Class Notes for CS 818A4 - Spring 2004

SEPARATION LOGIC

2. Expressions, Commands, and Assertions

John C. Reynolds
Department of Computer Science
Carnegie Mellon University

Revised March 24, 2004

Syntax of the Programming Language

$$
\begin{aligned}
\langle\text{exp}\rangle ::=\ & 0 \mid 1 \mid 2 \mid \cdots \\
& \mid \langle\text{var}\rangle \mid -\langle\text{exp}\rangle \mid \langle\text{exp}\rangle + \langle\text{exp}\rangle \mid \langle\text{exp}\rangle - \langle\text{exp}\rangle \\
& \mid \langle\text{exp}\rangle \times \langle\text{exp}\rangle \mid \cdots
\end{aligned}
$$

$$
\begin{aligned}
\langle\text{boolexp}\rangle ::=\ & \textbf{true} \mid \textbf{false} \\
& \mid \langle\text{exp}\rangle = \langle\text{exp}\rangle \mid \langle\text{exp}\rangle \neq \langle\text{exp}\rangle \mid \cdots \\
& \mid \neg\,\langle\text{boolexp}\rangle \mid \langle\text{boolexp}\rangle \wedge \langle\text{boolexp}\rangle \mid \langle\text{boolexp}\rangle \vee \langle\text{boolexp}\rangle \\
& \mid \langle\text{boolexp}\rangle \Rightarrow \langle\text{boolexp}\rangle \mid \langle\text{boolexp}\rangle \Leftrightarrow \langle\text{boolexp}\rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle\text{comm}\rangle ::=\ & \langle\text{var}\rangle := \langle\text{exp}\rangle \mid \textbf{skip} \mid \langle\text{comm}\rangle\ ;\ \langle\text{comm}\rangle \\
& \mid \textbf{if}\ \langle\text{boolexp}\rangle\ \textbf{then}\ \langle\text{comm}\rangle\ \textbf{else}\ \langle\text{comm}\rangle \\
& \mid \textbf{while}\ \langle\text{boolexp}\rangle\ \textbf{do}\ \langle\text{comm}\rangle \\
& \mid \textbf{newvar}\ \langle\text{var}\rangle\ \textbf{in}\ \langle\text{comm}\rangle \\
& \mid \textbf{newvar}\ \langle\text{var}\rangle := \langle\text{exp}\rangle\ \textbf{in}\ \langle\text{comm}\rangle \\
& \mid \langle\text{var}\rangle := \textbf{cons}(\langle\text{exp}\rangle, \ldots, \langle\text{exp}\rangle) \\
& \mid \langle\text{var}\rangle := [\langle\text{exp}\rangle] \\
& \mid [\langle\text{exp}\rangle] := \langle\text{exp}\rangle \\
& \mid \textbf{dispose}\,\langle\text{exp}\rangle
\end{aligned}
$$

Semantic Functions for Expressions

$$[\![e \in \langle \text{exp} \rangle]\!]_{\text{exp}} \in (\textstyle\bigcup_{V \overset{\text{fin}}{\supseteq} \text{FV}(e)} \text{Stores}_V) \to \mathbf{Z}.$$

$$[\![0]\!]_{\text{exp}} s = 0 \qquad \text{(and similarly for 1, 2, \ldots),}$$

$$[\![v]\!]_{\text{exp}} s = sv,$$

$$[\![-e]\!]_{\text{exp}} s = -[\![e]\!]_{\text{exp}} s,$$

$$[\![e_0 + e_1]\!]_{\text{exp}} s = [\![e_0]\!]_{\text{exp}} s + [\![e_1]\!]_{\text{exp}} s$$

$$\text{(and similarly for } -, \times, \ldots\text{).}$$

$$[\![b \in \langle \text{boolexp} \rangle]\!]_{\text{boolexp}} \in (\textstyle\bigcup_{V \overset{\text{fin}}{\supseteq} \text{FV}(b)} \text{Stores}_V) \to \mathbf{B}.$$

$$[\![\mathbf{true}]\!]_{\text{boolexp}} s = \mathbf{true},$$

$$[\![\mathbf{false}]\!]_{\text{boolexp}} s = \mathbf{false},$$

$$[\![e_0 = e_1]\!]_{\text{boolexp}} s = ([\![e_0]\!]_{\text{exp}} s = [\![e_1]\!]_{\text{exp}} s)$$

$$\text{(and similarly for } \neq, \ldots\text{),}$$

$$[\![\neg\, b]\!]_{\text{boolexp}} s = \neg [\![b]\!]_{\text{boolexp}} s,$$

$$[\![b_0 \wedge b_1]\!]_{\text{boolexp}} s = [\![b_0]\!]_{\text{boolexp}} s \wedge [\![b_1]\!]_{\text{boolexp}} s$$

$$\text{(and similarly for } \vee, \Rightarrow, \Leftrightarrow\text{).}$$

For example:

$$\begin{aligned}
[\![\mathsf{x} + \mathsf{y} = \mathsf{x}]\!]_{\text{boolexp}} s &= ([\![\mathsf{x} + \mathsf{y}]\!]_{\text{exp}} s = [\![\mathsf{x}]\!]_{\text{exp}} s) \\
&= ([\![\mathsf{x}]\!]_{\text{exp}} s + [\![\mathsf{y}]\!]_{\text{exp}} s = [\![\mathsf{x}]\!]_{\text{exp}} s) \\
&= (s\,\mathsf{x} + s\,\mathsf{y} = s\,\mathsf{x}) \\
&= (s\,\mathsf{y} = 0).
\end{aligned}$$

Free Variables

For any phrase p,

$$\mathrm{FV}(p) = \text{“The variables occurring free in } p\text{”}.$$

There are no binding constructions in expressions or boolean expressions, so that for these phrases $\mathrm{FV}(e)$ is the set of all variables occurring in e.

