsc{Skip}$$

Big-Step Semantics: Basic Commands

$$\frac{}{\langle \sigma, h, \textbf{skip} \rangle \Downarrow (\sigma, h)} \textsc{Skip}$$

$$\frac{\langle \sigma, e \rangle \Downarrow v}{\langle \sigma, h, x := e \rangle \Downarrow (\sigma[x \mapsto v], h)} \textsc{Assign}$$

Big-Step Semantics: Basic Commands

$$\frac{}{\langle \sigma, h, \textbf{skip} \rangle \Downarrow (\sigma, h)} \textsc{Skip}$$

$$\frac{\langle \sigma, e \rangle \Downarrow v}{\langle \sigma, h, x := e \rangle \Downarrow (\sigma[x \mapsto v], h)} \textsc{Assign}$$

$$\frac{\langle \sigma, h, c_1 \rangle \Downarrow (\sigma_1, h_1) \qquad \langle \sigma_1, h_1, c_2 \rangle \Downarrow (\sigma_2, h_2)}{\langle \sigma, h, c_1; c_2 \rangle \Downarrow (\sigma_2, h_2)} \textsc{Seq}$$

Semantics: Conditionals and Loops

$$\frac{\langle \sigma, b \rangle \Downarrow \textbf{true} \qquad \langle \sigma, h, c_1 \rangle \Downarrow (\sigma', h')}{\langle \sigma, h, \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \rangle \Downarrow (\sigma', h')} \textsc{If-True}$$

$$\frac{\langle \sigma, b \rangle \Downarrow \textbf{false} \qquad \langle \sigma, h, c_2 \rangle \Downarrow (\sigma', h')}{\langle \sigma, h, \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \rangle \Downarrow (\sigma', h')} \textsc{If-False}$$

Semantics: Conditionals and Loops

$$\frac{\langle \sigma, b \rangle \Downarrow \textbf{true} \qquad \langle \sigma, h, c_1 \rangle \Downarrow (\sigma', h')}{\langle \sigma, h, \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \rangle \Downarrow (\sigma', h')} \textsc{If-True}$$

$$\frac{\langle \sigma, b \rangle \Downarrow \textbf{false} \qquad \langle \sigma, h, c_2 \rangle \Downarrow (\sigma', h')}{\langle \sigma, h, \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \rangle \Downarrow (\sigma', h')} \textsc{If-False}$$

$$\frac{\langle \sigma, b \rangle \Downarrow \textbf{false}}{\langle \textbf{while } b \textbf{ do } c, \rangle \Downarrow (\sigma, h)} \textsc{While-False}$$

$$\frac{\langle \sigma, b \rangle \Downarrow \textbf{true} \qquad \langle \sigma, h, c \rangle \Downarrow (\sigma_1, h_1) \qquad \langle \sigma_1, h_1, \textbf{while } b \textbf{ do } c \rangle \Downarrow (\sigma_2, h_2)}{\langle \sigma, h, \textbf{while } b \textbf{ do } c \rangle \Downarrow (\sigma_2, h_2)} \textsc{While-True}$$

Semantics: Load and Store

$$\frac{\langle \sigma, e \rangle \Downarrow p \qquad p \in dom(h) \qquad h(p) = v}{\langle \sigma, h, x := *e \rangle \Downarrow (\sigma[x \mapsto v], h)} \textsc{Load}$$

Semantics: Load and Store

$$\frac{\langle \sigma, e \rangle \Downarrow p \qquad p \in dom(h) \qquad h(p) = v}{\langle \sigma, h, x := *e \rangle \Downarrow (\sigma[x \mapsto v], h)} \textsc{Load}$$

$$\frac{\langle \sigma, e_1 \rangle \Downarrow p \qquad \langle \sigma, e_2 \rangle \Downarrow v \qquad p \in dom(h)}{\langle \sigma, h, *e_1 := e_2 \rangle \Downarrow (\sigma,\ h[p \mapsto v])} \textsc{Store}$$

Semantics: New and Free

$$\frac{\langle \sigma, e \rangle \Downarrow v \qquad p \notin dom(h)}{\langle \sigma, h, x := \textbf{new}(e) \rangle \Downarrow (\sigma[x \mapsto p],\ h[p \mapsto v])} \textsc{New}$$

Semantics: New and Free

$$\frac{\langle \sigma, e \rangle \Downarrow v \qquad p \notin dom(h)}{\langle \sigma, h, x := \textbf{new}(e) \rangle \Downarrow (\sigma[x \mapsto p],\ h[p \mapsto v])} \textsc{New}$$

$$\frac{\langle \sigma, e \rangle \Downarrow p \qquad p \in dom(h)}{\langle \sigma, h, \textbf{free}(e) \rangle \Downarrow (\sigma,\ h \setminus \{p\})} \textsc{Free}$$

