

CLASS NOTES FOR CS 818A3 - SPRING 2005

AN INTRODUCTION TO SEPARATION LOGIC

3. Basic Separation Logic

John C. Reynolds
Department of Computer Science
Carnegie Mellon University

Revised February 9, 2005

©2005 John C. Reynolds

Moving to Separation Logic

The central change from Hoare logic to separation logic is that, instead of the *state* of the computation being simply a *store*, which map variables into values, the state is now a pair consisting of a store and a *heap*, which maps addresses into values (or record-like sequences of values).

We will introduce new commands for manipulating the heap, and new assertions for describing it.

Expressions however, will remain unchanged. In particular, their meaning will depend upon the store, but not the heap.

Thus the syntax and semantics of expressions ($\langle \text{exp} \rangle$) and boolean expressions ($\langle \text{boolexp} \rangle$) remain unchanged.

Moreover, since separation logic introduces no new binders (except for the iterated separating conjunction, which will be introduced later), the definition of free variables, substitution (both total and partial), and the substitution theorems (Propositions 1, 4, 5) remain essentially the same.

States

Without address arithmetic (old version):

$$\text{Values} = \text{Integers} \cup \text{Atoms} \cup \text{Addresses}$$

where Integers, Atoms, and Addresses are disjoint

$$\mathbf{nil} \in \text{Atoms}$$

$$\text{Stores}_V = V \rightarrow \text{Values}$$

$$\text{Heaps} = \bigcup_{\substack{\text{fn} \\ A \subseteq \text{Addresses}}} (A \rightarrow \text{Values}^+)$$

$$\text{States}_V = \text{Stores}_V \times \text{Heaps}$$

where V is a finite set of variables.

With address arithmetic (new version):

$$\text{Values} = \text{Integers}$$

$$\text{Atoms} \cup \text{Addresses} \subseteq \text{Integers}$$

where Atoms and Addresses are disjoint

$$\mathbf{nil} \in \text{Atoms}$$

$$\text{Stores}_V = V \rightarrow \text{Values}$$

$$\text{Heaps} = \bigcup_{\substack{\text{fn} \\ A \subseteq \text{Addresses}}} (A \rightarrow \text{Values})$$

$$\text{States}_V = \text{Stores}_V \times \text{Heaps}$$

where V is a finite set of variables.

(We assume that all but a finite number of nonnegative integers are addresses.)

Syntax of Additional Commands

$$\begin{aligned}
 \langle \text{comm} \rangle ::= & \dots \\
 & | \langle \text{var} \rangle := \mathbf{cons}(\langle \text{exp} \rangle, \dots, \langle \text{exp} \rangle) \\
 & | \langle \text{var} \rangle := [\langle \text{exp} \rangle] \\
 & | [\langle \text{exp} \rangle] := \langle \text{exp} \rangle \\
 & | \mathbf{dispose} \langle \text{exp} \rangle
 \end{aligned}$$

Compositional Large-Step Semantics of Commands

$$\llbracket c \in \langle \text{comm} \rangle \rrbracket_{\text{comm}} \subseteq \bigcup_{\substack{\text{fin} \\ V \supseteq \text{FV}(c)}} \text{States}_V \times (\text{States}_V \cup \{\mathbf{abort}, \perp\})$$

is the relation such that

- $(s, h) \llbracket c \rrbracket_{\text{comm}} (s', h')$ iff, starting in state (s, h) , there is a finite execution of c that terminates in state (s', h') .
- $(s, h) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort}$ iff, starting in state (s, h) , there is a finite execution of c that terminates with **abort**.
- $(s, h) \llbracket c \rrbracket_{\text{comm}} \perp$ iff, starting in state (s, h) , there is a non-terminating execution of c .

Note that execution preserves the domain of the store, i.e., if $(s, h) \llbracket c \rrbracket_{\text{comm}} (s', h')$, then $\text{dom } s = \text{dom } s'$.

Note that we write $[f \mid x: y]$ to denote the function such that

$$\text{dom}[f \mid x: y] = \text{dom } f \cup \{x\}$$

$$[f \mid x: y](x) = y \quad \text{and} \quad [f \mid x: y](x') = f(x') \text{ when } x' \neq x.$$

Inference Rules for the Meanings of Commands

- Assignment

$$\frac{}{(s, h) \llbracket v := e \rrbracket_{\text{comm}} ([s \mid v: \llbracket e \rrbracket_{\text{exp}} s], h),}$$

when $\{v\} \cup \text{FV}(e) \subseteq \text{dom } s$.

- **skip**

$$\frac{}{(s, h) \llbracket \mathbf{skip} \rrbracket_{\text{comm}} (s, h).$$

- Sequencing

$$\frac{(s, h) \llbracket c_0 \rrbracket_{\text{comm}} (s', h') \quad (s', h') \llbracket c_1 \rrbracket_{\text{comm}} \tau''}{(s, h) \llbracket c_0 ; c_1 \rrbracket_{\text{comm}} \tau''}$$

$$\frac{(s, h) \llbracket c_0 \rrbracket_{\text{comm}} \mathbf{abort}}{(s, h) \llbracket c_0 ; c_1 \rrbracket_{\text{comm}} \mathbf{abort}} \quad \frac{(s, h) \llbracket c_0 \rrbracket_{\text{comm}} \perp}{(s, h) \llbracket c_0 ; c_1 \rrbracket_{\text{comm}} \perp}.$$

- Conditional

$$\frac{\llbracket b \rrbracket_{\text{boolexp}} s = \mathbf{true} \quad (s, h) \llbracket c_0 \rrbracket_{\text{comm}} \tau'}{(s, h) \llbracket \mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1 \rrbracket_{\text{comm}} \tau'}$$

$$\frac{\llbracket b \rrbracket_{\text{boolexp}} s = \mathbf{false} \quad (s, h) \llbracket c_1 \rrbracket_{\text{comm}} \tau'}{(s, h) \llbracket \mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1 \rrbracket_{\text{comm}} \tau',}$$

when $\text{FV}(b) \subseteq \text{dom } s$.

while Commands

- **while**

$$\frac{\forall i \in 0 \text{ to } n - 1 : \llbracket b \rrbracket_{\text{boolexp}} s_i = \mathbf{true} \quad (s_i, h_i) \llbracket c \rrbracket_{\text{comm}} (s_{i+1}, h_{i+1}) \quad \llbracket b \rrbracket_{\text{boolexp}} s_n = \mathbf{false}}{(s_0, h_0) \llbracket \mathbf{while } b \text{ do } c \rrbracket_{\text{comm}} (s_n, h_n)}$$

$$\frac{\forall i \in 0 \text{ to } n - 1 : \llbracket b \rrbracket_{\text{boolexp}} s_i = \mathbf{true} \quad (s_i, h_i) \llbracket c \rrbracket_{\text{comm}} (s_{i+1}, h_{i+1}) \quad \llbracket b \rrbracket_{\text{boolexp}} s_n = \mathbf{true} \quad (s_n, h_n) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort}}{(s_0, h_0) \llbracket \mathbf{while } b \text{ do } c \rrbracket_{\text{comm}} \mathbf{abort}}$$

$$\frac{\forall i \in 0 \text{ to } n - 1 : \llbracket b \rrbracket_{\text{boolexp}} s_i = \mathbf{true} \quad (s_i, h_i) \llbracket c \rrbracket_{\text{comm}} (s_{i+1}, h_{i+1}) \quad \llbracket b \rrbracket_{\text{boolexp}} s_n = \mathbf{true} \quad (s_n, h_n) \llbracket c \rrbracket_{\text{comm}} \perp}{(s_0, h_0) \llbracket \mathbf{while } b \text{ do } c \rrbracket_{\text{comm}} \perp}$$

$$\frac{\forall i \in 0 \text{ to } \infty : \llbracket b \rrbracket_{\text{boolexp}} s_i = \mathbf{true} \quad (s_i, h_i) \llbracket c \rrbracket_{\text{comm}} (s_{i+1}, h_{i+1})}{(s_0, h_0) \llbracket \mathbf{while } b \text{ do } c \rrbracket_{\text{comm}} \perp,}$$

when $\text{FV}(b) \subseteq \text{dom } s_0$.

