CLASS NOTES FOR CS 819A4 - SPRING 2003

8. Information Hiding and the Hypothetical Frame Rule

John C. Reynolds
Department of Computer Science
Carnegie Mellon University

Revised May 23, 2003

©2003 John C. Reynolds

Parameterless Proper Procedures (with free variables)

Let $k_1, \ldots, k_n$ be identifiers that stand for commands. We introduce judgements containing hypotheses about these identifiers:

$$\{p_1\}\ k_1\ \{q_1\}, \ldots, \{p_n\}\ k_n\ \{q_n\} \vdash \{p\}\ c\ \{q\}.$$

Then

$$\frac{}{\Gamma, \{p_i\}\ k_i\ \{q_i\}, \Gamma' \vdash \{p_i\}\ k_i\ \{q_i\}},$$

and

$$\frac{\Gamma, \{p_n\}\ k_n\ \{q_n\} \vdash \{p\}\ c\ \{q\} \qquad \{p_n\}\ c_n\ \{q_n\}}{\Gamma \vdash \{p\}\ (c/k_n \to c_n)\ \{q\}},$$

or

$$\frac{\Gamma, \{p_n\}\ k_n\ \{q_n\} \vdash \{p\}\ c\ \{q\} \qquad \{p_n\}\ c_n\ \{q_n\}}{\Gamma \vdash \{p\}\ \mathbf{proc}\ k_n = c_n\ \mathbf{in}\ c\ \{q\}}.$$

The Hypothetical Frame Rule

Subject to certain conditions:

$$\frac{\{p_1\}\ k_1\ \{q_1\}, \ldots, \{p_n\}\ k_n\ \{q_n\} \vdash \{p\}\ c\ \{q\}}{\{p_1 * r\}\ k_1\ \{q_1 * r\}, \ldots, \{p_n * r\}\ k_n\ \{q_n * r\} \vdash \{p * r\}\ c\ \{q * r\}}$$

where c does not assign to any variable free in r.

An Example of the Hypothetical Frame Rule

Let

$$\begin{aligned} \mathit{ALLOC} &\stackrel{\text{def}}{=} \mathbf{if}\ \mathsf{f} = \mathbf{nil}\ \mathbf{then}\ \mathsf{x} := \mathbf{cons}(0, \mathbf{nil})\ \mathbf{else} \\ &\qquad\qquad (\mathsf{x} := \mathsf{f}\,; \mathsf{f} := [\mathsf{x}+1]) \end{aligned}$$

$$\mathit{FREE} \stackrel{\text{def}}{=} ([\mathsf{x}+1] := \mathsf{f}\,; \mathsf{f} := \mathsf{x})$$

$$r \stackrel{\text{def}}{=} \exists\alpha.\ \mathsf{list}\ \alpha\,(\mathsf{f}, \mathbf{nil}),$$

so that

$$\{\mathbf{emp} \ast r\}\ \mathit{ALLOC}\ \{\mathsf{x} \mapsto -, - \ast r\}$$

and

$$\{\mathsf{x} \mapsto -, - \ast r\}\ \mathit{FREE}\ \{\mathbf{emp} \ast r\}.$$

Now suppose

$$\{\mathbf{emp}\}\ \mathsf{alloc}\ \{\mathsf{x} \mapsto -, -\}, \{\mathsf{x} \mapsto -, -\}\ \mathsf{free}\ \{\mathbf{emp}\} \vdash \{p\}\ c\ \{q\}.$$

Then the hypothetical frame rule gives

$$\begin{aligned} \{\mathbf{emp} \ast r\}\ \mathsf{alloc}\ \{\mathsf{x} \mapsto -, - \ast r\}, \{\mathsf{x} \mapsto -, - \ast r\}\ \mathsf{free}\ \{\mathbf{emp} \ast r\} \\ \vdash \{p \ast r\}\ c\ \{q \ast r\}, \end{aligned}$$

so that

$$\{p \ast r\}\ (c/\mathsf{alloc} \to \mathit{ALLOC}, \mathsf{free} \to \mathit{FREE})\ \{q \ast r\},$$

or

$$\{p \ast r\}\ \mathbf{proc}\ \mathsf{alloc} = \mathit{ALLOC}\ \mathbf{in}\ \mathbf{proc}\ \mathsf{free} = \mathit{FREE}\ \mathbf{in}\ c\ \{q \ast r\}.$$