Substitution

For any phrase p such that $\mathrm{FV}(p) \subseteq \{v_1, \ldots, v_n\}$, we write

$$p/v_1 \to e_1, \ldots, v_n \to e_n$$

to denote the phrase obtained from p by simultaneously substituting each expression e_i for the variable v_i, (When there are bound variables in p, they will be renamed to avoid capture.)

The Substitution Law for Expressions

Proposition 1 *Let δ abbreviate the substitution*

$$v_1 \to e_1, \ldots, v_n \to e_n,$$

let s be a store such that $\mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n) \subseteq \mathrm{dom}\, s$, *and let*

$$\widehat{s} = [\, v_1\colon [\![e_1]\!]_{\mathrm{exp}} s \mid \ldots \mid v_n\colon [\![e_n]\!]_{\mathrm{exp}} s \,].$$

If e is an expression (or boolean expression) such that $\mathrm{FV}(e) \subseteq \{v_1, \ldots, v_n\}$, *then*

$$[\![e/\delta]\!]_{\mathrm{exp}} s = [\![e]\!]_{\mathrm{exp}} \widehat{s}.$$

Free Variables of Commands

$$\mathrm{FV}(\mathbf{newvar}\ v\ \mathbf{in}\ c) = \mathrm{FV}(c) - \{v\}$$

$$\mathrm{FV}(\mathbf{newvar}\ v := e\ \mathbf{in}\ c) = \mathrm{FV}(e) \cup (\mathrm{FV}(c) - \{v\}),$$

and for other forms of commands,

$$\mathrm{FV}(\cdots p_1 \cdots p_n \cdots) = \mathrm{FV}(p_1) \cup \cdots \cup \mathrm{FV}(p_n).$$

Compositional Large-Step Semantics of Commands

$$[\![c \in \langle \mathrm{comm} \rangle]\!]_{\mathrm{comm}} \subseteq \bigcup_{V \overset{\mathrm{fin}}{\supseteq} \mathrm{FV}(c)} \mathrm{States}_V \times (\mathrm{States}_V \cup \{\mathbf{abort}, \perp\})$$

is the relation such that

- $(s, h)\, [\![c]\!]_{\mathrm{comm}}\, (s', h')$ iff, starting in state (s, h), there is a finite execution of c that terminates in state (s', h').
- $(s, h)\, [\![c]\!]_{\mathrm{comm}}\, \mathbf{abort}$ iff, starting in state (s, h), there is a finite execution of c that terminates with **abort**.
- $(s, h)\, [\![c]\!]_{\mathrm{comm}}\, \perp$ 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 $(s, h)\, [\![c]\!]_{\mathrm{comm}}\, (s', h')$, then $\mathrm{dom}\, s = \mathrm{dom}\, s'$.

We write $[\, f \mid x\colon y\,]$ to denote the function such that

$$\mathrm{dom}[\, f \mid x\colon y\,] = \mathrm{dom}\, f \cup \{x\}$$

$$[\, f \mid x\colon y\,](x) = y \qquad \text{and} \qquad [\, f \mid x\colon y\,](x') = f(x') \text{ when } x' \neq x.$$

Inference Rules for the Meanings of Commands

- Assignment

$$\frac{}{(s,h)\,[\![v := e]\!]_{\mathrm{comm}}\,([\,s \mid v\colon [\![e]\!]_{\mathrm{exp}} s\,], h),}$$

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

- **skip**

$$\frac{}{(s,h)\,[\![\mathbf{skip}]\!]_{\mathrm{comm}}\,(s,h).}$$

- Sequencing

$$\frac{(s,h)\,[\![c_0]\!]_{\mathrm{comm}}\,(s',h') \qquad (s',h')\,[\![c_1]\!]_{\mathrm{comm}}\,\tau''}{(s,h)\,[\![c_0\,;c_1]\!]_{\mathrm{comm}}\,\tau''}$$

$$\frac{(s,h)\,[\![c_0]\!]_{\mathrm{comm}}\,\mathbf{abort}}{(s,h)\,[\![c_0\,;c_1]\!]_{\mathrm{comm}}\,\mathbf{abort}} \qquad \frac{(s,h)\,[\![c_0]\!]_{\mathrm{comm}}\,\bot}{(s,h)\,[\![c_0\,;c_1]\!]_{\mathrm{comm}}\,\bot.}$$

- Conditional

$$\frac{[\![b]\!]_{\mathrm{boolexp}} s = \mathbf{true} \qquad (s,h)\,[\![c_0]\!]_{\mathrm{comm}}\,\tau'}{(s,h)\,[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]_{\mathrm{comm}}\,\tau'}$$

$$\frac{[\![b]\!]_{\mathrm{boolexp}} s = \mathbf{false} \qquad (s,h)\,[\![c_1]\!]_{\mathrm{comm}}\,\tau'}{(s,h)\,[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]_{\mathrm{comm}}\,\tau',}$$

