Introduction to Separation Logic

Extending Imp with Memory Accesses

Syntax:

$$
\begin{array}{rcl}
(comm)\ c & ::= & \ldots \\
& | & x := [e] \\
& | & [e] := e' \\
& | & x := \mathbf{cons}(e_1, e_2) \\
& | & \mathbf{dispose}(e) \\
& | & \ldots
\end{array}
$$

Program States

$$
\begin{array}{llll}
(store) & s & \in & Var \to \mathbf{Z} \\
(heap) & h & \in & \mathbf{N} \rightharpoonup_{\mathrm{fin}} \mathbf{Z} \\
(state) & \sigma & ::= & (s, h)
\end{array}
$$

		Store : x: 3, y: 4
		Heap : empty
Allocation	$x := \mathbf{cons}(1, 2)$;	$\Downarrow$
		Store : x: 37, y: 4
		Heap : 37: 1, 38: 2
Lookup	$y := [x]$;	$\Downarrow$
		Store : x: 37, y: 1
		Heap : 37: 1, 38: 2
Mutation	$[x + 1] := 3$;	$\Downarrow$
		Store : x: 37, y: 1
		Heap : 37: 1, 38: 3
Deallocation	$\mathbf{dispose}(x + 1)$	$\Downarrow$
		Store : x: 37, y: 1
		Heap : 37: 1

Note that expressions depend only on the store

Memory Faults

		Store : $x: 3,\ y: 4$
		Heap : empty
Allocation	$x := \mathbf{cons}(1, 2)\ ;$	$\Downarrow$
		Store : $x: 37,\ y: 4$
		Heap : $37: 1,\ 38: 2$
Lookup	$y := [x]\ ;$	$\Downarrow$
		Store : $x: 37,\ y: 1$
		Heap : $37: 1,\ 38: 2$
Mutation	$[x + 2] := 3\ ;$	$\Downarrow$
		abort

Faults can also be caused by out-of-range lookup or deallocation.

Operational Semantics

$$\frac{h(\llbracket e \rrbracket_{intexp}\, s) = n}{(x := [e], (s, h)) \longrightarrow (\textbf{skip}, (s\{x \rightsquigarrow n\}, h))}$$

$$\frac{\llbracket e \rrbracket_{intexp}\, s \notin \mathrm{dom}(h)}{(x := [e], (s, h)) \longrightarrow \textbf{abort}}$$

$$\frac{\llbracket e \rrbracket_{intexp}\, s = \ell \quad \ell \in \mathrm{dom}(h)}{([e] := e', (s, h)) \longrightarrow (\textbf{skip}, (s, h\{\ell \rightsquigarrow \llbracket e' \rrbracket_{intexp}\, s\}))}$$

$$\frac{\llbracket e \rrbracket_{intexp}\, s \notin \mathrm{dom}(h)}{([e] := e', (s, h)) \longrightarrow \textbf{abort}}$$

$$\frac{\llbracket e_1 \rrbracket_{intexp}\, s = n_1 \quad \llbracket e_2 \rrbracket_{intexp}\, s = n_2 \quad \{\ell, \ell + 1\} \cap \mathrm{dom}(h) = \emptyset}{(x := \textbf{cons}(e_1, e_2), (s, h)) \longrightarrow (\textbf{skip}, (s\{x \rightsquigarrow \ell\}, h\{\ell \rightsquigarrow n_1, \ell+1 \rightsquigarrow n_2\}))}$$

Assertions

Standard predicate logic assertions, plus

- $\mathbf{emp}$ (empty heap)
  The heap is empty.
- $e \mapsto e'$ (singleton heap)
  The heap contains one cell, at address e with contents e'.
- $p_1 \ * \ p_2$ (separating conjunction)
  The heap can be split into two disjoint parts such that p_1 holds for one part and p_2 holds for the other.
- $p_1 \ -\!\!* \ p_2$ (separating implication)
  If the heap is extended with a disjoint part in which p_1 holds, then p_2 holds for the extended heap.