Notice that:

- Each of the first three rules has a variable number of premisses.
- The last rule has an infinite number of premisses.
- The rules are compositional (i.e., syntax-directed).

Variable Declarations

- **newvar** (without initialization)

$$\frac{([s \mid v:n], h) \llbracket c \rrbracket_{\text{comm}} (s', h')}{(s, h) \llbracket \mathbf{newvar} \ v \ \mathbf{in} \ c \rrbracket_{\text{comm}} (s' \mid \text{dom } s, h'),}$$

when $v \notin \text{dom } s$,

$$\frac{([s \mid v:n], h) \llbracket c \rrbracket_{\text{comm}} (s', h')}{(s, h) \llbracket \mathbf{newvar} \ v \ \mathbf{in} \ c \rrbracket_{\text{comm}} ([s' \mid v:sv], h'),}$$

when $v \in \text{dom } s$,

$$\frac{([s \mid v:n], h) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort}}{(s, h) \llbracket \mathbf{newvar} \ v \ \mathbf{in} \ c \rrbracket_{\text{comm}} \mathbf{abort}}$$

$$\frac{([s \mid v:n], h) \llbracket c \rrbracket_{\text{comm}} \perp}{(s, h) \llbracket \mathbf{newvar} \ v \ \mathbf{in} \ c \rrbracket_{\text{comm}} \perp.}$$

Variable Declarations (continued)

- **newvar** (with initialization)

$$\frac{([s \mid v: \llbracket e \rrbracket_{\text{exp}} s], h) \llbracket c \rrbracket_{\text{comm}} (s', h')}{(s, h) \llbracket \mathbf{newvar} \ v := e \ \mathbf{in} \ c \rrbracket_{\text{comm}} (s' \upharpoonright \text{dom } s, h'),}$$

when $v \notin \text{dom } s$ and $\text{FV}(e) \subseteq \text{dom } s$,

$$\frac{([s \mid v: \llbracket e \rrbracket_{\text{exp}} s], h) \llbracket c \rrbracket_{\text{comm}} (s', h')}{(s, h) \llbracket \mathbf{newvar} \ v := e \ \mathbf{in} \ c \rrbracket_{\text{comm}} ([s' \mid v: s \ v], h'),}$$

when $v \in \text{dom } s$ and $\text{FV}(e) \subseteq \text{dom } s$.

$$\frac{([s \mid v: \llbracket e \rrbracket_{\text{exp}} s], h) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort}}{(s, h) \llbracket \mathbf{newvar} \ v := e \ \mathbf{in} \ c \rrbracket_{\text{comm}} \mathbf{abort}}$$

$$\frac{([s \mid v: \llbracket e \rrbracket_{\text{exp}} s], h) \llbracket c \rrbracket_{\text{comm}} \perp}{(s, h) \llbracket \mathbf{newvar} \ v := e \ \mathbf{in} \ c \rrbracket_{\text{comm}} \perp.}$$

Semantics of the New Commands

- Allocation

$$\frac{}{(s, h) \llbracket v := \mathbf{cons}(e_0, \dots, e_{n-1}) \rrbracket_{\text{comm}} \left([s \mid v: \ell], [h \mid \ell: \llbracket e_0 \rrbracket_{\text{exp}} s \mid \dots \mid \ell + n - 1: \llbracket e_{n-1} \rrbracket_{\text{exp}} s] \right),}$$

when $\ell, \dots, \ell + n - 1 \in \text{Addresses} - \text{dom } h$ and $\{v\} \cup \text{FV}(e_0, \dots, e_{n-1}) \subseteq \text{dom } s$.

- Lookup

$$\frac{}{(s, h) \llbracket v := [e] \rrbracket_{\text{comm}} \left([s \mid v: h(\llbracket e \rrbracket_{\text{exp}} s)], h \right),}$$

when $\llbracket e \rrbracket_{\text{exp}} s \in \text{dom } h$ and $\{v\} \cup \text{FV}(e) \subseteq \text{dom } s$,

$$\frac{}{(s, h) \llbracket v := [e] \rrbracket_{\text{comm}} \mathbf{abort},}$$

when $\llbracket e \rrbracket_{\text{exp}} s \notin \text{dom } h$ and $\{v\} \cup \text{FV}(e) \subseteq \text{dom } s$.

More New Commands

- Mutation

$$\frac{}{(s, h) \llbracket [e] := e' \rrbracket_{\text{comm}} (s, [h \mid \llbracket e \rrbracket_{\text{exp}} s : \llbracket e' \rrbracket_{\text{exp}} s])},$$

when $\llbracket e \rrbracket_{\text{exp}} s \in \text{dom } h$ and $\text{FV}(e) \cup \text{FV}(e') \subseteq \text{dom } s$,

$$\frac{}{(s, h) \llbracket [e] := e' \rrbracket_{\text{comm}} \mathbf{abort},$$

when $\llbracket e \rrbracket_{\text{exp}} s \notin \text{dom } h$ and $\text{FV}(e) \cup \text{FV}(e') \subseteq \text{dom } s$.

- Disposal

$$\frac{}{(s, h) \llbracket \mathbf{dispose } e \rrbracket_{\text{comm}} (s, h \upharpoonright (\text{dom } h - \{\llbracket e \rrbracket_{\text{exp}} s\}))},$$

when $\llbracket e \rrbracket_{\text{exp}} s \in \text{dom } h$ and $\text{FV}(e) \subseteq \text{dom } s$,

$$\frac{}{(s, h) \llbracket \mathbf{dispose } e \rrbracket_{\text{comm}} \mathbf{abort},$$

when $\llbracket e \rrbracket_{\text{exp}} s \notin \text{dom } h$ and $\text{FV}(e) \subseteq \text{dom } s$.

Totality

Proposition 6 *Our semantics is total: If $(s, h) \in \text{States}_V$ for some $V \stackrel{\text{fn}}{\supseteq} \text{FV}(c)$, then there is at least one $\tau' \in \text{States}_V \cup \{\mathbf{abort}, \perp\}$ such that $(s, h) \llbracket c \rrbracket_{\text{comm}} \tau'$.*

Some Definitions

When $(s, h) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort}$ is false, we say that c is *safe at* (s, h) .

When $(s, h) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort}$ and $(s, h) \llbracket c \rrbracket_{\text{comm}} \perp$ are both false, we say that c *must terminate normally at* (s, h) .

By Proposition 6, if c must terminate normally at (s, h) , then there is at least one state (s', h') such that $(s, h) \llbracket c \rrbracket_{\text{comm}} (s', h')$.