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

while Commands

- **while**

$$\frac{\begin{array}{l}\forall i \in 0 \textbf{ to } n-1 : \; [\![b]\!]_{\text{boolexp}} s_i = \textbf{true} \quad (s_i, h_i)\, [\![c]\!]_{\text{comm}}\, (s_{i+1}, h_{i+1}) \\ [\![b]\!]_{\text{boolexp}} s_n = \textbf{false}\end{array}}{(s_0, h_0)\, [\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{\text{comm}}\, (s_n, h_n)}$$

$$\frac{\begin{array}{l}\forall i \in 0 \textbf{ to } n-1 : \; [\![b]\!]_{\text{boolexp}} s_i = \textbf{true} \quad (s_i, h_i)\, [\![c]\!]_{\text{comm}}\, (s_{i+1}, h_{i+1}) \\ [\![b]\!]_{\text{boolexp}} s_n = \textbf{true} \quad (s_n, h_n)\, [\![c]\!]_{\text{comm}}\, \textbf{abort}\end{array}}{(s_0, h_0)\, [\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{\text{comm}}\, \textbf{abort}}$$

$$\frac{\begin{array}{l}\forall i \in 0 \textbf{ to } n-1 : \; [\![b]\!]_{\text{boolexp}} s_i = \textbf{true} \quad (s_i, h_i)\, [\![c]\!]_{\text{comm}}\, (s_{i+1}, h_{i+1}) \\ [\![b]\!]_{\text{boolexp}} s_n = \textbf{true} \quad (s_n, h_n)\, [\![c]\!]_{\text{comm}}\, \bot\end{array}}{(s_0, h_0)\, [\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{\text{comm}}\, \bot}$$

$$\frac{\forall i \in 0 \textbf{ to } \infty : \; [\![b]\!]_{\text{boolexp}} s_i = \textbf{true} \quad (s_i, h_i)\, [\![c]\!]_{\text{comm}}\, (s_{i+1}, h_{i+1})}{(s_0, h_0)\, [\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{\text{comm}}\, \bot,}$$

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

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)

$$\frac{([\, s \mid v\colon n\,], h)\, [\![c]\!]_{\text{comm}}\, (s', h')}{(s, h)\, [\![\mathbf{newvar}\ v\ \mathbf{in}\ c]\!]_{\text{comm}}\, (s' \rceil \operatorname{dom} s, h'),}$$

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

$$\frac{([\, s \mid v\colon n\,], h)\, [\![c]\!]_{\text{comm}}\, (s', h')}{(s, h)\, [\![\mathbf{newvar}\ v\ \mathbf{in}\ c]\!]_{\text{comm}}\, ([\, s' \mid v\colon s\, v\,], h'),}$$

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

$$\frac{([\, s \mid v\colon n\,], h)\, [\![c]\!]_{\text{comm}}\, \mathbf{abort}}{(s, h)\, [\![\mathbf{newvar}\ v\ \mathbf{in}\ c]\!]_{\text{comm}}\, \mathbf{abort}}$$

$$\frac{([\, s \mid v\colon n\,], h)\, [\![c]\!]_{\text{comm}}\, \bot}{(s, h)\, [\![\mathbf{newvar}\ v\ \mathbf{in}\ c]\!]_{\text{comm}}\, \bot.}$$

Variable Declarations (continued)

- **newvar** (with initialization)

$$\frac{([\, s \mid v\colon [\![e]\!]_{\mathrm{exp}} s \,], h) \, [\![c]\!]_{\mathrm{comm}} \, (s', h')}{(s, h) \, [\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]_{\mathrm{comm}} \, (s' \rceil \operatorname{dom} s, h'),}$$

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

$$\frac{([\, s \mid v\colon [\![e]\!]_{\mathrm{exp}} s \,], h) \, [\![c]\!]_{\mathrm{comm}} \, (s', h')}{(s, h) \, [\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]_{\mathrm{comm}} \, ([\, s' \mid v\colon s\, v \,], h'),}$$

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

$$\frac{([\, s \mid v\colon [\![e]\!]_{\mathrm{exp}} s \,], h) \, [\![c]\!]_{\mathrm{comm}} \, \mathbf{abort}}{(s, h) \, [\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]_{\mathrm{comm}} \, \mathbf{abort}}$$

$$\frac{([\, s \mid v\colon [\![e]\!]_{\mathrm{exp}} s \,], h) \, [\![c]\!]_{\mathrm{comm}} \, \bot}{(s, h) \, [\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]_{\mathrm{comm}} \, \bot.}$$

Semantics of the New Commands

- Allocation

$$\frac{}{\begin{array}{l}(s,h)\,[\![v := \mathbf{cons}(e_0,\ldots,e_{n-1})]\!]_{\mathrm{comm}}\\ \qquad ([\,s \mid v{:}\,\ell\,],[\,h \mid \ell{:}\,[\![e_0]\!]_{\exp}s \mid \ldots \mid \ell+n-1{:}\,[\![e_{n-1}]\!]_{\exp}s\,]),\end{array}}$$