Rule of Constancy

In standard Hoare logic, the following rule is admissible:

$$\frac{\{P\}\ c\ \{Q\} \quad \mathrm{fv}(R) \cap \mathrm{mod}(c) = \emptyset}{\{P \wedge R\}\ c\ \{Q \wedge R\}}\ \textsc{Const}$$

where

- $\mathrm{fv}(R)$ is the free variables of R.
- $\mathrm{mod}(c)$ is the set of variables that c may modify.

Free Variables

The function $fv(P)$ returns the set of program variables that occur free in an assertion P.

$$
\begin{aligned}
fv(a_1 \leq a_2) &= fv(a_1) \cup fv(a_2) \\
fv(P \wedge Q) &= fv(P) \cup fv(Q) \\
fv(P \vee Q) &= fv(P) \cup fv(Q) \\
fv(P \implies Q) &= fv(P) \cup fv(Q) \\
fv(\neg P) &= fv(P) \\
fv(\forall i.\, P) &= fv(P) \setminus \{i\} \\
fv(\exists i.\, P) &= fv(P) \setminus \{i\}
\end{aligned}
$$

Modified Variables

$$
\begin{aligned}
mod(\textbf{skip}) &= \emptyset \\
mod(x := e) &= \{x\} \\
mod(c_1; c_2) &= mod(c_1) \cup mod(c_2) \\
mod(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2) &= mod(c_1) \cup mod(c_2) \\
mod(\textbf{while } b \textbf{ do } c) &= mod(c) \\
mod(x := *e) &= \{x\} \\
mod(*e_1 := e_2) &= \emptyset \\
mod(x := \textbf{new}(e)) &= \{x\} \\
mod(\textbf{free}(e)) &= \emptyset
\end{aligned}
$$

Example: Constancy

Consider the Hoare triple:

$$\{ x > 0 \}\ x := x + 1\ \{ x > 1 \}$$

With CONSTANCY, we can strengthen pre- and post-conditions with an assertion:

$$\{ x > 0 \wedge y = 0 \}\ x := x + 1\ \{ x > 1 \wedge y = 0 \}$$

Example: Constancy

Consider the Hoare triple:

$$\{\, x > 0 \,\}\; x := x + 1 \;\{\, x > 1 \,\}$$

With CONSTANCY, we can strengthen pre- and post-conditions with an assertion:

$$\{\, x > 0 \wedge y = 0 \,\}\; x := x + 1 \;\{\, x > 1 \wedge y = 0 \,\}$$

Why is this allowed?

$$fv(y = 0) = \{y\} \qquad mod(x := x + 1) = \{x\}$$

As $fv(y = 0) \cap mod(x := x + 1) = \emptyset$, the CONSTANCY rule applies.

Example: Constancy

Now consider:

$$\{ x > 0 \}\ y := y + 1\ \{ x > 0 \}$$

Suppose we try to add $y = 0$ as a constant assertion:

$$\{ x > 0 \wedge y = 0 \}\ y := y + 1\ \{ x > 0 \wedge y = 0 \}$$

Example: Constancy

Now consider:

$$\{\, x > 0 \,\}\ y := y + 1\ \{\, x > 0 \,\}$$

Suppose we try to add $y = 0$ as a constant assertion:

$$\{\, x > 0 \wedge y = 0 \,\}\ y := y + 1\ \{\, x > 0 \wedge y = 0 \,\}$$

Here,

$$\mathit{fv}(y = 0) = \{y\} \qquad \mathit{mod}(y := y + 1) = \{y\}$$

So $\mathit{fv}(y = 0) \cap \mathit{mod}(y := y + 1) \neq \emptyset$. Indeed, the triple is invalid: the assignment breaks the constant property $y = 0$.

