Adding Effects: The fail Command

Syntax:

$$comm ::= \textbf{fail}$$

Semantics:

Must terminate program execution immediately, reporting the last state encountered.

⇒ failure is similar to nontermination:
if any executed command diverges, the whole program diverges
if any executed command fails, the whole program fails

⇒ semantics of sequencing may use a lifting function similar to $(-)_{\perp\!\!\!\perp}$
but propagating failure instead of nontermination

The Failure Domain

The semantic domain must be extended to account for failure:

$$\begin{aligned} \hat{\Sigma} &\overset{\text{def}}{=} \Sigma \cup (\{\mathbf{abort}\} \times \Sigma) \\ &\simeq \{\mathbf{normal}, \mathbf{abort}\} \times \Sigma \quad \text{(more abstract)} \\ &\simeq \Sigma + \Sigma \end{aligned}$$

The meanings of commands are now of type

$$\begin{aligned} [\![c]\!]_{comm} &\in \Sigma \to (\hat{\Sigma})_{\perp} \\ [\![\mathbf{fail}]\!]_{comm}\sigma &= \langle \mathbf{abort}, \sigma \rangle \end{aligned}$$

Semantic equations for the primitive commands remain “unchanged”:

$$\begin{aligned} [\![v := e]\!]_{comm}\sigma &= [\sigma \mid v : [\![e]\!]_{intexp}\sigma] \\ [\![\mathbf{skip}]\!]_{comm}\sigma &= \sigma \end{aligned}$$

but more abstractly they are modified to

$$\begin{aligned} [\![v := e]\!]_{comm}\sigma &= \langle \mathbf{normal}, [\sigma \mid v : [\![e]\!]_{intexp}\sigma] \rangle \\ [\![\mathbf{skip}]\!]_{comm}\sigma &= \langle \mathbf{normal}, \sigma \rangle \end{aligned}$$

Sequential Composition with Failure

Semantics of sequential composition uses another lifting:

$$[\![c_0 \; ; \; c_1]\!]_{comm} = ([\![c_1]\!]_{comm})_{*} \cdot [\![c_0]\!]_{comm}$$

where for every $f \in S \to \hat{T}_\perp$ the function $f_* \in \hat{S}_\perp \to \hat{T}_\perp$ is defined by

$$\begin{aligned} f_* \perp &= \perp \\ f_* \langle \mathbf{normal}, x \rangle &= f x \\ f_* \langle \mathbf{abort}, x \rangle &= \langle \mathbf{abort}, x \rangle \end{aligned}$$

The semantics of **while** was defined using that of sequencing, so

$$[\![\mathbf{while}\; b \;\mathbf{do}\; c]\!]_{comm} \overset{\text{def}}{=} \mathbf{Y}_{[\Sigma \to \hat{\Sigma}_\perp]} F$$

where $F f \sigma = \mathit{if}\; [\![b]\!]_{boolexp} \sigma = \mathbf{true}\; \mathit{then}\; f_*([\![c]\!]_{comm} \sigma)\; \mathit{else}\; \langle \mathbf{normal}, \sigma \rangle$

Note: These commands are semantically equivalent (for any command c) in a language without failure, but not in one with:

c ; **while true do skip** **while true do skip**

Local Declarations with Failure: Problem

Recall the semantics of local declarations

$$[\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]_{comm}\sigma = ([-\,|\,v : \sigma v])_{\perp\!\!\perp}\,([\![c]\!]_{comm}[\sigma\,|\,v : [\![e]\!]_{intexp}\sigma])$$