A Counterexample

Let

$$\mathbf{one} \stackrel{\text{def}}{=} \exists \mathsf{x}.\ \mathsf{x} \mapsto -.$$

Then

$$\{\mathbf{emp} \vee \mathbf{one}\}\ \mathsf{k}\ \{\mathbf{emp}\} \vdash \{\mathbf{emp} \vee \mathbf{one}\}\ \mathsf{k}\ \{\mathbf{emp}\}$$

$$\{\mathbf{emp} \vee \mathbf{one}\}\ \mathsf{k}\ \{\mathbf{emp}\} \vdash \{\mathbf{emp}\}\ \mathsf{k}\ \{\mathbf{emp}\}$$

$$\{\mathbf{emp} \vee \mathbf{one}\}\ \mathsf{k}\ \{\mathbf{emp}\} \vdash \{\mathbf{emp} * \mathbf{one}\}\ \mathsf{k}\ \{\mathbf{emp} * \mathbf{one}\} \tag{1}$$

$$\{\mathbf{emp} \vee \mathbf{one}\}\ \mathsf{k}\ \{\mathbf{emp}\} \vdash \{\mathbf{one}\}\ \mathsf{k}\ \{\mathbf{one}\}$$

$$\{\mathbf{emp} \vee \mathbf{one}\}\ \mathsf{k}\ \{\mathbf{emp}\} \vdash \{\mathbf{one}\}\ \mathsf{k}\ \{\mathbf{emp}\}$$

$$\{\mathbf{emp} \vee \mathbf{one}\}\ \mathsf{k}\ \{\mathbf{emp}\} \vdash \{\mathbf{one}\}\ \mathsf{k}\ \{\mathbf{emp} \wedge \mathbf{one}\} \tag{2}$$

$$\{\mathbf{emp} \vee \mathbf{one}\}\ \mathsf{k}\ \{\mathbf{emp}\} \vdash \{\mathbf{one}\}\ \mathsf{k}\ \{\mathbf{false}\}$$

$$\begin{aligned}\{(\mathbf{emp} \vee \mathbf{one}) * \mathbf{true}\}\ \mathsf{k}\ \{\mathbf{emp} * \mathbf{true}\} \\ \vdash \{\mathbf{one} * \mathbf{true}\}\ \mathsf{k}\ \{\mathbf{false} * \mathbf{true}\}\end{aligned} \tag{3}$$

$$\{\mathbf{true}\}\ \mathsf{k}\ \{\mathbf{true}\} \vdash \{\mathbf{one} * \mathbf{true}\}\ \mathsf{k}\ \{\mathbf{false}\}$$

$$\{\mathbf{true}\}\ \mathbf{skip}\ \{\mathbf{true}\}$$

$$\{\mathbf{one} * \mathbf{true}\}\ \mathbf{skip}\ \{\mathbf{false}\}$$

1. by frame rule

2. by conjunction rule

3. by hypothetical frame rule

A Solution

Let $k_1, \ldots, k_n$ be identifiers that stand for commands.

- If either r or each of $p_1, \ldots, p_n$ are precise assertions,
- or if all of the assertions are intuitionistic and r is a supported assertion,

Then

$$\frac{\{p_1\}\ k_1\ \{q_1\}, \ldots, \{p_n\}\ k_n\ \{q_n\} \vdash \{p\}\ c\ \{q\}}{\{p_1 * r\}\ k_1\ \{q_1 * r\}, \ldots, \{p_n * r\}\ k_n\ \{q_n * r\} \vdash \{p * r\}\ c\ \{q * r\}}$$

where c does not assign to any variable free in r.