Modified Variables

We extend the definition of $\text{MD}(c)$ as follows:

$$\begin{aligned} \text{MD}(v := e) &= \{v\} \\ \text{MD}(v := \mathbf{cons}(e_0, \dots, e_{n-1})) &= \{v\} \\ \text{MD}(v := [e]) &= \{v\} \\ \text{MD}(\mathbf{skip}) &= \{\} \\ \text{MD}([e] := e') &= \{\} \\ \text{MD}(\mathbf{dispose } e) &= \{\} \\ \text{MD}(c_0 ; c_1) &= \text{MD}(c_0) \cup \text{MD}(c_1) \\ \text{MD}(\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1) &= \text{MD}(c_0) \cup \text{MD}(c_1) \\ \text{MD}(\mathbf{while } b \mathbf{ do } c) &= \text{MD}(c) \\ \text{MD}(\mathbf{newvar } v \mathbf{ in } c) &= \text{MD}(c) - \{v\} \\ \text{MD}(\mathbf{newvar } v := e \mathbf{ in } c) &= \text{MD}(c) - \{v\} \end{aligned}$$

Then we still have the property

Proposition 7 *Suppose $v \in \text{dom}(s)$ and $v \notin \text{MD}(c)$. If*

$$(s, h) \llbracket c \rrbracket_{\text{comm}} (s', h'),$$

then $s'v = sv$.

Splitting the Heap

We define

$$h_0 \perp h_1 \text{ iff } \text{dom } h_0 \cap \text{dom } h_1 = \{\}.$$

We will regard a heap as the set of pairs called its graph, so that heaps can be compared by the subset relation and composed by set union and difference.

When $h_0 \perp h_1$, we will write $h_0 \cdot h_1$ for $h_0 \cup h_1$. (This is the case where $h_0 \cup h_1$ is a function.)

Proposition 8 *Suppose $\hat{h} \subseteq h$. Then*

- *If $(s, h) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort}$, then*

$$(s, h - \hat{h}) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort}.$$

- *If $(s, h) \llbracket c \rrbracket_{\text{comm}} (s', h')$, then*

$$(s, h - \hat{h}) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort}$$

$$\text{or } \hat{h} \subseteq h' \text{ and } (s, h - \hat{h}) \llbracket c \rrbracket_{\text{comm}} (s', h' - \hat{h}).$$

- *If $(s, h) \llbracket c \rrbracket_{\text{comm}} \perp$, then*

$$(s, h - \hat{h}) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort}$$

$$\text{or } (s, h - \hat{h}) \llbracket c \rrbracket_{\text{comm}} \perp.$$

Proof of Proposition 8

We use structural induction on the command c , with a case analysis on the type of the command. We present two cases:

(1) When c is $[e] := e'$. Suppose $\hat{h} \subseteq h$.

- If $(s, h) \llbracket [e] := e' \rrbracket_{\text{comm}} \mathbf{abort}$, then $\llbracket e \rrbracket_{\text{exp}} s \notin \text{dom } h$. Then $\llbracket e \rrbracket_{\text{exp}} s \notin \text{dom}(h - \hat{h})$, so that

$$(s, h - \hat{h}) \llbracket [e] := e' \rrbracket_{\text{comm}} \mathbf{abort}.$$

- If $(s, h) \llbracket [e] := e' \rrbracket_{\text{comm}} (s', h')$, then $\llbracket e \rrbracket_{\text{exp}} s \in \text{dom } h$ and

$$(s, h) \llbracket [e] := e' \rrbracket_{\text{comm}} (s, [h \mid \llbracket e \rrbracket_{\text{exp}} s : \llbracket e' \rrbracket_{\text{exp}} s]),$$

so that $s' = s$ and $h' = [h \mid \llbracket e \rrbracket_{\text{exp}} s : \llbracket e' \rrbracket_{\text{exp}} s]$.

If $\llbracket e \rrbracket_{\text{exp}} s \notin \text{dom}(h - \hat{h})$, then

$$(s, h - \hat{h}) \llbracket [e] := e' \rrbracket_{\text{comm}} \mathbf{abort},$$

as before. But if $\llbracket e \rrbracket_{\text{exp}} s \in \text{dom}(h - \hat{h})$, then

$$(s, h - \hat{h}) \llbracket [e] := e' \rrbracket_{\text{comm}} (s, [h - \hat{h} \mid \llbracket e \rrbracket_{\text{exp}} s : \llbracket e' \rrbracket_{\text{exp}} s]),$$

and $h' = [h \mid \llbracket e \rrbracket_{\text{exp}} s : \llbracket e' \rrbracket_{\text{exp}} s] = [(h - \hat{h}) \cdot \hat{h} \mid \llbracket e \rrbracket_{\text{exp}} s : \llbracket e' \rrbracket_{\text{exp}} s]$
 $= [h - \hat{h} \mid \llbracket e \rrbracket_{\text{exp}} s : \llbracket e' \rrbracket_{\text{exp}} s] \cdot \hat{h}$. Then $\hat{h} \subseteq h'$ and

$$(s, h - \hat{h}) \llbracket [e] := e' \rrbracket_{\text{comm}} (s, h' - \hat{h}).$$

- The case $(s, h) \llbracket [e] := e' \rrbracket_{\text{comm}} \perp$ is impossible.

Proof of Proposition 8 (continued)

(2) When c is **while** b **do** c . Suppose $\hat{h} \subseteq h$ and that

$$(s, h) \llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket_{\text{comm}} \tau. \quad (2)$$

Since the final step in deriving the triple (2) must have been an application of one of the rules for the **while** command, there must be a sequence of states (s_i, h_i) , for $i \in 0$ **to** n , such that $s_0 = s$, $h_0 = h$, and for $i \in 0$ **to** $n - 1$, $\llbracket b \rrbracket_{\text{boolexp}} s_i = \mathbf{true}$ and

$$(s_i, h_i) \llbracket c \rrbracket_{\text{comm}} (s_{i+1}, h_{i+1}).$$

(Because of the fourth **while** rule, we must include the possibility that n is ∞ .)

By the induction hypothesis, the above triples satisfy the second part of Proposition 8:

If $\hat{h} \subseteq h$ and $(s, h) \llbracket c \rrbracket_{\text{comm}} (s', h')$, then

$$\begin{aligned} & (s, h - \hat{h}) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort} \\ & \text{or } \hat{h} \subseteq h' \text{ and } (s, h - \hat{h}) \llbracket c \rrbracket_{\text{comm}} (s', h' - \hat{h}). \end{aligned}$$

Since the condition $\hat{h} \subseteq h$ here reproduces itself as $\hat{h} \subseteq h'$, we can repeatedly apply the second part of Proposition 8 to successive triples, terminating if either **abort** appears or the sequence of triples is exhausted. We find that there is a $k \in 0$ **to** n such that, for $i \in 0$ **to** $k - 1$,

$$(s_i, h_i - \hat{h}) \llbracket c \rrbracket_{\text{comm}} (s_{i+1}, h_{i+1} - \hat{h})$$

Proof of Proposition 8 (continued)

and either k is n (which may be ∞) or

$$(s_k, h_k - \hat{h}) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort}.$$

In the latter case, the second rule for the **while** command gives

$$(s, h - \hat{h}) \llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket_{\text{comm}} \mathbf{abort}.$$

Otherwise, $k = n$, and our argument depends on which **while** rule was used as the last step in deriving the triple (2). If the first rule was used, then $\tau = (s_n, h_n)$ and $\llbracket b \rrbracket_{\text{boolexp}}^{s_n} = \mathbf{false}$. Then we can use the first rule to obtain

$$(s, h - \hat{h}) \llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket_{\text{comm}} (s_n, h_n - \hat{h}).$$