The naïve generalization in the presence of failure

$$[\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]_{comm}\sigma = ([-\,|\,v : \sigma v])_{*}\,([\![c]\!]_{comm}[\sigma\,|\,v : [\![e]\!]_{intexp}\sigma])$$

doesn't quite work: if c fails, the result shows the state when c failed:

$$[\![\mathbf{newvar}\ \mathsf{x} := 1\ \mathbf{in}\ \mathbf{fail}]\!]_{comm}\sigma = \langle \mathbf{abort},\ [\sigma\,|\,\mathsf{x} : 1]\rangle$$

so names of local variables can be exported out of scope

⇒ renaming does not preserve meaning:

$$\begin{aligned}[\![\mathsf{x} := 0\ ;\ \mathbf{newvar}\ \mathsf{x} := 1\ \mathbf{in}\ \mathbf{fail}]\!]_{comm}\sigma &= \langle \mathbf{abort},\ [\sigma\,|\,\mathsf{x} : 1]\rangle \\ [\![\mathsf{x} := 0\ ;\ \mathbf{newvar}\ \mathsf{y} := 1\ \mathbf{in}\ \mathbf{fail}]\!]_{comm}\sigma &= \langle \mathbf{abort},\ [\sigma\,|\,\mathsf{x} : 0\,|\,\mathsf{y} : 1]\rangle\end{aligned}$$

Conclusion: The old bindings of local variables must be restored
even when the result is in $\{\mathbf{abort}\} \times \Sigma$.

Local Declarations with Failure: Solution

Use yet another lifting function to restore bindings: if $f \in S \to T$, then $f_\dagger \in \hat{S}_\perp \to \hat{T}_\perp$

$$\begin{aligned} f_\dagger \perp &= \perp \\ f_\dagger \langle \mathbf{abort},\ x \rangle &= \langle \mathbf{abort},\ f x \rangle \\ f_\dagger \langle \mathbf{normal},\ x \rangle &= \langle \mathbf{normal},\ f x \rangle \end{aligned}$$

Then

$$[\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]_{comm} \sigma = ([- \mid v : \sigma v])_\dagger \, ([\![c]\!]_{comm} [\sigma \mid v : [\![e]\!]_{intexp} \sigma])$$

Effectively failure is “caught” at local declarations
and “re-raised” after the old binding is restored.

Semantics of Failure

$$\begin{aligned}
\hat{\Sigma} &= \{\mathbf{normal},\ \mathbf{abort}\} \times \Sigma \\
[\![c]\!]_{comm} &\in \Sigma \to (\hat{\Sigma})_{\perp}
\end{aligned}$$

$$\begin{aligned}
[\![\mathbf{fail}]\!]_{comm}\sigma &= \langle \mathbf{abort},\ \sigma \rangle \\
[\![v := e]\!]_{comm}\sigma &= \langle \mathbf{normal},\ [\sigma \mid v : [\![e]\!]_{intexp}\sigma] \rangle \\
[\![\mathbf{skip}]\!]_{comm}\sigma &= \langle \mathbf{normal},\ \sigma \rangle \\
[\![c_0\ ;\ c_1]\!]_{comm}\sigma &= ([\![c_1]\!]_{comm})_{*}([\![c_0]\!]_{comm}\sigma) \\
[\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]_{comm}\sigma &= ([-\mid v : \sigma v])_{\dagger}\ ([\![c]\!]_{comm}[\sigma \mid v : [\![e]\!]_{intexp}\sigma])
\end{aligned}$$

$$\begin{aligned}
f_{*}\perp &= \perp \\
f_{*}\langle \mathbf{normal},\ \sigma \rangle &= f\,\sigma \\
f_{*}\langle \mathbf{abort}, \sigma \rangle &= \langle \mathbf{abort},\ \sigma \rangle
\end{aligned}$$

$$\begin{aligned}
f_{\dagger}\perp &= \perp \\
f_{\dagger}\langle \mathbf{normal},\ \sigma \rangle &= \langle \mathbf{normal},\ f\,\sigma \rangle \\
f_{\dagger}\langle \mathbf{abort},\ \sigma \rangle &= \langle \mathbf{abort},\ f\,\sigma \rangle
\end{aligned}$$

(the equations for the conditional and the loop look unchanged)