Adding Command Variables (Parameterless Procedures)

$$\langle\text{comm}\rangle ::= \langle\text{cvar}\rangle \mid \textbf{proc}\ \langle\text{cvar}\rangle = \langle\text{comm}\rangle\ \textbf{in}\ \langle\text{comm}\rangle \mid\ \cdots$$

$$\begin{aligned}
\mathrm{FV}(k) &= \{\} \\
\mathrm{FV}(\textbf{proc}\ k = c_p\ \textbf{in}\ c) &= \mathrm{FV}(c_p) \cup \mathrm{FV}(c)
\end{aligned}$$

$$\begin{aligned}
\mathrm{FC}(k) &= \{k\} \\
\mathrm{FC}(\textbf{proc}\ k = c_p\ \textbf{in}\ c) &= \mathrm{FC}(c_p) \cup (\mathrm{FC}(c) - \{k\}) \\
\mathrm{FC}(\cdots c_1 \cdots c_n \cdots) &= \mathrm{FC}(c_1) \cup \cdots \cup \mathrm{FC}(c_n) \\
&\qquad \text{for other constructions of commands.}
\end{aligned}$$

We define

$$\mathrm{Comm}_{KV} \stackrel{\text{def}}{=} \{\, c \mid c \in \langle\text{comm}\rangle \text{ and } \mathrm{FC}(c) \subseteq K \text{ and } \mathrm{FV}(c) \subseteq V \,\}.$$

Structural Operational Semantics Revisited

$$
\begin{aligned}
\mathrm{Stores}_V &= V \to \mathrm{Values}\\
\mathrm{Heaps} &= \bigcup\nolimits_{A \overset{\mathrm{fin}}{\subseteq} \mathrm{Addresses}} (A \to \mathrm{Values})\\
\mathrm{States}_V &= \mathrm{Stores}_V \times \mathrm{Heaps}\\
\mathrm{Nonterminals}_{KV} &= \mathrm{Comm}_{KV} \times \mathrm{States}_V\\
\mathrm{Terminals}_V &= \mathrm{States}_V \cup \{\mathbf{abort}, \bot\}\\
\mathrm{CMeanings}_V &\subseteq \mathcal{P}(\mathrm{States}_V \times \mathrm{Terminals}_V)\\
\mathrm{Environments}_{KV} &= K \to \mathrm{CMeanings}_V.
\end{aligned}
$$

When $\eta \in \mathrm{Environments}_{KV}$,

$$\overset{\eta V}{\rightsquigarrow} \in \mathcal{P}(\mathrm{Nonterminals}_{KV} \times (\mathrm{Nonterminals}_{KV} \cup \mathrm{Terminals}_V))$$

is defined by the inference rules in Section 2 (with appropriate decorations), plus the rules

- Call

  When $(s, h), \tau \in \eta k$:

$$\frac{}{k, (s, h) \overset{\eta V}{\rightsquigarrow} \tau}$$

and

Structural Operational Semantics (continued)

- Procedure Declaration

  When $\eta \in \text{Environment}_{KV}$ and $\eta' = [\,\eta \mid k\colon [\![c_p]\!]^{KV}_{\text{comm}}\,]\eta$:

$$\frac{c, (s,h) \overset{\eta' V}{\rightsquigarrow} c', (s',h')}{\textbf{proc}\ k = c_p\ \textbf{in}\ c, (s,h) \overset{\eta V}{\rightsquigarrow} \textbf{proc}\ k = c_p\ \textbf{in}\ c', (s',h')}$$

$$\frac{c, (s,h) \overset{\eta' V}{\rightsquigarrow} \tau}{\textbf{proc}\ k = c_p\ \textbf{in}\ c, (s,h) \overset{\eta V}{\rightsquigarrow} \tau}$$

Here

$$[\![c]\!]^{KV}_{\text{comm}} \in \text{Environments}_{KV} \to \text{CMeanings}_V.$$

is the large-step semantic function such that $[\![c]\!]^{KV}_{\text{comm}}\eta$ is the total relation satisfying

$(s,h), (s',h') \in [\![c]\!]^{KV}_{\text{comm}}\eta$ iff there is a finite execution sequence

$$c, (s,h) \overset{\eta V}{\rightsquigarrow} \cdots \overset{\eta V}{\rightsquigarrow} (s',h'),$$