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

- Lookup

$$\frac{}{(s,h)\,[\![v := [e]]\!]_{\mathrm{comm}}\,([\,s \mid v{:}\,h([\![e]\!]_{\exp}s)\,],h),}$$

  when $[\![e]\!]_{\exp}s \in \mathrm{dom}\,h$ and $\{v\} \cup \mathrm{FV}(e) \subseteq \mathrm{dom}\,s$,

$$\frac{}{(s,h)\,[\![v := [e]]\!]_{\mathrm{comm}}\,\mathbf{abort},}$$

  when $[\![e]\!]_{\exp}s \notin \mathrm{dom}\,h$ and $\{v\} \cup \mathrm{FV}(e) \subseteq \mathrm{dom}\,s$.

More New Commands

- Mutation

$$\overline{(s,h)\,[\![[e] := e']\!]_{\mathrm{comm}}\,(s, [\, h \mid [\![e]\!]_{\mathrm{exp}}s \colon [\![e']\!]_{\mathrm{exp}}s\,]),}$$

when $[\![e]\!]_{\mathrm{exp}}s \in \mathrm{dom}\,h$ and $\mathrm{FV}(e) \cup \mathrm{FV}(e') \subseteq \mathrm{dom}\,s$,

$$\overline{(s,h)\,[\![[e] := e']\!]_{\mathrm{comm}}\,\mathbf{abort},}$$

when $[\![e]\!]_{\mathrm{exp}}s \notin \mathrm{dom}\,h$ and $\mathrm{FV}(e) \cup \mathrm{FV}(e') \subseteq \mathrm{dom}\,s$.

- Disposal

$$\overline{(s,h)\,[\![\mathbf{dispose}\ e]\!]_{\mathrm{comm}}\,(s, h\rceil(\mathrm{dom}\,h - \{[\![e]\!]_{\mathrm{exp}}s\})),}$$

when $[\![e]\!]_{\mathrm{exp}}s \in \mathrm{dom}\,h$ and $\mathrm{FV}(e) \subseteq \mathrm{dom}\,s$,

$$\overline{(s,h)\,[\![\mathbf{dispose}\ e]\!]_{\mathrm{comm}}\,\mathbf{abort},}$$

when $[\![e]\!]_{\mathrm{exp}}s \notin \mathrm{dom}\,h$ and $\mathrm{FV}(e) \subseteq \mathrm{dom}\,s$.

Totality

Proposition 2 *Our semantics is* total: *If* $(s, h) \in \mathrm{States}_V$ *for some* $V \overset{\text{fin}}{\supseteq} \mathrm{FV}(c)$*, then there is at least one* $\tau' \in \mathrm{States}_V \cup \{\mathbf{abort}, \bot\}$ *such that* $(s, h) \llbracket c \rrbracket_{\mathrm{comm}} \tau'$*.*

Some Definitions

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

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

By Proposition 2, if c must terminate normally at (s, h), then there is at least one state (s', h') such that $(s, h) \llbracket c \rrbracket_{\mathrm{comm}} (s', h')$.

Modified Variables

For a command c, the set $\mathrm{MD}(c)$, whose members are called the *variables modified by* c, is defined by structural induction:

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

Proposition 3 *Suppose* $v \in \mathrm{dom}(s)$ *and* $v \notin \mathrm{MD}(c)$*. If*

$$(s, h)\ [\![c]\!]_{\mathrm{comm}}\ (s', h'),$$

then $s'v = s\,v$*.*

Substitution in Commands

Let c be a command such that $\mathrm{FV}(c) \subseteq \{v_1, \ldots, v_n\}$, and let δ denote a substitution

$$v_1 \rightarrow e_1, \ldots, v_n \rightarrow e_n$$

such that, when $v_i \in \mathrm{MD}(c)$, e_i is a variable. When c is a variable declaration, c/δ is defined by

$$(\mathbf{newvar}\ v\ \mathbf{in}\ c)/\delta = \mathbf{newvar}\ v_{\mathrm{new}}\ \mathbf{in}\ (c/[\delta|v \rightarrow v_{\mathrm{new}}])$$

$$(\mathbf{newvar}\ v := e\ \mathbf{in}\ c)/\delta = \mathbf{newvar}\ v_{\mathrm{new}} := (e/\delta)\ \mathbf{in}\ (c/[\delta|v \rightarrow v_{\mathrm{new}}]),$$

where $v_{\mathrm{new}} \notin \mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n)$, and $[\delta|v \rightarrow v_{\mathrm{new}}]$ is the substitution that is like δ except that it maps v into v_{new}.

For other forms of commands,

$$(\cdots p_1 \cdots p_n \cdots)/\delta = \cdots (p_1/\delta) \cdots (p_n/\delta) \cdots .$$

The Substitution Law for Commands

Proposition 4 *Let* c *be a command such that* $\mathrm{FV}(c) \subseteq \{v_1, \ldots, v_n\}$, *and let* δ *be a substitution*

$$v_1 \to e_1, \ldots, v_n \to e_n$$

such that, when $v_i \in \mathrm{MD}(c)$, e_i *is a variable that does not occur in any other* e_j.