If the second **while** rule was used as the last step in deriving (2), then $\tau = \mathbf{abort}$, $\llbracket b \rrbracket_{\text{boolexp}}^{s_n} = \mathbf{true}$, and

$$(s_n, h_n) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort},$$

so by the induction hypothesis,

$$(s_n, h_n - \hat{h}) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort},$$

and the second rule gives

$$(s, h - \hat{h}) \llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket_{\text{comm}} \mathbf{abort}.$$

Proof of Proposition 8 (continued)

If the third **while** rule was used as the last step in deriving (2), then $\tau = \perp$, $\llbracket b \rrbracket_{\text{boolexp}} s_n = \mathbf{true}$, and

$$(s_n, h_n) \llbracket c \rrbracket_{\text{comm}} \perp,$$

so by the induction hypothesis,

$$(s_n, h_n - \hat{h}) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort} \text{ or } (s_n, h_n - \hat{h}) \llbracket c \rrbracket_{\text{comm}} \perp,$$

and the second or third rule gives

$$(s, h - \hat{h}) \llbracket \mathbf{while } b \mathbf{ do } c \rrbracket_{\text{comm}} \mathbf{abort} \text{ or } (s, h - \hat{h}) \llbracket \mathbf{while } b \mathbf{ do } c \rrbracket_{\text{comm}} \perp.$$

If the fourth **while** rule was used as the last step in deriving (2), then $\tau = \perp$ and $n = \infty$. Then the fourth rule gives

$$(s, h - \hat{h}) \llbracket \mathbf{while } b \mathbf{ do } c \rrbracket_{\text{comm}} \perp.$$

END OF PROOF

Putting It Positively

Proposition 8 is equivalent to the conjunction of the following propositions:

Safety Monotonicity

Proposition 9 *If $\hat{h} \subseteq h$ and c is safe at $(s, h - \hat{h})$, then c is safe at (s, h) .*

If $\hat{h} \subseteq h$ and c must terminate normally at $(s, h - \hat{h})$, then c must terminate normally at (s, h) .

The Frame Property

Proposition 10 *If*

$$\hat{h} \subseteq h, c \text{ is safe at } (s, h - \hat{h}), \text{ and } (s, h) \llbracket c \rrbracket_{\text{comm}} (s', h'),$$

then

$$\hat{h} \subseteq h' \text{ and } (s, h - \hat{h}) \llbracket c \rrbracket_{\text{comm}} (s', h' - \hat{h}).$$

Syntax of Assertions

$\langle \text{assert} \rangle ::= \dots$

| **emp**

| $\langle \text{exp} \rangle \mapsto \langle \text{exp} \rangle$

| $\langle \text{assert} \rangle * \langle \text{assert} \rangle$

| $\langle \text{assert} \rangle \multimap \langle \text{assert} \rangle$

The Semantics of Assertions

$$\llbracket p \in \langle \text{assert} \rangle \rrbracket_{\text{assert}} \in \left(\bigcup_{V \supseteq \text{FV}(p)}^{\text{fin}} \text{States}_V \right) \rightarrow \mathbf{B}.$$

Instead of $\llbracket p \rrbracket_{\text{assert}}(s, h) = \mathbf{true}$, however, we write $s, h \models p$.

$$s, h \models b \text{ iff } \llbracket b \rrbracket_{\text{boolexp}} s = \mathbf{true},$$

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

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

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

$$s, h \models \forall v. p \text{ iff } \forall x \in \mathbf{Z}. [s \mid v: x], h \models p,$$

$$s, h \models \exists v. p \text{ iff } \exists x \in \mathbf{Z}. [s \mid v: x], h \models p,$$

$$s, h \models \mathbf{emp} \text{ iff } \text{dom } h = \{\},$$

$$s, h \models e \mapsto e' \text{ iff } \text{dom } h = \{\llbracket e \rrbracket_{\text{exp}} s\} \text{ and } h(\llbracket e \rrbracket_{\text{exp}} s) = \llbracket e' \rrbracket_{\text{exp}} s,$$

$$s, h \models p_0 * p_1 \text{ iff } \exists h_0, h_1. h_0 \perp h_1 \text{ and } h_0 \cdot h_1 = h \text{ and}$$

$$s, h_0 \models p_0 \text{ and } s, h_1 \models p_1,$$

$$s, h \models p_0 \multimap p_1 \text{ iff } \forall h'. (h' \perp h \text{ and } s, h' \models p_0) \text{ implies}$$

$$s, h \cdot h' \models p_1.$$

An Example

$s, h \models x \mapsto 0 * y \mapsto 1$ iff $\exists h_0, h_1. h_0 \perp h_1$ and $h_0 \cdot h_1 = h$
 and $s, h_0 \models x \mapsto 0$
 and $s, h_1 \models y \mapsto 1$

iff $\exists h_0, h_1. h_0 \perp h_1$ and $h_0 \cdot h_1 = h$
 and $\text{dom } h_0 = \{s x\}$ and $h_0(s x) = 0$
 and $\text{dom } h_1 = \{s y\}$ and $h_1(s y) = 1$

iff $s x \neq s y$
 and $\text{dom } h = \{s x, s y\}$
 and $h(s x) = 0$ and $h(s y) = 1$

iff $s x \neq s y$ and $h = [s x: 0 \mid s y: 1]$.

Some Abbreviations

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

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

$$e \mapsto e_0, \dots, e_{n-1} \stackrel{\text{def}}{=} e \mapsto e_0 * \dots * e_{n-1} \mapsto e_{n-1}$$

$$e \hookrightarrow e_0, \dots, e_{n-1} \stackrel{\text{def}}{=} e \hookrightarrow e_0 * \dots * e_{n-1} \hookrightarrow e_{n-1}$$

iff $e \mapsto e_0, \dots, e_{n-1} * \mathbf{true}$.

Examples

$$s, h \models x \mapsto y \text{ iff } \text{dom } h = \{s x\} \text{ and } h(s x) = s y$$

$$s, h \models x \mapsto - \text{ iff } \text{dom } h = \{s x\}$$

$$s, h \models x \hookrightarrow y \text{ iff } s x \in \text{dom } h \text{ and } h(s x) = s y$$

$$s, h \models x \hookrightarrow - \text{ iff } s x \in \text{dom } h$$

$$s, h \models x \mapsto y, z \text{ iff } h = [s x: s y \mid s x + 1: s z]$$

$$s, h \models x \mapsto -, - \text{ iff } \text{dom } h = \{s x, s x + 1\}$$

$$s, h \models x \hookrightarrow y, z \text{ iff } h \supseteq [s x: s y \mid s x + 1: s z]$$

$$s, h \models x \hookrightarrow -, - \text{ iff } \text{dom } h \supseteq \{s x, s x + 1\}.$$

Examples of $*$

Suppose $s x$ and $s y$ are distinct addresses, so that

$$h_1 = \{\langle s x, 1 \rangle\} \quad \text{and} \quad h_2 = \{\langle s y, 2 \rangle\}$$

are heaps with disjoint domains. Then

If p is:	then $s, h \models p$ iff:
$x \mapsto 1$	$h = h_1$
$y \mapsto 2$	$h = h_2$
$x \mapsto 1 * y \mapsto 2$	$h = h_1 \cdot h_2$
$x \mapsto 1 * x \mapsto 1$	false
$x \mapsto 1 \vee y \mapsto 2$	$h = h_1$ or $h = h_2$
$x \mapsto 1 * (x \mapsto 1 \vee y \mapsto 2)$	$h = h_1 \cdot h_2$
$(x \mapsto 1 \vee y \mapsto 2) * (x \mapsto 1 \vee y \mapsto 2)$	$h = h_1 \cdot h_2$
$x \mapsto 1 * y \mapsto 2 * (x \mapsto 1 \vee y \mapsto 2)$	false
$x \mapsto 1 * \mathbf{true}$	$h_1 \subseteq h$
$x \mapsto 1 * \neg x \mapsto 1$	$h_1 \subseteq h.$