Heap Assertions in Separation Logic

Now let's add some assertions for describing the heap.

$$H, J, K ::= \mathbf{emp} \mid [P] \mid e_1 \hookrightarrow e_2 \mid H_1 \star H_2 \mid H_1 \mathbin{-\!\!\star} H_2 \mid H_1 \mathbin{\mathbb{W}} H_2 \mid \forall x.\, H \mid \exists x.\, H$$

Heap Assertions in Separation Logic

Now let's add some assertions for describing the heap.

$$H, J, K ::= \mathbf{emp} \mid [P] \mid e_1 \hookrightarrow e_2 \mid H_1 \star H_2 \mid H_1 \mathbin{⫽} H_2 \mid H_1 \mathbin{⅋} H_2 \mid \forall x.\, H \mid \exists x.\, H$$

Intuitively:

- **emp** : heap is empty,
- $[P]$: heap is empty and P holds,
- $e_1 \hookrightarrow e_2$: heap consists of exactly one cell with pointer e_1 and value e_2,
- $H_1 \star H_2$: heap can be split into two disjoint pieces, one satisfying H_1 and the other satisfying H_2,
- $H_1 \mathbin{⫽} H_2$ and $H_1 \mathbin{⅋} H_2$ are heap assertion versions of the standard boolean connectives, and
- $\forall x.\, H$ and $\exists x.\, H$: are heap assertion versions of the standard quantifiers.

Heap Assertions in Separation Logic

Now let's add some assertions for describing the heap.

$$H, J, K ::= \mathbf{emp} \mid [P] \mid e_1 \hookrightarrow e_2 \mid H_1 \star H_2 \mid H_1 \mathbin{-\!\!\!\ast} H_2 \mid H_1 \mathbin{\rotatebox{180}{$-\!\!\!\ast$}} H_2 \mid \forall x.\, H \mid \exists x.\, H$$

Formally:

$$\begin{array}{lll}
(\sigma, h) \models_I \mathbf{emp} & \text{if} & h = \emptyset \\
(\sigma, h) \models_I [P] & \text{if} & h = \emptyset \wedge \sigma \models_I P \\
(\sigma, h) \models_I e_1 \hookrightarrow e_2 & \text{if} & h = \{(p, v)\} \text{ where } \langle \sigma, e_1 \rangle_I \Downarrow p \text{ and } \langle \sigma, e_2 \rangle_I \Downarrow v \\
(\sigma, h) \models_I H_1 \star H_2 & \text{if} & \exists h_1, h_2.\, h = h_1 \uplus h_2 \wedge (\sigma, h_1) \models_I P_1 \wedge (\sigma, h_2) \models_I P_2 \\
(\sigma, h) \models_I H_1 \mathbin{-\!\!\!\ast} H_2 & \text{if} & (\sigma, h) \models_I H_1 \text{ and } (\sigma, h) \models_I H_2 \\
(\sigma, h) \models_I H_1 \mathbin{\rotatebox{180}{$-\!\!\!\ast$}} H_2 & \text{if} & (\sigma, h) \models_I H_1 \text{ or } (\sigma, h) \models_I H_2 \\
(\sigma, h) \models_I \forall x.\, H & \text{if} & (\sigma, h) \models_{I[x \mapsto v]} H \text{ for all } v \\
(\sigma, h) \models_I \exists x.\, H & \text{if} & (\sigma, h) \models_{I[x \mapsto v]} H \text{ for some } v
\end{array}$$

Constancy and Aliasing

Abbreviation: write $x \hookrightarrow -$ as shorthand for $\exists v.\, x \hookrightarrow v$

Constancy and Aliasing

Abbreviation: write $x \hookrightarrow -$ as shorthand for $\exists v.\, x \hookrightarrow v$

Consider the following application of CONSTANCY

$$\frac{}{\{x \hookrightarrow -\}\ *x := 4\ \{x \hookrightarrow 4\}}$$

Constancy and Aliasing

Abbreviation: write $x \hookrightarrow -$ as shorthand for $\exists v.\, x \hookrightarrow v$

Consider the following application of CONSTANCY

$$\frac{\{x \hookrightarrow -\}\ *x := 4\ \{x \hookrightarrow 4\}}{\{x \hookrightarrow - \wedge y \hookrightarrow 3\}\ *x := 4\ \{x \hookrightarrow 4 \wedge y \hookrightarrow 3\}}$$

Constancy and Aliasing

Abbreviation: write $x \hookrightarrow -$ as shorthand for $\exists v.\, x \hookrightarrow v$

Consider the following application of CONSTANCY

$$\frac{\{x \hookrightarrow -\}\ *x := 4\ \{x \hookrightarrow 4\}}{\{x \hookrightarrow - \wedge y \hookrightarrow 3\}\ *x := 4\ \{x \hookrightarrow 4 \wedge y \hookrightarrow 3\}}$$

We have

$$fv(y \hookrightarrow 3) = \{y\} \qquad mod(x := 4) = \{x\}$$

Problem: but what if x and y are aliases that point to the same heap location?!

Separation Logic

We want something like the CONSTANCY rule that supports local reasoning involving the heap.