$(s,h), \textbf{abort} \in [\![c]\!]^{KV}_{\text{comm}}\eta$ iff there is a finite execution sequence

$$c, (s,h) \overset{\eta V}{\rightsquigarrow} \cdots \overset{\eta V}{\rightsquigarrow} \textbf{abort},$$

$(s,h), \perp \in [\![c]\!]^{KV}_{\text{comm}}\eta$ iff there is a finite execution sequence

$$c, (s,h) \overset{\eta V}{\rightsquigarrow} \cdots \overset{\eta V}{\rightsquigarrow} \perp,$$

or there is an infinite execution sequence

$$c, (s,h) \overset{\eta V}{\rightsquigarrow} \cdots .$$

Structural Operational Semantics (continued)

We must also change the rules for initialized variable declarations. We assume that $v \notin V$, which can always be achieved by renaming. Let $V' = V \cup \{v\}$ and η' be such that, for all $k \in \operatorname{dom} \eta$:

$$
\begin{aligned}
(s, h), (s', h') \in \eta' k \ \text{ iff } \ & (s \rceil V, h), (s' \rceil V, h') \in \eta k \text{ and } sv = s'v \\
(s, h), \textbf{abort} \in \eta' k \ \text{ iff } \ & (s \rceil V, h), \textbf{abort} \in \eta k \\
(s, h), \bot \in \eta' k \ \text{ iff } \ & (s \rceil V, h), \bot \in \eta k.
\end{aligned}
$$

Then

$$\frac{c, ([\, s \mid v{:}\, [\![e]\!]_{\exp} s \,], h) \overset{\eta' V'}{\rightsquigarrow} (s', h')}{\textbf{newvar } v := e \textbf{ in } c, (s, h) \overset{\eta V}{\rightsquigarrow} (s' \rceil V, h'),}$$

$$\frac{c, ([\, s \mid v{:}\, [\![e]\!]_{\exp} s \,], h) \overset{\eta' V'}{\rightsquigarrow} c', (s', h')}{\begin{array}{l}\textbf{newvar } v := e \textbf{ in } c, (s, h) \overset{\eta V}{\rightsquigarrow} \\ \qquad\qquad \textbf{newvar } v := s'v \textbf{ in } c', (s' \rceil V, h'),\end{array}}$$

$$\frac{c, ([\, s \mid v{:}\, [\![e]\!]_{\exp} s \,], h) \overset{\eta' V'}{\rightsquigarrow} \textbf{abort}}{\textbf{newvar } v := e \textbf{ in } c, (s, h) \overset{\eta V}{\rightsquigarrow} \textbf{abort},}$$

$$\frac{c, ([\, s \mid v{:}\, [\![e]\!]_{\exp} s \,], h) \overset{\eta' V'}{\rightsquigarrow} \bot}{\textbf{newvar } v := e \textbf{ in } c, (s, h) \overset{\eta V}{\rightsquigarrow} \bot.}$$

The Copy Rule

When $c \in \mathrm{Comm}_{KV}$ and $\eta \in \mathrm{Environments}_{KV}$,

$$[\![\mathbf{proc}\ k = c_p\ \mathbf{in}\ c]\!]^{KV}_{\mathrm{comm}} = [\![c/k \to c_p]\!]^{KV}_{\mathrm{comm}}.$$

Specifications Revisited

Suppose that $V \supseteq \mathrm{FV}(p) \cup \mathrm{FV}(c) \cup \mathrm{FV}(q)$, $K \supseteq \mathrm{FC}(c)$, and $\eta \in \mathrm{Environments}_{KV}$. Then:

Partial correctness:

$[\![\{p\}\ c\ \{q\}]\!]^{KV}_{\mathrm{spec}}\eta$ iff $\forall (s,h) \in \mathrm{States}_V.\ s,h \vDash p$ implies

$(s,h), \mathbf{abort} \notin [\![c]\!]^{KV}_{\mathrm{comm}}\eta$

and $\big(\forall (s',h') \in \mathrm{States}_V.\ (s,h),(s',h') \in [\![c]\!]^{KV}_{\mathrm{comm}}\eta$ implies $s',h' \vDash q\big)$.