Rules and Axiom Schemata for $*$ and $-*$

$$p_1 * p_2 \Leftrightarrow p_2 * p_1$$

$$(p_1 * p_2) * p_3 \Leftrightarrow p_1 * (p_2 * p_3)$$

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

$$(p_1 \vee p_2) * q \Leftrightarrow (p_1 * q) \vee (p_2 * q)$$

$$(p_1 \wedge p_2) * q \Rightarrow (p_1 * q) \wedge (p_2 * q)$$

$$(\exists x. p_1) * p_2 \Leftrightarrow \exists x. (p_1 * p_2) \quad \text{when } x \text{ not free in } p_2$$

$$(\forall x. p_1) * p_2 \Rightarrow \forall x. (p_1 * p_2) \quad \text{when } x \text{ not free in } p_2$$

$$\frac{p_1 \Rightarrow p_2 \quad q_1 \Rightarrow q_2}{p_1 * q_1 \Rightarrow p_2 * q_2} \quad (\text{monotonicity})$$

$$\frac{p_1 * p_2 \Rightarrow p_3}{p_1 \Rightarrow (p_2 -* p_3)} \quad (\text{currying}) \qquad \frac{p_1 \Rightarrow (p_2 -* p_3)}{p_1 * p_2 \Rightarrow p_3} \quad (\text{decurrying})$$

Axiom Schemata for \mapsto

$$e_1 \mapsto e'_1 \wedge e_2 \mapsto e'_2 \Leftrightarrow e_1 \mapsto e'_1 \wedge e_1 = e_2 \wedge e'_1 = e'_2$$

$$e_1 \mapsto e'_1 * e_2 \mapsto e'_2 * \mathbf{true} \Rightarrow e_1 \neq e_2$$

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

$$(e \hookrightarrow e') \wedge p \Rightarrow (e \mapsto e') * ((e \mapsto e') -* p).$$

A Proof of Soundness

To show that the axiom

$$(e \hookrightarrow e') \wedge p \Rightarrow (e \mapsto e') * ((e \mapsto e') \multimap p)$$

is sound, suppose $s, h \models (e \hookrightarrow e') \wedge p$. Then:

- $s, h \models (e \mapsto e') * \mathbf{true}$ and $s, h \models p$.
- There are heaps h_0 and h_1 such that $h_0 \perp h_1$, $h_0 \cdot h_1 = h$, and $s, h_0 \models e \mapsto e'$.
- Thus $h_0 = [[e]s : [e']s]$.
- To see that $s, h_1 \models (e \mapsto e') \multimap p$, let h' be any heap such that $h' \perp h_1$ and $s, h' \models e \mapsto e'$. Then $h' = [[e]s : [e']s] = h_0$, so that $h' \cdot h_1 = h_0 \cdot h_1 = h$, and thus $s, h' \cdot h_1 \models p$.
- Then $s, h_0 \cdot h_1 \models (e \mapsto e') * ((e \mapsto e') \multimap p)$, so that $s, h \models (e \mapsto e') * ((e \mapsto e') \multimap p)$.

Pure Assertions

- An assertion is *pure* iff, for any store, it is independent of the heap.
- Syntactically, an assertion is pure if it does not contain **emp**, \mapsto , or \hookrightarrow .

Axiom Schemata for Purity

$$\begin{array}{ll}
 p_1 \wedge p_2 \Rightarrow p_1 * p_2 & \text{when } p_1 \text{ or } p_2 \text{ is pure} \\
 p_1 * p_2 \Rightarrow p_1 \wedge p_2 & \text{when } p_1 \text{ and } p_2 \text{ are pure} \\
 (p \wedge q) * r \Leftrightarrow (p * r) \wedge q & \text{when } q \text{ is pure} \\
 (p_1 \multimap p_2) \Rightarrow (p_1 \Rightarrow p_2) & \text{when } p_1 \text{ is pure} \\
 (p_1 \Rightarrow p_2) \Rightarrow (p_1 \multimap p_2) & \text{when } p_1 \text{ and } p_2 \text{ are pure.}
 \end{array}$$

Two Unsound Axiom Schemata

$$\begin{array}{ll}
 p \not\Rightarrow p * p & \text{(Contraction)} \\
 & \text{e.g. } p : x \mapsto 1 \\
 \\
 p * q \not\Rightarrow p & \text{(Weakening)} \\
 & \text{e.g. } p : x \mapsto 1 \\
 & \quad q : y \mapsto 2
 \end{array}$$

Some Derived Inference Rules

$$\frac{}{q * (q \multimap p) \Rightarrow p}$$

1. $q * (q \multimap p) \Rightarrow (q \multimap p) * q$ ($p_1 * p_2 \Rightarrow p_2 * p_1$)
2. $(q \multimap p) \Rightarrow (q \multimap p)$ ($p \Rightarrow p$)
3. $(q \multimap p) * q \Rightarrow p$ (decurrying, 2)
4. $q * (q \multimap p) \Rightarrow p$ (trans impl, 1, 3)

where *transitive implication* is the inference rule

$$\frac{p \Rightarrow q \quad q \Rightarrow r}{p \Rightarrow r.}$$

$$\frac{}{r \Rightarrow (q \multimap (q * r))}$$

1. $(r * q) \Rightarrow (q * r)$ ($p_1 * p_2 \Rightarrow p_2 * p_1$)
2. $r \Rightarrow (q \multimap (q * r))$ (currying, 1)

More Derived Inference Rules

$$\overline{(p * r) \Rightarrow (p * (q \multimap (q * r)))}$$

1. $p \Rightarrow p$ ($p \Rightarrow p$)
2. $r \Rightarrow (q \multimap (q * r))$ (derived above)
3. $(p * r) \Rightarrow (p * (q \multimap (q * r)))$ (monotonicity, 1, 2)

$$\frac{p_1 \Rightarrow (q \multimap r) \quad p_2 \Rightarrow (r \multimap s)}{p_2 * p_1 \Rightarrow (q \multimap s)}$$

1. $p_2 \Rightarrow p_2$ ($p \Rightarrow p$)
2. $p_1 \Rightarrow (q \multimap r)$ (assumption)
3. $p_1 * q \Rightarrow r$ (decurrying, 2)
4. $p_2 * p_1 * q \Rightarrow p_2 * r$ (monotonicity, 1, 3)
5. $p_2 \Rightarrow (r \multimap s)$ (assumption)
6. $p_2 * r \Rightarrow s$ (decurrying, 5)
7. $p_2 * p_1 * q \Rightarrow s$ (trans impl, 4, 6)
8. $p_2 * p_1 \Rightarrow (q \multimap s)$ (currying, 7)

More Derived Inference Rules

$$\frac{p' \Rightarrow p \quad q \Rightarrow q'}{(p \multimap q) \Rightarrow (p' \multimap q')}.$$