Specifications with Failure

Recall semantics of total and partial correctness:

$$\begin{array}{rcl} [\![\, [p]\ c\ [q]\,]\!]_{spec} & = & \forall \sigma \in \Sigma.\ [\![p]\!]_{assert}\sigma \Rightarrow \\ & & ([\![c]\!]_{comm}\sigma \neq \bot \text{ and } [\![q]\!]_{assert}([\![c]\!]_{comm}\sigma)) \\ [\![\{p\}\ c\ \{q\}]\!]_{spec} & = & \forall \sigma \in \Sigma.\ [\![p]\!]_{assert}\sigma \Rightarrow \\ & & ([\![c]\!]_{comm}\sigma = \bot \text{ or } [\![q]\!]_{assert}([\![c]\!]_{comm}\sigma)) \end{array}$$

Our assertion language cannot handle results in $\{\mathbf{abort}\} \times \Sigma$,
so we treat these results as failing to satisfy an assertion:

$$\begin{array}{rcl} [\![\, [p]\ c\ [q]\,]\!] & = & \forall \sigma \in \Sigma.\ [\![p]\!]\sigma \Rightarrow ([\![c]\!]\sigma \notin \{\bot\} \cup (\{\mathbf{abort}\} \times \Sigma) \text{ and } [\![q]\!]([\![c]\!]\sigma)) \\ [\![\{p\}\ c\ \{q\}]\!] & = & \forall \sigma \in \Sigma.\ [\![p]\!]\sigma \Rightarrow ([\![c]\!]\sigma \in \{\bot\} \cup (\{\mathbf{abort}\} \times \Sigma) \text{ or } [\![q]\!]([\![c]\!]\sigma)) \end{array}$$

Then the strongest rules for **fail** are

$$\text{(FLT)}\quad [\mathbf{false}]\ \mathbf{fail}\ [\mathbf{false}]$$

$$\text{(FLP)}\quad \{\mathbf{true}\}\ \mathbf{fail}\ \{\mathbf{false}\}$$

More Effects: Intermediate Output

Syntax:

$$comm ::= \texttt{!}\, intexp$$

Intended semantics:

$\texttt{!}e$ outputs the value of e (and then the execution continues).

Major change in program meaning:

- Even two nonterminating programs may have observably different behaviors.
- Part of the result of executing a program is its output, which can be an infinite object.

Semantics of Output: The Domain of Sequences

Example:

$$\texttt{!0}\ ;\ \mathbf{while}\ \texttt{n} \geq \texttt{0}\ \mathbf{do}\ \mathbf{if}\ \texttt{n} \neq \texttt{0}\ \mathbf{then}\ (\texttt{!n}\ ;\ \texttt{n:=n+1})\ \mathbf{else}\ \mathbf{skip}$$

A program can behave in one of three ways:

- Output a finite sequence and then terminate (normally or failing)
- Output a finite sequence and then diverge without further output
- Output an infinite sequence

⇒ the output domain can be defined (up to isomorphism) as

$$\Omega \overset{\text{def}}{=} \bigcup_{n=0}^{\infty} (\mathbf{Z}^n \times \hat{\Sigma}) \quad \cup \quad \bigcup_{n=0}^{\infty} \mathbf{Z}^n \quad \cup \quad \mathbf{Z}^{\mathbf{N}}$$

Partial Order in the Domain of Sequences

The partial order should reflect the idea that

$$\omega \sqsubseteq \omega' \text{ if output } \omega' \text{ is “more defined” than output } \omega.$$

If “more defined” is interpreted with respect to the length of observation, we get

$$\omega \sqsubseteq \omega' \iff \omega \text{ is a prefix of } \omega'$$

Then the empty sequence $\langle\rangle$ is the least element of Ω.