Total correctness:

$[\![[\ p\ ]\ c\ [\ q\ ]]\!]^{KV}_{\mathrm{spec}}\eta$ iff $\forall (s,h) \in \mathrm{States}_V.\ s,h \vDash p$ implies

$(s,h), \mathbf{abort} \notin [\![c]\!]^{KV}_{\mathrm{comm}}\eta$ and $(s,h), \perp \notin [\![c]\!]^{KV}_{\mathrm{comm}}\eta$

and $\big(\forall (s',h') \in \mathrm{States}_V.\ (s,h),(s',h') \in [\![c]\!]^{KV}_{\mathrm{comm}}\eta$ implies $s',h' \vDash q\big)$.

Hypothetical Specifications

Suppose that $V \supseteq \mathrm{FV}(\mathcal{S}_1) \cup \cdots \cup \mathrm{FV}(\mathcal{S}_n) \cup \mathrm{FV}(\mathcal{S})$ and $K \supseteq \mathrm{FC}(\mathcal{S}_1) \cup \cdots \cup \mathrm{FC}(\mathcal{S}_n) \cup \mathrm{FC}(\mathcal{S})$. Then:

$$[\![\mathcal{S}_1, \ldots, \mathcal{S}_n \vdash \mathcal{S}]\!]^{KV}_{\mathrm{hspec}} \text{ iff } \forall \eta \in \mathrm{Environments}_{KV}.$$
$$[\![\mathcal{S}_1]\!]^{KV}_{\mathrm{spec}}\eta \text{ and } \cdots \text{ and } [\![\mathcal{S}_n]\!]^{KV}_{\mathrm{spec}}\eta \text{ implies } [\![\mathcal{S}]\!]^{KV}_{\mathrm{spec}}\eta.$$

(This is independent of the particular choice of V and K.)

The Heart of the Matter

We must still define the set

$$\mathrm{CMeanings}_V \subseteq \mathcal{P}(\mathrm{States}_V \times \mathrm{Terminals}_V).$$

Because of the second half of Proposition 6 (in Section 2), we may impose the following requirement on command meanings:

Suppose $\rho \subseteq \mathrm{States}_V \times \mathrm{Terminals}_V$. Then $\rho \in \mathrm{CMeanings}$ iff, whenever $h = h_0 \cdot h_1$ where $h_0 \perp h_1$,

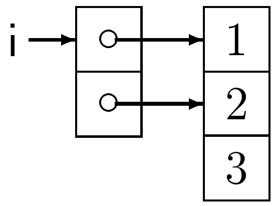
- If $(s, h), \mathbf{abort} \in \rho$, then $(s, h_0), \mathbf{abort} \in \rho$.
- If $(s, h), (s'h') \in \rho$, then $(s, h_0), \mathbf{abort} \in \rho$ or $(s, h_0), (s', h_0') \in \rho$, where $h_0' \perp h_1$ and $h' = h_0' \cdot h_1$.
- If $(s, h), \perp \in \rho$, then $(s, h_0), \mathbf{abort} \in \rho$ or $(s, h_0), \perp \in \rho$.

Future Directions: Prohibiting Skewed Sharing

Our definition of dag permits *skewed sharing.* For example,

$$\mathsf{dag}\,((1 \cdot 2) \cdot (2 \cdot 3))\,(\mathsf{i})$$

holds when



Similar problems would occur for the Schorr-Waite proof in the logic with address arithmetic.

A Possible Solution:

- We add to the state a mapping ϕ from the domain of the heap to natural numbers, called the field count.