1. $(p \multimap q) \Rightarrow (p \multimap q)$ ($p \Rightarrow p$)
2. $p' \Rightarrow p$ (assumption)
3. $(p \multimap q) * p' \Rightarrow (p \multimap q) * p$ (monotonicity, 1, 2)
4. $(p \multimap q) * p \Rightarrow q$ (decourrying, 1)
5. $(p \multimap q) * p' \Rightarrow q$ (trans impl, 3, 4)
6. $q \Rightarrow q'$ (assumption)
7. $(p \multimap q) * p' \Rightarrow q'$ (trans impl, 5, 6)
8. $(p \multimap q) \Rightarrow (p' \multimap q')$ (currying, 7)

Specifications

As before, we have the syntax:

$$\begin{aligned} \langle \text{spec} \rangle &::= \{ \langle \text{assert} \rangle \} \langle \text{comm} \rangle \{ \langle \text{assert} \rangle \} && \text{(partial correctness)} \\ &| [\langle \text{assert} \rangle] \langle \text{comm} \rangle [\langle \text{assert} \rangle] && \text{(total correctness)} \end{aligned}$$

Now however,

- Specifications are implicitly quantified over both stores and heaps — and over all executions.
- An execution giving a memory fault falsifies a specification.

In particular, let $V = \text{FV}(p) \cup \text{FV}(c) \cup \text{FV}(q)$. Then

Partial correctness:

$\{p\} c \{q\}$ holds iff $\forall (s, h) \in \text{States}_V. s, h \models p$ implies

$$\begin{aligned} &\neg (s, h) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort} \\ &\quad \{ \text{i.e., } c \text{ is safe at } (s, h) \} \end{aligned}$$

and $(\forall (s', h') \in \text{States}_V. (s, h) \llbracket c \rrbracket_{\text{comm}} (s', h') \text{ implies } s', h' \models q)$

Total correctness:

$[p] c [q]$ holds iff $\forall (s, h) \in \text{States}_V. s, h \models p$ implies

$$\begin{aligned} &\neg (s, h) \llbracket c \rrbracket_{\text{comm}} \mathbf{abort} \text{ and } \neg (s, h) \llbracket c \rrbracket_{\text{comm}} \perp \\ &\quad \{ \text{i.e., } c \text{ must terminate normally at } (s, h) \} \end{aligned}$$

and $(\forall (s', h') \in \text{States}_V. (s, h) \llbracket c \rrbracket_{\text{comm}} (s', h') \text{ implies } s', h' \models q)$

Examples of Specifications

$$\begin{aligned}
 & \{\mathbf{emp}\} x := \mathbf{cons}(1, 2) \{x \mapsto 1, 2\} \\
 & \{x \mapsto 1, 2\} y := [x] \{x \mapsto 1, 2 \wedge y = 1\} \\
 & \{x \mapsto 1, 2 \wedge y = 1\} [x + 1] := 3 \{x \mapsto 1, 3 \wedge y = 1\} \\
 & \{x \mapsto 1, 3 \wedge y = 1\} \mathbf{dispose} x \{x + 1 \mapsto 3 \wedge y = 1\},
 \end{aligned}$$

and similarly

$$\begin{aligned}
 & [\mathbf{emp}] x := \mathbf{cons}(1, 2) [x \mapsto 1, 2] \\
 & [x \mapsto 1, 2] y := [x] [x \mapsto 1, 2 \wedge y = 1] \\
 & [x \mapsto 1, 2 \wedge y = 1] [x + 1] := 3 [x \mapsto 1, 3 \wedge y = 1] \\
 & [x \mapsto 1, 3 \wedge y = 1] \mathbf{dispose} x [x + 1 \mapsto 3 \wedge y = 1].
 \end{aligned}$$

The Rules of Hoare Logic

All of the inference rules we have given for Hoare logic (including those for partial and total correctness, annotated specifications, weakest preconditions, and weakest liberal preconditions), *except* the rule of constancy (CONST), remain sound for separation logic.

The Unsound Rule of Constancy

$$\frac{\{p\} c \{q\}}{\{p \wedge r\} c \{q \wedge r\}},$$

when no variable occurs in both $FV(r)$ and $MD(c)$.

Counterexamples

$$\frac{\{x = y \wedge x \mapsto 3\} [x] := 4 \{x = y \wedge x \mapsto 4\}}{\{x = y \wedge x \mapsto 3 \wedge y \mapsto 3\} [x] := 4 \{x = y \wedge x \mapsto 4 \wedge y \mapsto 3\}}$$

$$\frac{\{x \mapsto 3\} [x] := 4 \{x \mapsto 4\}}{\{x \mapsto 3 \wedge x \mapsto 3\} [x] := 4 \{x \mapsto 4 \wedge x \mapsto 3\}}.$$

Additional Inference Rules for Separation Logic

The rule of constancy is replaced by a new structural rule called the *frame rule*. There are also several rules specific to each of the new heap-manipulating commands. All of these rules are the same for partial and total correctness.

The Frame Rule (FR)

$$\frac{\{p\} c \{q\}}{\{p * r\} c \{q * r\}},$$

when no variable occurs in both $FV(r)$ and $MD(c)$.

Proposition 11 *The frame rule is sound.*

PROOF

- Assume $\{p\} c \{q\}$ is valid, and $s, h \models p * r$. Then there is an $\hat{h} \subseteq h$ such that $s, h - \hat{h} \models p$ and $s, \hat{h} \models r$.
- From $\{p\} c \{q\}$, we know that c is safe at $(s, h - \hat{h})$ (and in the total-correctness case it must terminate normally). Then, by Proposition 9, we know that c is safe at (s, h) (and in the total-correctness case it must terminate normally).
- Suppose we have $(s, h) \llbracket c \rrbracket_{\text{comm}} (s', h')$. Since c is safe at $(s, h - \hat{h})$, we know by Proposition 10 that $\hat{h} \subseteq h'$ and $(s, h - \hat{h}) \llbracket c \rrbracket_{\text{comm}} (s', h' - \hat{h})$. Then $\{p\} c \{q\}$ and $s, h - \hat{h} \models p$ imply that $s', h' - \hat{h} \models q$.
- Since $(s, h) \llbracket c \rrbracket_{\text{comm}} (s', h')$, Proposition 7 gives $s v = s' v$ for all v that are not modified by c . Then, since these include the free variables of r , $s, \hat{h} \models r$ implies that $s', \hat{h} \models r$. Thus $s', h' \models q * r$.

END OF PROOF

Inference Rules for Mutation

The local form (MUL):

$$\frac{}{\{e \mapsto -\} [e] := e' \{e \mapsto e'\}}.$$

The global form (MUG):

$$\frac{}{\{(e \mapsto -) * r\} [e] := e' \{(e \mapsto e') * r\}}.$$

The backward-reasoning form (MUBR):

$$\frac{}{\{(e \mapsto -) * ((e \mapsto e') \multimap p)\} [e] := e' \{p\}}.$$

One can derive (MUG) from (MUL) by using the frame rule; one can go in the opposite direction by taking r to be **emp**.