Some Abbreviations

$$e \mapsto - \stackrel{\text{def}}{=} \exists x'.\, e \mapsto x' \quad \text{where } x' \text{ not free in } e$$

$$e \hookrightarrow e' \stackrel{\text{def}}{=} e \mapsto e' * \mathbf{true}$$

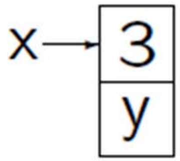
$$e \mapsto e_1, \ldots, e_n \stackrel{\text{def}}{=} e \mapsto e_1 * \cdots * e + n - 1 \mapsto e_n$$

$$e \hookrightarrow e_1, \ldots, e_n \stackrel{\text{def}}{=} e \hookrightarrow e_1 * \cdots * e + n - 1 \hookrightarrow e_n$$

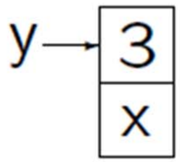
$$\text{iff } e \mapsto e_1, \ldots, e_n * \mathbf{true}$$

Examples of Separating Conjunction

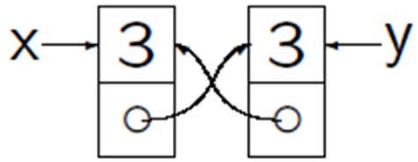
1. $x \mapsto 3, y$ asserts that x points to an adjacent pair of cells containing 3 and y.



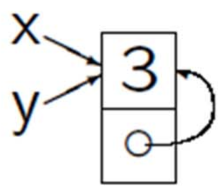
2. $y \mapsto 3, x$ asserts that y points to an adjacent pair of cells containing 3 and x.



3. $x \mapsto 3, y \; * \; y \mapsto 3, x$ asserts that situations (1) and (2) hold for separate parts of the heap.



4. $x \mapsto 3, y \wedge y \mapsto 3, x$ asserts that situations (1) and (2) hold for the same heap, which can only happen if the values of x and y are the same.



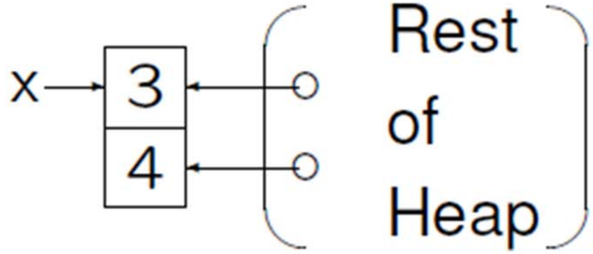
5. $x \hookrightarrow 3, y \wedge y \hookrightarrow 3, x$ asserts that either (3) or (4) may hold, and that the heap may contain additional cells.

An Example of Separating Implication

Suppose p holds for

Store : x: $\alpha, \ldots$

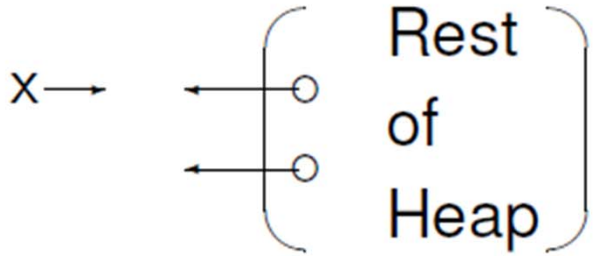
Heap : $\alpha\colon 3, \alpha + 1\colon 4, \ldots$



Then $(\mathsf{x} \mapsto 3, 4) \mathrel{-\!\!*} p$ holds for

Store : x: $\alpha, \ldots$

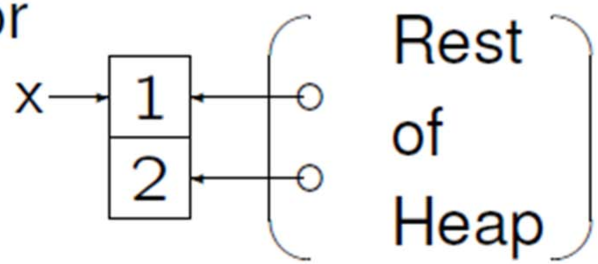
Heap : $\ldots$



and $\mathsf{x} \mapsto 1, 2 \; * \; ((\mathsf{x} \mapsto 3, 4) \mathrel{-\!\!*} p)$ holds for