- When $x := \mathbf{cons}(e_1, \ldots, e_n)$ sets x to the address a. the field count is extended so that

$$\phi(a) = n \qquad \phi(a+1) = 0 \qquad \cdots \qquad \phi(a+n-1) = 0.$$

- We write $e \overset{h}{\mapsto} e_1, \ldots, e_n$ iff

$$e \mapsto e_1, \ldots, e_n \text{ and } \phi(e) = n$$
$$\text{and } \phi(e+1) = 0 \text{ and } \ldots \text{ and } \phi(e+n-1) = 0.$$

- Then

$$e \overset{h}{\hookrightarrow} e_1, \ldots, e_m \wedge e' \overset{h}{\hookrightarrow} e'_1, \ldots, e'_n \wedge e \neq e' \Rightarrow$$
$$e \overset{h}{\mapsto} e_1, \ldots, e_m \;*\; e' \overset{h}{\mapsto} e'_1, \ldots, e'_n \;*\; \mathbf{true}.$$

Future Directions: Adding Higher-Order Procedures

- Separation Logic

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

where

$$\begin{aligned} \text{Stores} &= \text{Variables} \to \text{Values} \\ \text{Heaps} &= \text{Addresses} \to \text{Values} \\ \text{Addresses} &\subseteq \text{Values} \end{aligned}$$

- Algol-like Languages

$$\text{States} = \text{Environments} \times \text{Stores}$$

where

$$\begin{aligned} \text{Environments} &= \text{Identifiers} \to \text{Meanings} \\ \text{Variables} &\subseteq \text{Meanings} \end{aligned}$$

- Algol with Heaps

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

  - Two levels of aliasing, needs specification logic or syntactic control of interference.
  - No meanings (i.e. closures) in the heap.

An Alternative

- ML (or C)

$$\text{States} = \text{Environments} \times \text{Heaps}$$

where

$$\begin{aligned}\text{Environments} &= \text{Identifiers} \to \text{Values}\\ \text{Heaps} &= \text{Addresses} \to \text{Values}\\ \text{Addresses} &\subseteq \text{Values}\\ \text{Meanings} &\subseteq \text{Values}\end{aligned}$$

The Problem

It is reasonable to replace

$$x := e\,;c \qquad \text{with} \qquad \mathbf{let}\ x = e\ \mathbf{in}\ c.$$

But what about

$$(\mathbf{if}\ p\ \mathbf{then}\ x := e\ \mathbf{else}\ y := e')\,;c\ ?$$

Flat Commands versus Continuation Passing Style

$$
\begin{aligned}
\langle \mathrm{fc} \rangle ::= {} & \langle \mathrm{var} \rangle := \langle \mathrm{exp} \rangle \; ; \langle \mathrm{fc} \rangle \\
& \mid \langle \mathrm{var} \rangle := \mathbf{cons}(\langle \mathrm{exp} \rangle, \ldots, \langle \mathrm{exp} \rangle) \; ; \langle \mathrm{fc} \rangle \\
& \mid \langle \mathrm{var} \rangle := [\langle \mathrm{exp} \rangle] \; ; \langle \mathrm{fc} \rangle \\
& \mid [\langle \mathrm{exp} \rangle] := \langle \mathrm{exp} \rangle \; ; \langle \mathrm{fc} \rangle \\
& \mid \mathbf{dispose} \; \langle \mathrm{exp} \rangle \; ; \langle \mathrm{fc} \rangle \\
& \mid \mathbf{if} \; \langle \mathrm{boolexp} \rangle \; \mathbf{then} \; \langle \mathrm{fc} \rangle \; \mathbf{else} \; \langle \mathrm{fc} \rangle \\
& \mid \mathbf{goto} \; \langle \mathrm{var} \rangle(\langle \mathrm{exp} \rangle, \ldots, \langle \mathrm{exp} \rangle)
\end{aligned}
$$

(where dynamic binding is used).