There are three kinds of chains in Ω:

$\langle\rangle \sqsubseteq \langle 7\rangle \sqsubseteq \langle 7,0\rangle \sqsubseteq \langle 7,0\rangle \sqsubseteq \langle 7,0\rangle \sqsubseteq \ldots$ (diverging with finite output)

$\langle\rangle \sqsubseteq \langle 7\rangle \sqsubseteq \langle 7,0\rangle \sqsubseteq \langle 7,0,\hat{\sigma}\rangle \sqsubseteq \langle 7,0,\hat{\sigma}\rangle \sqsubseteq \ldots$ (terminating)

$\langle\rangle \sqsubseteq \langle 7\rangle \sqsubseteq \langle 7,0\rangle \sqsubseteq \langle 7,0,7\rangle \sqsubseteq \langle 7,0,7,1\rangle \sqsubseteq \ldots$

Only chains of the latter kind are interesting, and their limits are in Ω since $\mathbf{Z}^{\mathbf{N}} \subseteq \Omega$:

$$\text{if } \omega_0 \sqsubseteq \omega_1 \sqsubseteq \ldots \text{ is such a chain, then } \bigsqcup_{n=0}^{\infty} \omega_n = \{\, [i,\, \omega_j\, i] \mid j \in \mathbf{N} \text{ and } i \in \operatorname{dom} \omega_j \,\}$$

The Domain of Sequences as an Initial Continuous Algebra

Idea: represent $\Omega = \bigcup_{n=0}^{\infty} (\mathbf{Z}^n \times \hat{\Sigma}) \cup \bigcup_{n=0}^{\infty} \mathbf{Z}^n \cup \mathbf{Z}^{\mathrm{N}}$ using abstract syntax.

The constructors are

$$
\begin{array}{rcl@{\qquad}rcl}
\iota_{\perp} & \in & \{\{\}\} \to \Omega & \iota_{\perp}\,\langle\rangle & = & \langle\rangle \\
\iota_{\mathsf{term}} & \in & \Sigma \to \Omega & \iota_{\mathsf{term}}\,\sigma & = & \langle\sigma\rangle \\
\iota_{\mathsf{abort}} & \in & \Sigma \to \Omega & \iota_{\mathsf{abort}}\,\sigma & = & \langle\langle\mathbf{abort},\ \sigma\rangle\rangle \\
\iota_{\mathsf{out}} & \in & \mathbf{Z} \times \Omega \to \Omega & \iota_{\mathsf{out}}\,\langle n,\ \omega\rangle & = & \langle n\rangle \mathbin{+\!\!+} \omega
\end{array}
$$

($\mathbin{+\!\!+}$ is concatenation of sequences)

Finite applications of constructors define an initial algebra – the finite sequences in Ω.

Completing this set with its limits defines Ω as an initial continuous algebra.

Semantics in the Domain of Sequences

The semantic equations become

$$
\begin{array}{rcl}
[\![-]\!]_{comm} & \in & comm \rightarrow \Sigma \rightarrow \Omega \\[1ex]
[\![\mathbf{skip}]\!]_{comm}\sigma & = & \iota_{\mathsf{term}}\,\sigma \\
[\![v := e]\!]_{comm}\sigma & = & \iota_{\mathsf{term}}\,[\sigma \mid v : [\![e]\!]_{intexp}\sigma] \\
[\![\mathbf{fail}]\!]_{comm}\sigma & = & \iota_{\mathsf{abort}}\,\sigma \\
[\![!\,e]\!]_{comm}\sigma & = & \iota_{\mathsf{out}}\,\langle [\![e]\!]_{intexp}\sigma,\ \iota_{\mathsf{term}}\,\sigma \rangle \\
[\![c_0\ ;\ c_1]\!]_{comm}\sigma & = & ([\![c_1]\!]_{comm})_*([\![c_0]\!]_{comm}\sigma) \\
[\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]_{comm}\sigma & = & ([- \mid v : \sigma v])_\dagger\ ([\![c]\!]_{comm}[\sigma \mid v : [\![e]\!]_{intexp}\sigma])
\end{array}
$$