One can derive (MUBR) from (MUG) by taking r to be $(e \mapsto e') \multimap p$ and using the law $q * (q \multimap p) \Rightarrow p$:

$$\frac{\{(e \mapsto -) * ((e \mapsto e') \multimap p)\} [e] := e' \{(e \mapsto e') * ((e \mapsto e') \multimap p)\} \{p\}}{\{(e \mapsto -) * ((e \mapsto e') \multimap p)\} [e] := e' \{p\}}.$$

One can go in the opposite direction by taking p to be $(e \mapsto e') * r$ and using the law $(p * r) \Rightarrow (p * (q \multimap (q * r)))$:

$$\frac{\{(e \mapsto -) * r\} \{(e \mapsto -) * ((e \mapsto e') \multimap ((e \mapsto e') * r))\} [e] := e' \{(e \mapsto e') * r\}}{\{(e \mapsto -) * r\} [e] := e' \{(e \mapsto e') * r\}}.$$

Inference Rules for Deallocation

The local form (DISL):

$$\frac{}{\{e \mapsto -\} \mathbf{dispose} e \{\mathbf{emp}\}}.$$

The global (and backward-reasoning) form (DISG):

$$\frac{}{\{(e \mapsto -) * r\} \mathbf{dispose} e \{r\}}.$$

One can derive (DISG) from (DISL) by using (FR); one can go in the opposite direction by taking r to be **emp**.

Inference Rules for Noninterfering Allocation

The local form (CONSNIL):

$$\frac{}{\{\mathbf{emp}\} v := \mathbf{cons}(e_0, \dots, e_{n-1}) \{v \mapsto e_0, \dots, e_{n-1}\}},$$

where $v \notin \text{FV}(e_0, \dots, e_{n-1})$.

The global form (CONSNIG):

$$\frac{}{\{r\} v := \mathbf{cons}(e_0, \dots, e_{n-1}) \{(v \mapsto e_0, \dots, e_{n-1}) * r\}},$$

where $v \notin \text{FV}(e_0, \dots, e_{n-1}, r)$.

One can derive (CONSNIG) from (CONSNIL) by using the frame rule (FR); one can go in the opposite direction by taking r to be **emp**.

Inference Rules for General Allocation

The local form (CONSL):

$$\frac{}{\{v = v' \wedge \mathbf{emp}\} v := \mathbf{cons}(e_0, \dots, e_{n-1}) \{v \mapsto e'_0, \dots, e'_{n-1}\}}.$$

where v' is distinct from v , and e'_i denotes $e_i/v \rightarrow v'$.

The global form (CONSG):

$$\frac{}{\{r\} v := \mathbf{cons}(e_0, \dots, e_{n-1}) \{\exists v'. (v \mapsto e'_0, \dots, e'_{n-1}) * r'\}},$$

where v' is distinct from v , $v' \notin \text{FV}(e_0, \dots, e_{n-1}, r)$, e'_i denotes $e_i/v \rightarrow v'$, and r' denotes $r/v \rightarrow v'$.

The backward-reasoning form (CONSBR):

$$\frac{}{\{\forall v''. (v'' \mapsto e_0, \dots, e_{n-1}) \multimap p''\} v := \mathbf{cons}(e_0, \dots, e_{n-1}) \{p\}},$$

where v'' is distinct from v , $v'' \notin \text{FV}(e_0, \dots, e_{n-1}, p)$, and p'' denotes $p/v \rightarrow v''$.

Inference Rules for General Allocation (continued)

Let \bar{e} abbreviate e_0, \dots, e_{n-1} , \bar{e}' abbreviate e'_0, \dots, e'_{n-1} , and similarly for e'' . One can derive (CONSG) from (CONSL) by using (FR) and (EQ):

$$\begin{array}{ll}
 \{v = v' \wedge r'\} & \{r\} \\
 \{v = v' \wedge (\mathbf{emp} * r')\} & \{\exists v'. v = v' \wedge r\} \\
 \{(v = v' \wedge \mathbf{emp}) * r'\} & \{\exists v'. v = v' \wedge r'\} \\
 v := \mathbf{cons}(\bar{e}) & v := \mathbf{cons}(\bar{e}) \\
 \{(v \mapsto \bar{e}') * r'\}, & \{\exists v'. (v \mapsto \bar{e}') * r'\}.
 \end{array}$$

To go in the other direction:

$$\begin{array}{l}
 \{v = v' \wedge \mathbf{emp}\} \\
 v := \mathbf{cons}(\bar{e}) \\
 \{\exists v''. (v \mapsto \bar{e}'') * (v'' = v' \wedge \mathbf{emp})\} \\
 \{\exists v''. ((v \mapsto \bar{e}'') * \mathbf{emp}) \wedge v'' = v'\} \\
 \{\exists v''. (v \mapsto \bar{e}'') \wedge v'' = v'\} \\
 \{\exists v''. (v \mapsto \bar{e}') \wedge v'' = v'\} \\
 \{v \mapsto \bar{e}'\}.
 \end{array}$$

Inference Rules for General Allocation (continued)

In order to derive (CONSBR) from (CONSG), we choose $v' \notin \text{FV}(e_0, \dots, e_{n-1}, p)$ to be distinct from v and v'' , take r to be $\forall v''. (v'' \mapsto \bar{e}) \multimap p''$, and use various laws about quantifiers, as well as $q * (q \multimap p) \Rightarrow p$:

$$\begin{aligned}
 & \{\forall v''. (v'' \mapsto \bar{e}) \multimap p''\} \\
 & v := \mathbf{cons}(\bar{e}) \\
 & \{\exists v'. (v \mapsto \bar{e}') * (\forall v''. (v'' \mapsto \bar{e}') \multimap p'')\} \\
 & \{\exists v'. (v \mapsto \bar{e}') * ((v \mapsto \bar{e}') \multimap p)\} \\
 & \{\exists v'. p\} \\
 & \{p\}.
 \end{aligned}$$

To go in the other direction, we choose $v'' \notin \text{FV}(e_0, \dots, e_{n-1}, r)$ to be distinct from v and v' , take p to be $\exists v'. (v \mapsto \bar{e}') * r'$, and use various laws about quantifiers, as well as $r \Rightarrow (q \multimap (q * r))$:

$$\begin{aligned}
 & \{r\} \\
 & \{\forall v''. r\} \\
 & \{\forall v''. (v'' \mapsto \bar{e}) \multimap ((v'' \mapsto \bar{e}) * r)\} \\
 & \{\forall v''. (v'' \mapsto \bar{e}) \multimap (((v'' \mapsto \bar{e}') * r') / v' \rightarrow v)\} \\
 & \{\forall v''. (v'' \mapsto \bar{e}) \multimap (\exists v'. (v'' \mapsto \bar{e}') * r')\} \\
 & v := \mathbf{cons}(\bar{e}) \\
 & \{\exists v'. (v \mapsto \bar{e}') * r'\}.
 \end{aligned}$$

Inference Rules for Lookup

The local form (LKL):

$$\frac{}{\{v = v' \wedge (e \mapsto v'')\} v := [e] \{v = v'' \wedge (e' \mapsto v'')\}},$$

where v , v' , and v'' are distinct, and e' denotes $e/v \rightarrow v'$.

The global form (LKG):

$$\frac{}{\{\exists v''. (e \mapsto v'') * (r/v' \rightarrow v)\} v := [e] \{\exists v'. (e' \mapsto v) * (r/v'' \rightarrow v)\}},$$

where v , v' , and v'' are distinct, v' , $v'' \notin \text{FV}(e)$, $v \notin \text{FV}(r)$, and e' denotes $e/v \rightarrow v'$.

The backward-reasoning form (LKBR):

$$\frac{}{\{\exists v''. (e \hookrightarrow v'') \wedge p''\} v := [e] \{p\}},$$

where $v'' \notin \text{FV}(e) \cup (\text{FV}(p) - \{v\})$, and p'' denotes $p/v \rightarrow v''$.