$$
\begin{aligned}
\langle \mathrm{cpst} \rangle ::= {} & \mathbf{let} \; \langle \mathrm{var} \rangle = \langle \mathrm{exp} \rangle \; \mathbf{in} \; \langle \mathrm{cpst} \rangle \\
& \mid \mathbf{let} \; \langle \mathrm{var} \rangle = \mathbf{cons}(\langle \mathrm{exp} \rangle, \ldots, \langle \mathrm{exp} \rangle) \; \mathbf{in} \; \langle \mathrm{cpst} \rangle \\
& \mid \mathbf{let} \; \langle \mathrm{var} \rangle = [\langle \mathrm{exp} \rangle] \; \mathbf{in} \; \langle \mathrm{cpst} \rangle \\
& \mid [\langle \mathrm{exp} \rangle] := \langle \mathrm{exp} \rangle \; ; \langle \mathrm{cpst} \rangle \\
& \mid \mathbf{dispose} \; \langle \mathrm{exp} \rangle \; ; \langle \mathrm{cpst} \rangle \\
& \mid \mathbf{if} \; \langle \mathrm{boolexp} \rangle \; \mathbf{then} \; \langle \mathrm{cpst} \rangle \; \mathbf{else} \; \langle \mathrm{cpst} \rangle \\
& \mid \langle \mathrm{var} \rangle(\langle \mathrm{exp} \rangle, \ldots, \langle \mathrm{exp} \rangle)
\end{aligned}
$$

(where static binding is used).

We choose to use continuation passing style, possibly with the prohibition of free variables in lambda expressions.

Continuation Passing Style - An Example

Consider a function for appending lists (a la LISP):

$$\begin{aligned}&\textbf{letrec}\ \mathsf{append}(\mathsf{x},\mathsf{y}) =\\&\quad \textbf{if}\ \mathsf{x} = \textbf{nil then}\ \mathsf{y}\ \textbf{else cons}([\mathsf{x}], \mathsf{append}([\mathsf{x}+1], \mathsf{y})).\end{aligned}$$

In continuation passing style, this becomes

$$\begin{aligned}&\textbf{letrec}\ \mathsf{append}(\mathsf{x},\mathsf{y},\mathsf{r}) = \textbf{if}\ \mathsf{x} = \textbf{nil then}\ \mathsf{r}(\mathsf{y})\ \textbf{else}\\&\quad \textbf{let}\ \mathsf{a} = [\mathsf{x}], \mathsf{b} = [\mathsf{x}+1],\\&\qquad \mathsf{k}(\mathsf{z}) = \textbf{let}\ \mathsf{w} = \textbf{cons}(\mathsf{a},\mathsf{z})\ \textbf{in}\ \mathsf{r}(\mathsf{w})\\&\quad \textbf{in append}(\mathsf{b},\mathsf{y},\mathsf{k}).\end{aligned}$$

A Reflection Operator

If t is a CPS term, then $\# \, t$ is an assertion that holds for a state if t never aborts when executed in that state — in this situation we say that it is *safe* to execute t in that state.

Suppose

- c is a command satisfying $\{p\}\ c\ \{q\}$.
- t is a CPS term such that executing t has the same effect as executing c and then calling the continuation $r(x_1, ...x_n)$.

Then we can specify t by the assertion

$$(p \ * \ (\forall x_1, \ldots, x_m.\ q \mathrel{-\!\!*} \# \, r(x_1, \ldots, x_n))) \Rightarrow \# \, t.$$

- In the absence of a heap, this asserts that it is safe to execute t in any state such that
  - p is true, and
  - any change of the variables $x_1, \ldots, x_m$ that makes q true will make it safe to execute $r(x_1, ..., x_n)$.
- In general, this is asserts that it is safe to execute t in any state where the heap can be split into disjoint components such that:
  - p is true in one component, and
  - for any change of the variables $x_1, \ldots, x_m$, if the other component is extended with a heap in which q is true, then it is safe to execute $r(x_1, ..., x_n)$ in the extended heap.