For any store s *such that* $\mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n) \subseteq \mathrm{dom}\, s$, *let*

$$\widehat{s} = [\, v_1\colon [\![e_1]\!]_{\exp} s \mid \ldots \mid v_n\colon [\![e_n]\!]_{\exp} s \,].$$

Then

- *If* $(s, h)\, [\![c/\delta]\!]_{\mathrm{comm}}\, (s', h')$, *then* $(\widehat{s}, h)\, [\![c]\!]_{\mathrm{comm}}\, (\widehat{s'}, h')$.
- *If* $(s, h)\, [\![c/\delta]\!]_{\mathrm{comm}}\, \mathbf{abort}$, *then* $(\widehat{s}, h)\, [\![c]\!]_{\mathrm{comm}}\, \mathbf{abort}$.
- *If* $(s, h)\, [\![c/\delta]\!]_{\mathrm{comm}}\, \bot$, *then* $(\widehat{s}, h)\, [\![c]\!]_{\mathrm{comm}}\, \bot$.

Splitting the Heap

We define

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

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 5 *Suppose $\hat{h} \subseteq h$. Then*

- *If $(s, h)\, [\![c]\!]_{\text{comm}}\, \textbf{abort}$, then*

$$(s, h - \hat{h})\, [\![c]\!]_{\text{comm}}\, \textbf{abort}.$$

- *If $(s, h)\, [\![c]\!]_{\text{comm}}\, (s', h')$, then*

$$(s, h - \hat{h})\, [\![c]\!]_{\text{comm}}\, \textbf{abort}$$
$$\textit{or } \hat{h} \subseteq h' \textit{ and } (s, h - \hat{h})\, [\![c]\!]_{\text{comm}}\, (s', h' - \hat{h}).$$

- *If $(s, h)\, [\![c]\!]_{\text{comm}}\, \perp$, then*

$$(s, h - \hat{h})\, [\![c]\!]_{\text{comm}}\, \textbf{abort}$$
$$\textit{or } (s, h - \hat{h})\, [\![c]\!]_{\text{comm}}\, \perp.$$

Proof of Proposition 5

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

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

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

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

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

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

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

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

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

  and $h' = [\, h \mid \llbracket e \rrbracket_{\text{exp}} s \colon \llbracket e' \rrbracket_{\text{exp}} s \,] = [\, (h - \hat{h}) \cdot \hat{h} \mid \llbracket e \rrbracket_{\text{exp}} s \colon \llbracket e' \rrbracket_{\text{exp}} s \,]$ $= [\, h - \hat{h} \mid \llbracket e \rrbracket_{\text{exp}} s \colon \llbracket e' \rrbracket_{\text{exp}} s \,] \cdot \hat{h}$. 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 5 (continued)

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

$$(s, h)[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]_{\text{comm}}\tau. \tag{1}$$

Since the final step in deriving the triple (1) 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** $n - 1$, $[\![b]\!]_{\text{boolexp}} s_i = \mathbf{true}$ and

$$(s_i, h_i)\,[\![c]\!]_{\text{comm}}\,(s_{i+1}, h_{i+1}).$$

(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 5:

If $\hat{h} \subseteq h$ and $(s, h)\,[\![c]\!]_{\text{comm}}\,(s', h')$, then

$(s, h - \hat{h})\,[\![c]\!]_{\text{comm}}\,\mathbf{abort}$
or $\hat{h} \subseteq h'$ and $(s, h - \hat{h})\,[\![c]\!]_{\text{comm}}\,(s', h' - \hat{h})$.

Since the condition $\hat{h} \subseteq h$ here reproduces itself as $\hat{h} \subseteq h'$, we can repeatedly apply the second part of Proposition 5 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** $k - 1$,

$$(s_i, h_i - \hat{h})\,[\![c]\!]_{\text{comm}}\,(s_{i+1}, h_{i+1} - \hat{h})$$

Proof of Proposition 5 (continued)

and either k is n (which may be ∞) or

$$(s_k, h_k - \hat{h})\, [\![c]\!]_{\text{comm}}\, \textbf{abort}.$$

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

$$(s, h - \hat{h})\, [\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{\text{comm}}\, \textbf{abort}.$$

Otherwise, $k = n$, and our argument depends on which **while** rule was used as the last step in deriving the triple (1). If the first rule was used, then $\tau = (s_n, h_n)$ and $[\![b]\!]_{\text{boolexp}} s_n = \textbf{false}$. Then we can use the first rule to obtain

$$(s, h - \hat{h})\, [\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{\text{comm}}\, (s_n, h_n - \hat{h}).$$

If the second **while** rule was used as the last step in deriving (1), then $\tau = \textbf{abort}$, $[\![b]\!]_{\text{boolexp}} s_n = \textbf{true}$, and

$$(s_n, h_n)\, [\![c]\!]_{\text{comm}}\, \textbf{abort},$$

so by the induction hypothesis,

$$(s_n, h_n - \hat{h})\, [\![c]\!]_{\text{comm}}\, \textbf{abort},$$

and the second rule gives

$$(s, h - \hat{h})\, [\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{\text{comm}}\, \textbf{abort}.$$

Proof of Proposition 5 (continued)