Store : x: $\alpha, \ldots$

Heap : $\alpha\colon 1, \alpha + 1\colon 2, \ldots$



In particular,

$$\{\mathsf{x} \mapsto 1, 2 \; * \; ((\mathsf{x} \mapsto 3, 4) \mathrel{-\!\!*} p)\} \; [\mathsf{x}] := 3 \, ; [\mathsf{x} + 1] := 4 \; \{p\},$$

and more generally,

$$\{\mathsf{x} \mapsto -, - \; * \; ((\mathsf{x} \mapsto 3, 4) \mathrel{-\!\!*} p)\} \; [\mathsf{x}] := 3 \, ; [\mathsf{x} + 1] := 4 \; \{p\}.$$

The Meaning of Assertions

When s is a store, h is a heap, and p is an assertion whose free variables belong to the domain of s, we write

$$s, h \vDash p$$

to indicate that the state s, h *satisfies* p, or p *is true in* s, h, or p *holds in* s, h. Then:

$$s, h \vDash b \text{ iff } [\![b]\!]_{\mathsf{boolexp}} s = \mathbf{true},$$

$$s, h \vDash \neg\, p \text{ iff } s, h \vDash p \text{ is false},$$

$$s, h \vDash p_0 \wedge p_1 \text{ iff } s, h \vDash p_0 \text{ and } s, h \vDash p_1$$

(and similarly for $\vee$, $\Rightarrow$, $\Leftrightarrow$),

$$
\begin{aligned}
s, h \vDash \forall v.\ p \ &\text{iff}\ \forall x \in \mathbf{Z}.\ [\, s \mid v\colon x \,], h \vDash p, \\
s, h \vDash \exists v.\ p \ &\text{iff}\ \exists x \in \mathbf{Z}.\ [\, s \mid v\colon x \,], h \vDash p, \\
s, h \vDash \mathbf{emp} \ &\text{iff}\ \mathsf{dom}\, h = \{\}, \\
s, h \vDash e \mapsto e' \ &\text{iff}\ \mathsf{dom}\, h = \{[\![e]\!]_{\mathsf{exp}} s\} \text{ and} \\
&\qquad h([\![e]\!]_{\mathsf{exp}} s) = [\![e']\!]_{\mathsf{exp}} s, \\
s, h \vDash p_0 * p_1 \ &\text{iff}\ \exists h_0, h_1.\ h_0 \perp h_1 \text{ and } h_0 \cdot h_1 = h \text{ and} \\
&\qquad s, h_0 \vDash p_0 \text{ and } s, h_1 \vDash p_1, \\
s, h \vDash p_0 \mathrel{-\!\!*} p_1 \ &\text{iff}\ \forall h'.\ (h' \perp h \text{ and } s, h' \vDash p_0) \text{ implies} \\
&\qquad s, h \cdot h' \vDash p_1.
\end{aligned}
$$

When $s, h \vDash p$ holds for all states s, h (such that the domain of s contains the free variables of p), we say that p is *valid*.

When $s, h \vDash p$ holds for some state s, h, we say that p is *satisfiable*.