The global noninterfering form (LKNIG):

$$\frac{}{\{\exists v''. (e \mapsto v'') * p''\} v := [e] \{(e \mapsto v) * p\}},$$

where $v \notin \text{FV}(e)$, $v'' \notin \text{FV}(e) \cup (\text{FV}(p) - \{v\})$, and p'' denotes $p/v \rightarrow v''$. As a special case,

$$\frac{}{\{\exists v. (e \mapsto v) * p\} v := [e] \{(e \mapsto v) * p\}},$$

where $v \notin \text{FV}(e)$.

Inference Rules for Lookup (continued)

To derive (LKG) from (LKL), we use the frame rule (FR):

$$\begin{array}{l}
 \{(v = v' \wedge (e \mapsto v'')) * (r/v' \rightarrow v)\} \\
 \{(v = v' \wedge (e \mapsto v'')) * r\} \\
 v := [e] \\
 \{(v = v'' \wedge (e' \mapsto v'')) * r\} \\
 \{(v = v'' \wedge (e' \mapsto v)) * (r/v'' \rightarrow v)\},
 \end{array}$$

and the rule for existential quantification (EQ):

$$\begin{array}{l}
 \{\exists v''. (e \mapsto v'') * (r/v' \rightarrow v)\} \\
 \{\exists v', v''. (v = v' \wedge (e \mapsto v'')) * (r/v' \rightarrow v)\} \\
 v := [e] \\
 \{\exists v', v''. (v = v'' \wedge (e' \mapsto v)) * (r/v'' \rightarrow v)\} \\
 \{\exists v'. (e' \mapsto v) * (r/v'' \rightarrow v)\}.
 \end{array}$$

To derive (LKBR) from (LKG), we take r to be $(e' \mapsto v'') \multimap p''$ and use the axiom schemata $(e \hookrightarrow e') \wedge p \Rightarrow (e \mapsto e') * ((e \mapsto e') \multimap p)$ and $q * (q \multimap p) \Rightarrow p$. (In the first line, we can rename v'' to be any variable not in $\text{FV}(e) \cup (\text{FV}(p) - \{v\})$.)

$$\begin{array}{l}
 \{\exists v''. (e \hookrightarrow v'') \wedge p''\} \\
 \{\exists v''. (e \mapsto v'') * ((e \mapsto v'') \multimap p'')\} \\
 v := [e] \\
 \{\exists v'. (e' \mapsto v) * ((e' \mapsto v) \multimap p)\} \\
 \{\exists v'. p\} \\
 \{p\}.
 \end{array}$$

Inference Rules for Lookup (continued)

To derive (LKL) from (LKBR), we take p to be $v = v'' \wedge (e' \mapsto v'')$, after renaming v'' to \hat{v} :

$$\begin{aligned}
& \{v = v' \wedge (e \mapsto v'')\} \\
& \{v = v' \wedge (e \hookrightarrow v'') \wedge (e' \mapsto v'')\} \\
& \{(e \hookrightarrow v'') \wedge (e' \mapsto v'')\} \\
& \{\exists \hat{v}. (e \hookrightarrow v'') \wedge \hat{v} = v'' \wedge (e' \mapsto v'')\} \\
& \{\exists \hat{v}. (e \hookrightarrow \hat{v}) \wedge \hat{v} = v'' \wedge (e' \mapsto v'')\} \\
& v := [e] \\
& \{v = v'' \wedge (e' \mapsto v'')\}.
\end{aligned}$$

To derive (LKNIG) from (LKG), suppose v and v'' are distinct variables, $v \notin \text{FV}(e)$, and $v'' \notin \text{FV}(e) \cup \text{FV}(p)$. We take v' to be a variable distinct from v and v'' that does not occur free in e or p , and r to be $p'' = p/v \rightarrow v''$. Then

$$\begin{aligned}
& \{\exists v''. (e \mapsto v'') * p''\} \\
& \{\exists v''. (e \mapsto v'') * (p''/v' \rightarrow v)\} \\
& v := [e] \\
& \{\exists v'. (e' \mapsto v) * (p''/v'' \rightarrow v)\} \\
& \{\exists v'. (e \mapsto v) * p\} \\
& \{(e \mapsto v) * p\}.
\end{aligned}$$

In the first line, we can rename v'' to be any variable not in $\text{FV}(e) \cup (\text{FV}(p) - \{v\})$.

Weakest Preconditions for the New Commands

For each of the new heap-manipulating commands, the backward-reasoning form gives both the weakest liberal precondition and (since these commands always terminate) the weakest precondition.

Mutation:

$$\mathbf{wp}([e] := e', p) = (e \mapsto -) * ((e \mapsto e') \multimap p).$$

Allocation:

$$\mathbf{wp}(v := \mathbf{cons}(e_0, \dots, e_{n-1}), p) = \forall v''. (v'' \mapsto e_0, \dots, e_{n-1}) \multimap p'',$$

where v'' is distinct from v , $v'' \notin \mathbf{FV}(e_0, \dots, e_{n-1}, p)$, and p'' denotes $p/v \rightarrow v''$.

Disposal:

$$\mathbf{wp}(\mathbf{dispose } e, r) = (e \mapsto -) * r.$$

Lookup:

$$\mathbf{wp}(v := [e], p) = \exists v''. (e \hookrightarrow v'') \wedge p'',$$

where $v'' \notin \mathbf{FV}(e) \cup (\mathbf{FV}(p) - \{v\})$, and p'' denotes $p/v \rightarrow v''$.

Similar rules hold for **wlp**.

An Annotated Specification Revisited

{emp}

$x := \mathbf{cons}(a, a) ;$ (CONSNIL)

$\{x \mapsto a, a\}$ i.e., $\{x \mapsto a * x + 1 \mapsto a\}$

$y := \mathbf{cons}(b, b) ;$ (CONSNIG)

$\{(x \mapsto a, a) * (y \mapsto b, b)\}$

i.e., $\{x \mapsto a * x + 1 \mapsto a * y \mapsto b * y + 1 \mapsto b\}$

$(p/v \rightarrow e \Rightarrow \exists v. p)$

$\{(x \mapsto a, -) * (y \mapsto b, b)\}$

i.e., $\{x \mapsto a * (\exists a. x + 1 \mapsto a) * y \mapsto b * y + 1 \mapsto b\}$

$[x + 1] := y - x ;$ (MUG)

$\{(x \mapsto a, y - x) * (y \mapsto b, b)\}$

i.e., $\{x \mapsto a * x + 1 \mapsto y - x * y \mapsto b * y + 1 \mapsto b\}$

$(p/v \rightarrow e \Rightarrow \exists v. p)$

$\{(x \mapsto a, y - x) * (y \mapsto b, -)\}$

i.e., $\{x \mapsto a * x + 1 \mapsto y - x * y \mapsto b * (\exists b. y + 1 \mapsto b)\}$

$[y + 1] := x - y ;$ (MUG)

$\{(x \mapsto a, y - x) * (y \mapsto b, x - y)\}$

i.e., $\{x \mapsto a * x + 1 \mapsto y - x * y \mapsto b * y + 1 \mapsto x - y\}$

$(p/v \rightarrow e \Rightarrow \exists v. p)$

$\{\exists o. (x \mapsto a, o) * (x + o \mapsto b, -o)\}$

i.e., $\{x \mapsto a * x + 1 \mapsto o * x + o \mapsto b * x + o + 1 \mapsto -o\}$