If the third **while** rule was used as the last step in deriving (1), then $\tau = \perp$, $[\![b]\!]_{\text{boolexp}} s_n = \textbf{true}$, and

$$(s_n, h_n)\, [\![c]\!]_{\text{comm}}\, \perp,$$

so by the induction hypothesis,

$$(s_n, h_n - \hat{h})\, [\![c]\!]_{\text{comm}}\, \textbf{abort} \text{ or } (s_n, h_n - \hat{h})\, [\![c]\!]_{\text{comm}}\, \perp,$$

and the second or third rule gives

$$(s, h - \hat{h})\, [\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{\text{comm}}\, \textbf{abort} \text{ or } (s, h - \hat{h})\, [\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{\text{comm}}\, \perp.$$

If the fourth **while** rule was used as the last step in deriving (1), then $\tau = \perp$ and $n = \infty$. Then the fourth rule gives

$$(s, h - \hat{h})\, [\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{\text{comm}}\, \perp.$$

END OF PROOF

Putting It Positively

Proposition 5 is equivalent to the conjunction of the following propositions:

Safety Monotonicity

Proposition 6 *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 7 *If*

$$\hat{h} \subseteq h, \; c \text{ is safe at } (s, h - \hat{h}), \text{ and } (s, h) \, [\![c]\!]_{\text{comm}} \, (s', h'),$$

then

$$\hat{h} \subseteq h' \text{ and } (s, h - \hat{h}) \, [\![c]\!]_{\text{comm}} \, (s', h' - \hat{h}).$$

Syntax of Assertions

$$
\begin{aligned}
\langle\text{assert}\rangle ::= {} & \langle\text{boolexp}\rangle \\
& \mid \neg\,\langle\text{assert}\rangle \mid \langle\text{assert}\rangle \wedge \langle\text{assert}\rangle \mid \langle\text{assert}\rangle \vee \langle\text{assert}\rangle \\
& \mid \langle\text{assert}\rangle \Rightarrow \langle\text{assert}\rangle \mid \langle\text{assert}\rangle \Leftrightarrow \langle\text{assert}\rangle \\
& \mid \forall\langle\text{var}\rangle.\ \langle\text{assert}\rangle \mid \exists\langle\text{var}\rangle.\ \langle\text{assert}\rangle \\
& \mid \mathbf{emp} \\
& \mid \langle\text{exp}\rangle \mapsto \langle\text{exp}\rangle \\
& \mid \langle\text{assert}\rangle \ * \ \langle\text{assert}\rangle \\
& \mid \langle\text{assert}\rangle \mathrel{-\!\!*} \langle\text{assert}\rangle
\end{aligned}
$$

The Semantics of Assertions

$$[\![p \in \langle\text{assert}\rangle]\!]_{\text{assert}} \in (\bigcup_{V \overset{\text{fin}}{\supseteq} \text{FV}(p)} \text{States}_V) \to \mathbf{B}.$$

Instead of $[\![p]\!]_{\text{assert}}(s,h) = \mathbf{true}$, however, we write $s,h \vDash p$.

$$\begin{aligned}
s,h \vDash b &\text{ iff } [\![b]\!]_{\text{boolexp}} s = \mathbf{true},\\
s,h \vDash \neg\, p &\text{ iff } s,h \vDash p \text{ is false},\\
s,h \vDash p_0 \wedge p_1 &\text{ iff } s,h \vDash p_0 \text{ and } s,h \vDash p_1\\
&\qquad\qquad\qquad\text{(and similarly for } \vee,\ \Rightarrow,\ \Leftrightarrow),\\
s,h \vDash \forall v.\ p &\text{ iff } \forall x \in \mathbf{Z}.\ [\, s \mid v{:}\, x\,], h \vDash p,\\
s,h \vDash \exists v.\ p &\text{ iff } \exists x \in \mathbf{Z}.\ [\, s \mid v{:}\, x\,], h \vDash p,\\
s,h \vDash \mathbf{emp} &\text{ iff } \operatorname{dom} h = \{\},\\
s,h \vDash e \mapsto e' &\text{ iff } \operatorname{dom} h = \{[\![e]\!]_{\text{exp}} s\} \text{ and } h([\![e]\!]_{\text{exp}} s) = [\![e']\!]_{\text{exp}} s,\\
s,h \vDash p_0 \ast p_1 &\text{ iff } \exists h_0, h_1.\ h_0 \perp h_1 \text{ and } h_0 \cdot h_1 = h \text{ and}\\
&\qquad\qquad\qquad s,h_0 \vDash p_0 \text{ and } s,h_1 \vDash p_1,\\
s,h \vDash p_0 -\!\!\ast\, p_1 &\text{ iff } \forall h'.\ (h' \perp h \text{ and } s,h' \vDash p_0) \text{ implies}\\
&\qquad\qquad\qquad s, h \cdot h' \vDash p_1.
\end{aligned}$$