An Example

$$
\begin{array}{l}
\textbf{letrec}\ \mathsf{append}(\mathsf{x},\mathsf{y},\mathsf{r}) = \textbf{if}\ \mathsf{x} = \textbf{nil}\ \textbf{then}\ \mathsf{r}(\mathsf{y})\ \textbf{else} \\
\quad \textbf{let}\ \mathsf{a} = [\mathsf{x}], \mathsf{b} = [\mathsf{x}+1], \\
\qquad \mathsf{k}(\mathsf{z}) = \textbf{let}\ \mathsf{w} = \textbf{cons}(\mathsf{a},\mathsf{z})\ \textbf{in}\ \mathsf{r}(\mathsf{w}) \\
\quad \textbf{in}\ \textbf{append}(\mathsf{b},\mathsf{y},\mathsf{k})
\end{array}
$$

is specified by

$$
\begin{array}{l}
\Big(\mathsf{list}\,\alpha\,(\mathsf{x},\textbf{nil}) \\
\qquad * \Big(\forall \mathsf{z}.\ \ (\mathsf{list}\,\alpha\,(\mathsf{x},\textbf{nil}) * \mathsf{list}\,\alpha\,(\mathsf{z},\mathsf{y})) \mathrel{-\!\!*} \#\,\mathsf{r}(\mathsf{z})\Big)\Big) \\
\quad \Rightarrow \#\,\mathsf{append}(\mathsf{x},\mathsf{y},\mathsf{r}).
\end{array}
$$

Eliminating Global Variables

$$\begin{array}{l}
\textbf{letrec}\ \mathsf{append}(\mathsf{x},\mathsf{y},\mathsf{r}) = \textbf{if}\ \mathsf{x} = \textbf{nil then}\ \mathsf{r}(\mathsf{y})\ \textbf{else} \\
\quad \textbf{let}\ \mathsf{a} = [\mathsf{x}], \mathsf{b} = [\mathsf{x}+1], \\
\qquad \mathsf{k}(\mathsf{z}) = \textbf{let}\ \mathsf{w} = \textbf{cons}(\mathsf{a},\mathsf{z})\ \textbf{in}\ \mathsf{r}(\mathsf{w}) \\
\quad \textbf{in append}(\mathsf{b},\mathsf{y},\mathsf{k})
\end{array}$$

can be transformed into

$$\begin{array}{l}
\textbf{letrec}\ \mathsf{append}(\mathsf{x},\mathsf{y},\mathsf{r},\mathsf{rr}) = \textbf{if}\ \mathsf{x} = \textbf{nil then}\ \mathsf{r}(\mathsf{rr},\mathsf{y})\ \textbf{else} \\
\quad \textbf{let}\ \mathsf{a} = [\mathsf{x}], \mathsf{b} = [\mathsf{x}+1], \mathsf{kk} = \textbf{cons}(\mathsf{r},\mathsf{a},\mathsf{rr}), \\
\qquad \mathsf{k}(\mathsf{kk},\mathsf{z}) = \\
\qquad\quad (\textbf{let}\ \mathsf{r} = [\mathsf{kk}], \mathsf{a} = [\mathsf{kk}+1], \mathsf{rr} = [\mathsf{kk}+2] \\
\qquad\quad \textbf{in}\ (\textbf{dispose}\ \mathsf{kk}, \mathsf{kk}+1, \mathsf{kk}+2\ ; \\
\qquad\qquad \textbf{let}\ \mathsf{w} = \textbf{cons}(\mathsf{a},\mathsf{z})\ \textbf{in}\ \mathsf{r}(\mathsf{rr},\mathsf{w})) \\
\quad \textbf{in append}(\mathsf{b},\mathsf{y},\mathsf{k},\mathsf{kk}),
\end{array}$$

which is specified by

$$\begin{array}{l}
\Big((\mathsf{list}\ \alpha\ (\mathsf{x},\textbf{nil}) * \mathsf{list}\ \beta\ (\mathsf{y},\textbf{nil})) \\
\qquad * \ \Big(\forall \mathsf{z}.\ \ (\mathsf{list}\ \alpha\ (\mathsf{x},\textbf{nil}) * \mathsf{list}\ \alpha\ (\mathsf{z},\mathsf{y}) * \mathsf{list}\ \beta\ (\mathsf{y},\textbf{nil})) \\
\qquad\quad -\!\!* \ \#\ \mathsf{r}(\mathsf{rr},\mathsf{z})\Big)\Big) \\
\quad \Rightarrow \#\ \mathsf{append}(\mathsf{x},\mathsf{y},\mathsf{r},\mathsf{rr}).
\end{array}$$