Examples

$$
\begin{aligned}
s, h \vDash \mathsf{x} \mapsto \mathsf{y} &\text{ iff } \operatorname{dom} h = \{s\,\mathsf{x}\} \text{ and } h(s\,\mathsf{x}) = s\,\mathsf{y} \\
s, h \vDash \mathsf{x} \mapsto - &\text{ iff } \operatorname{dom} h = \{s\,\mathsf{x}\} \\
s, h \vDash \mathsf{x} \hookrightarrow \mathsf{y} &\text{ iff } s\,\mathsf{x} \in \operatorname{dom} h \text{ and } h(s\,\mathsf{x}) = s\,\mathsf{y} \\
s, h \vDash \mathsf{x} \hookrightarrow - &\text{ iff } s\,\mathsf{x} \in \operatorname{dom} h \\
s, h \vDash \mathsf{x} \mapsto \mathsf{y}, \mathsf{z} &\text{ iff } h = [\, s\,\mathsf{x} \colon s\,\mathsf{y} \mid s\,\mathsf{x} + 1 \colon s\,\mathsf{z} \,] \\
s, h \vDash \mathsf{x} \mapsto -, - &\text{ iff } \operatorname{dom} h = \{s\,\mathsf{x}, s\,\mathsf{x} + 1\} \\
s, h \vDash \mathsf{x} \hookrightarrow \mathsf{y}, \mathsf{z} &\text{ iff } h \supseteq [\, s\,\mathsf{x} \colon s\,\mathsf{y} \mid s\,\mathsf{x} + 1 \colon s\,\mathsf{z} \,] \\
s, h \vDash \mathsf{x} \hookrightarrow -, - &\text{ iff } \operatorname{dom} h \supseteq \{s\,\mathsf{x}, s\,\mathsf{x} + 1\}.
\end{aligned}
$$

Inference Rules for $*$ and $-\!\!*$

$$p_0 * p_1 \Leftrightarrow p_1 * p_0$$

$$(p_0 * p_1) * p_2 \Leftrightarrow p_0 * (p_1 * p_2)$$

$$p * \mathbf{emp} \Leftrightarrow p$$

$$(p_0 \vee p_1) * q \Leftrightarrow (p_0 * q) \vee (p_1 * q)$$

$$(p_0 \wedge p_1) * q \Rightarrow (p_0 * q) \wedge (p_1 * q)$$

$$(\exists x.\, p_0) * p_1 \Leftrightarrow \exists x.\, (p_0 * p_1) \quad \text{when } x \text{ not free in } p_1$$

$$(\forall x.\, p_0) * p_1 \Rightarrow \forall x.\, (p_0 * p_1) \quad \text{when } x \text{ not free in } p_1$$

$$\frac{p_0 \Rightarrow p_1 \qquad q_0 \Rightarrow q_1}{p_0 * q_0 \Rightarrow p_1 * q_1} \quad \text{(monotonicity)}$$

$$\frac{p_0 * p_1 \Rightarrow p_2}{p_0 \Rightarrow (p_1 -\!\!* p_2)} \quad \text{(currying)} \qquad \frac{p_0 \Rightarrow (p_1 -\!\!* p_2)}{p_0 * p_1 \Rightarrow p_2.} \quad \text{(decurrying)}$$

Some Axiom Schemata for $\mapsto$ and $\hookrightarrow$

$$e_0 \mapsto e'_0 \wedge e_1 \mapsto e'_1 \Leftrightarrow e_0 \mapsto e'_0 \wedge e_0 = e_1 \wedge e'_0 = e'_1$$

$$e_0 \hookrightarrow e'_0 \ast e_1 \hookrightarrow e'_1 \Rightarrow e_0 \neq e_1$$

$$\mathbf{emp} \Leftrightarrow \forall x.\ \neg(x \hookrightarrow -)$$

$$(e \hookrightarrow e') \wedge p \Rightarrow (e \mapsto e') \ast ((e \mapsto e') \mathrel{-\!\!\ast} p).$$

Two Unsound Axiom Schemata

$$p \Rightarrow p \ast p$$ (Contraction — unsound)

e.g. $p : \mathsf{x} \mapsto 1$

$$p \ast q \Rightarrow p$$ (Weakening — unsound)

e.g. $p : \mathsf{x} \mapsto 1$

$q : \mathsf{y} \mapsto 2$