Examples

$$\begin{aligned}
s, h \vDash \mathsf{x} + \mathsf{y} = \mathsf{x} \ &\text{iff}\ [\![\mathsf{x} + \mathsf{y} = \mathsf{x}]\!]_{\text{boolexp}} s \\
&\text{iff}\ s\,\mathsf{y} = 0. \\
s, h \vDash \forall \mathsf{x}.\ \mathsf{x} + \mathsf{y} = \mathsf{x} \ &\text{iff}\ \forall n \in \mathbf{Z}.\ [\, s \mid \mathsf{x}\colon n\,], h \vDash \mathsf{x} + \mathsf{y} = \mathsf{x} \\
&\text{iff}\ \forall n \in \mathbf{Z}.\ [\, s \mid \mathsf{x}\colon n\,]\,\mathsf{y} = 0 \\
&\text{iff}\ \forall n \in \mathbf{Z}.\ s\,\mathsf{y} = 0 \\
&\text{iff}\ s\,\mathsf{y} = 0. \\
s, h \vDash \mathsf{x} \mapsto \mathsf{y} \ &\text{iff}\ \operatorname{dom} h = \{s\,\mathsf{x}\} \text{ and } h(s\,\mathsf{x}) = s\,\mathsf{y} \\
&\text{iff}\ h = \{\langle s\,\mathsf{x}, s\,\mathsf{y}\rangle\}.
\end{aligned}$$

Some Abbreviations

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

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

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

$$e \hookrightarrow e_0, \ldots, e_{n-1} \stackrel{\text{def}}{=} e \hookrightarrow e_0 \ * \ \cdots \ * \ e + n - 1 \hookrightarrow e_{n-1}$$
$$\text{iff } e \mapsto e_0, \ldots, e_{n-1} \ * \ \mathbf{true}.$$

Examples

$$s, h \vDash \mathsf{x} \mapsto \mathsf{y} \text{ iff } \operatorname{dom} h = \{s\,\mathsf{x}\} \text{ and } h(s\,\mathsf{x}) = s\,\mathsf{y}$$
$$s, h \vDash \mathsf{x} \mapsto - \text{ iff } \operatorname{dom} h = \{s\,\mathsf{x}\}$$

$$s, h \vDash \mathsf{x} \hookrightarrow \mathsf{y} \text{ iff } s\,\mathsf{x} \in \operatorname{dom} h \text{ and } h(s\,\mathsf{x}) = s\,\mathsf{y}$$
$$s, h \vDash \mathsf{x} \hookrightarrow - \text{ iff } s\,\mathsf{x} \in \operatorname{dom} h$$

$$s, h \vDash \mathsf{x} \mapsto \mathsf{y}, \mathsf{z} \text{ iff } h = \{\langle s\,\mathsf{x}, s\,\mathsf{y}\rangle, \langle s\,\mathsf{x} + 1, s\,\mathsf{z}\rangle\}$$
$$s, h \vDash \mathsf{x} \mapsto -, - \text{ iff } \operatorname{dom} h = \{s\,\mathsf{x}, s\,\mathsf{x} + 1\}$$

$$s, h \vDash \mathsf{x} \hookrightarrow \mathsf{y}, \mathsf{z} \text{ iff } h \supseteq \{\langle s\,\mathsf{x}, s\,\mathsf{y}\rangle, \langle s\,\mathsf{x} + 1, s\,\mathsf{z}\rangle\}$$
$$s, h \vDash \mathsf{x} \hookrightarrow -.- \text{ iff } \operatorname{dom} h \supseteq \{s\,\mathsf{x}, s\,\mathsf{x} + 1\}.$$

Examples of $*$

Suppose $s\,\mathsf{x}$ and $s\,\mathsf{y}$ are distinct addresses, so that

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

are heaps with disjoint domains. Then

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

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

$$p_1 * p_2 \Leftrightarrow p_2 * p_1$$
$$(p_1 * p_2) * p_3 \Leftrightarrow p_1 * (p_2 * p_3)$$
$$p * \mathbf{emp} \Leftrightarrow p$$
$$(p_1 \vee p_2) * q \Leftrightarrow (p_1 * q) \vee (p_2 * q)$$
$$(p_1 \wedge p_2) * q \Rightarrow (p_1 * q) \wedge (p_2 * q)$$
$$(\exists x.\ p_1) * p_2 \Leftrightarrow \exists x.\ (p_1 * p_2) \quad \text{when } x \text{ not free in } p_2$$
$$(\forall x.\ p_1) * p_2 \Rightarrow \forall x.\ (p_1 * p_2) \quad \text{when } x \text{ not free in } p_2$$

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

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

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

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

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

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

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

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

Two Unsound Axiom Schemata

$$p \not\Rightarrow p \ast p \qquad \text{(Contraction)}$$

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

$$p \ast q \not\Rightarrow p \qquad \text{(Weakening)}$$

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

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

Some Derived Inference Rules

$$\frac{}{q \ast (q \mathrel{-\!\!\ast} p) \Rightarrow p}$$

1. $q \ast (q \mathrel{-\!\!\ast} p) \Rightarrow (q \mathrel{-\!\!\ast} p) \ast q$ $\qquad (p_1 \ast p_2 \Rightarrow p_2 \ast p_1)$
2. $(q \mathrel{-\!\!\ast} p) \Rightarrow (q \mathrel{-\!\!\ast} p)$ $\qquad (p \Rightarrow p)$
3. $(q \mathrel{-\!\!\ast} p) \ast q \Rightarrow p$ $\qquad$ (decurrying, 2)
4. $q \ast (q \mathrel{-\!\!\ast} p) \Rightarrow p$ $\qquad$ (trans impl, 1, 3)

where *transitive implication* is the inference rule

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

$$\frac{}{r \Rightarrow (q \mathrel{-\!\!\ast} (q \ast r))}$$

1. $(r \ast q) \Rightarrow (q \ast r)$ $\qquad (p_1 \ast p_2 \Rightarrow p_2 \ast p_1)$
2. $r \Rightarrow (q \mathrel{-\!\!\ast} (q \ast r))$ $\qquad$ (currying, 1)