This is exactly what the FRAME rule does:

$$\frac{\vdash \{H\}\ c\ \{J\} \quad fvs(K) \cap \quad \text{mod } c = \emptyset}{\vdash \{H \star K\}\ c\ \{J \star K\}}\ \textsc{Frame}$$

Separation Logic Triples

Definition (Hoare Triple (Total Correctness))

A Hoare logic triple is valid, written $\models_{\text{Hoare}} \{H\}\ c\ \{J\}$, if for all σ, h, and I, if $\sigma, h \models_I H$ then $\langle \sigma, h, c \rangle \Downarrow (\sigma', h')$ and $\sigma', h' \models_I J$

Separation Logic Triples

Definition (Hoare Triple (Total Correctness))

A Hoare logic triple is valid, written $\models_{\text{Hoare}} \{H\}\ c\ \{J\}$, if for all σ, h, and I, if $\sigma, h \models_I H$ then $\langle \sigma, h, c \rangle \Downarrow (\sigma', h')$ and $\sigma', h' \models_I J$

Definition (Separation Logic Triple (Total Correctness))

A separation logic triple is valid, written $\models \{H\}\ c\ \{J\}$, if for all K we have $\models_{\text{Hoare}} \{H \star K\}\ c\ \{J \star K\}$

Separation Logic Triples: Alternate Definition

Write $h_1 \perp h_2$ to mean that h_1 and h_2 are disjoint heaps.

Definition (Separation Logic Triple (Total Correctness))

A Hoare logic triple is valid, written $\models_{\text{Hoare}} \{H\}\ c\ \{J\}$, if for all σ, h_1, h_2 and I, if $\sigma, h_1 \models_I H$ and $h_1 \perp h_2$ then $\langle \sigma, h_1 \uplus h_2, c \rangle \Downarrow (\sigma', h_1' \uplus h_2)$ and $\sigma', h_1' \models_I J$

Separation Logic Triples: Alternate Definition

It should be clear that the FRAME rule is sound...

$$\frac{\vdash \{H\}\ c\ \{J\} \quad fvs(K) \cap \quad \text{mod } c = \emptyset}{\vdash \{H \star K\}\ c\ \{J \star K\}}\ \textsc{Frame}$$

... as the notion of correctness "bakes it in"

Definition (Separation Logic Triple (Total Correctness))

A separation logic triple is valid, written $\models \{H\}\ c\ \{J\}$, if for all K we have $\models_{\text{Hoare}} \{H \star K\}\ c\ \{J \star K\}$

Small Axioms for Heap Commands

$$\frac{}{\{\, e \hookrightarrow e' \,\}\ x := *e\ \{\, [x = e'] \star e \hookrightarrow e' \,\}}\ \textsc{Load}$$

Small Axioms for Heap Commands

$$\frac{}{\{\, e \hookrightarrow e' \,\}\ x := *e\ \{\, [x = e'] \star e \hookrightarrow e' \,\}}\ \textsc{Load}$$

$$\frac{}{\{\, x \hookrightarrow - \,\}\ *x := e\ \{\, x \hookrightarrow e \,\}}\ \textsc{Store}$$

Small Axioms for Heap Commands

$$\frac{}{\{\, e \hookrightarrow e' \,\}\ x := *e\ \{\, [x = e'] \star e \hookrightarrow e' \,\}}\ \textsc{Load}$$

$$\frac{}{\{\, x \hookrightarrow - \,\}\ *x := e\ \{\, x \hookrightarrow e \,\}}\ \textsc{Store}$$

$$\frac{}{\{\, \mathbf{emp} \,\}\ x := \mathbf{new}(e)\ \{\, [x = p] \star p \hookrightarrow e \,\}}\ \textsc{Alloc}$$

Small Axioms for Heap Commands

$$\frac{}{\{\, e \hookrightarrow e' \,\}\ x := *e\ \{\, [x = e'] \star e \hookrightarrow e' \,\}}\ \textsc{Load}$$

$$\frac{}{\{\, x \hookrightarrow - \,\}\ *x := e\ \{\, x \hookrightarrow e \,\}}\ \textsc{Store}$$

$$\frac{}{\{\, \mathbf{emp} \,\}\ x := \mathbf{new}(e)\ \{\, [x = p] \star p \hookrightarrow e \,\}}\ \textsc{Alloc}$$

$$\frac{}{\{\, e \hookrightarrow - \,\}\ \mathbf{free}(e)\ \{\, \mathbf{emp} \,\}}\ \textsc{Free}$$