More Derived Inference Rules

$$\frac{}{(p \ast r) \Rightarrow (p \ast (q \mathrel{-\!\!\ast} (q \ast r)))}$$

1.	$p \Rightarrow p$	$(p \Rightarrow p)$
2.	$r \Rightarrow (q \mathrel{-\!\!\ast} (q \ast r))$	(derived above)
3.	$(p \ast r) \Rightarrow (p \ast (q \mathrel{-\!\!\ast} (q \ast r)))$	(monotonicity, 1, 2)

$$\frac{p_1 \Rightarrow (q \mathrel{-\!\!\ast} r) \qquad p_2 \Rightarrow (r \mathrel{-\!\!\ast} s)}{p_2 \ast p_1 \Rightarrow (q \mathrel{-\!\!\ast} s)}$$

1.	$p_2 \Rightarrow p_2$	$(p \Rightarrow p)$
2.	$p_1 \Rightarrow (q \mathrel{-\!\!\ast} r)$	(assumption)
3.	$p_1 \ast q \Rightarrow r$	(decurrying, 2)
4.	$p_2 \ast p_1 \ast q \Rightarrow p_2 \ast r$	(monotonicity, 1, 3)
5.	$p_2 \Rightarrow (r \mathrel{-\!\!\ast} s)$	(assumption)
6.	$p_2 \ast r \Rightarrow s$	(decurrying, 5)
7.	$p_2 \ast p_1 \ast q \Rightarrow s$	(trans impl, 4, 6)
8.	$p_2 \ast p_1 \Rightarrow (q \mathrel{-\!\!\ast} s)$	(currying, 7)

More Derived Inference Rules

$$\frac{p' \Rightarrow p \qquad q \Rightarrow q'}{(p \mathrel{-\!\!*} q) \Rightarrow (p' \mathrel{-\!\!*} q').}$$

1.	$(p \mathrel{-\!\!*} q) \Rightarrow (p \mathrel{-\!\!*} q)$	$(p \Rightarrow p)$
2.	$p' \Rightarrow p$	(assumption)
3.	$(p \mathrel{-\!\!*} q) * p' \Rightarrow (p \mathrel{-\!\!*} q) * p$	(monotonicity, 1, 2)
4.	$(p \mathrel{-\!\!*} q) * p \Rightarrow q$	(decurrying, 1)
5.	$(p \mathrel{-\!\!*} q) * p' \Rightarrow q$	(trans impl, 3, 4)
6.	$q \Rightarrow q'$	(assumption)
7.	$(p \mathrel{-\!\!*} q) * p' \Rightarrow q'$	(trans impl, 5, 6)
8.	$(p \mathrel{-\!\!*} q) \Rightarrow (p' \mathrel{-\!\!*} q')$	(currying, 7)

Free Variables of Assertions

$$\mathrm{FV}(\forall v.\ p) = \mathrm{FV}(p) - \{v\}$$
$$\mathrm{FV}(\exists v.\ p) = \mathrm{FV}(p) - \{v\},$$

and for other forms of assertions,

$$\mathrm{FV}(\cdots p_1 \cdots p_n \cdots) = \mathrm{FV}(p_1) \cup \cdots \cup \mathrm{FV}(p_n).$$

Substitution in Assertions

Let p be an assertion such that $\mathrm{FV}(p) \subseteq \{v_1, \ldots, v_n\}$, and let δ denote a substitution $v_1 \to e_1, \ldots, v_n \to e_n$. When p is a quantified assertion, p/δ is defined by

$$(\forall v.\ p)/\delta = \forall v_{\mathrm{new}}.\ (p/[\delta | v \to v_{\mathrm{new}}])$$
$$(\exists v.\ p)/\delta = \exists v_{\mathrm{new}}.\ (p/[\delta | v \to v_{\mathrm{new}}])$$

where $v_{\mathrm{new}} \notin \mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n)$. For other forms of assertions,

$$(\cdots p_1 \cdots p_n \cdots)/\delta = \cdots (p_1/\delta) \cdots (p_n/\delta) \cdots .$$

The Substitution Law for Assertions

Proposition 8 *Let δ abbreviate the substitution*

$$v_1 \to e_1, \ldots, v_n \to e_n,$$

let s be a store such that $\mathrm{FV}(e_1) \cup \cdots \cup \mathrm{FV}(e_n) \subseteq \mathrm{dom}\, s$, and let

$$\widehat{s} = [\, v_1\colon [\![e_1]\!]_{\mathrm{exp}} s \mid \ldots \mid v_n\colon [\![e_n]\!]_{\mathrm{exp}} s \,].$$

If p is an assertion such that $\mathrm{FV}(p) \subseteq \{v_1, \ldots, v_n\}$, then

$$s, h \vDash (p/\delta) \quad \textit{iff} \quad \widehat{s}, h \vDash p.$$