$$
\begin{array}{rcl}
f_*\bot & = & \bot \\
f_*(\iota_{\mathsf{term}}\,\sigma) & = & f\,\sigma \\
f_*(\iota_{\mathsf{abort}}\,\sigma) & = & \iota_{\mathsf{abort}}\,\sigma \\
f_*(\iota_{\mathsf{out}}\,\langle n,\ \omega\rangle) & = & \iota_{\mathsf{out}}\,\langle n,\ f_*\,\omega\rangle
\end{array}
\qquad
\begin{array}{rcl}
f_\dagger\bot & = & \bot \\
f_\dagger(\iota_{\mathsf{term}}\,\sigma) & = & \iota_{\mathsf{term}}\,(f\,\sigma) \\
f_\dagger(\iota_{\mathsf{abort}}\,\sigma) & = & \iota_{\mathsf{abort}}\,(f\,\sigma) \\
f_\dagger(\iota_{\mathsf{out}}\,\langle n,\ \omega\rangle) & = & \iota_{\mathsf{out}}\,\langle n,\ f_\dagger\,\omega\rangle
\end{array}
$$

(the equations for the conditional and the loop still look unchanged)

Semantics of Output: An Example

$$
\begin{aligned}
& [\![\texttt{!3 ; !6 ;}\ \mathbf{fail}]\!]\, \sigma \\
= \quad & [\![\mathbf{fail}]\!]_* \, ([\![\texttt{!6}]\!]_* \, ([\![\texttt{!3}]\!]\, \sigma)) \\
= \quad & [\![\mathbf{fail}]\!]_* \, ([\![\texttt{!6}]\!]_* \, (\iota_{\mathsf{out}} \langle 3,\ \iota_{\mathsf{term}}\, \sigma \rangle)) \\
= \quad & [\![\mathbf{fail}]\!]_* \, (\iota_{\mathsf{out}} \langle 3,\ [\![\texttt{!6}]\!]_* \, (\iota_{\mathsf{term}}\, \sigma) \rangle) \\
= \quad & [\![\mathbf{fail}]\!]_* \, (\iota_{\mathsf{out}} \langle 3,\ [\![\texttt{!6}]\!]\, \sigma \rangle) \\
= \quad & [\![\mathbf{fail}]\!]_* \, (\iota_{\mathsf{out}} \langle 3,\ \iota_{\mathsf{out}} \langle 6,\ \iota_{\mathsf{term}}\, \sigma \rangle \rangle) \\
= \quad & \iota_{\mathsf{out}} \langle 3,\ [\![\mathbf{fail}]\!]_* \, (\iota_{\mathsf{out}} \langle 6,\ \iota_{\mathsf{term}}\, \sigma \rangle) \rangle \\
= \quad & \iota_{\mathsf{out}} \langle 3,\ \iota_{\mathsf{out}} \langle 6,\ [\![\mathbf{fail}]\!]_* \, (\iota_{\mathsf{term}}\, \sigma) \rangle \rangle \\
= \quad & \iota_{\mathsf{out}} \langle 3,\ \iota_{\mathsf{out}} \langle 6,\ [\![\mathbf{fail}]\!]\, \sigma \rangle \rangle \\
= \quad & \iota_{\mathsf{out}} \langle 3,\ \iota_{\mathsf{out}} \langle 6,\ \iota_{\mathsf{abort}}\, \sigma \rangle \rangle
\end{aligned}
$$

$$f_* \, (\iota_{\mathsf{out}} \langle n,\ \omega \rangle) = \iota_{\mathsf{out}} \langle n,\ f_* \, \omega \rangle$$

$$f_* \, (\iota_{\mathsf{term}}\, \sigma) = f\, \sigma$$

Products of Predomains

If $P_1, \ldots, P_n$ are predomains, then $P_1 \times \ldots \times P_n$ is the predomain over their Cartesian product

$$\{\, \langle x_1, \ldots, x_n \rangle \mid x_1 \in P_1 \text{ and } \ldots \text{ and } x_n \in P_n \,\}$$

with the induced componentwise partial order

$$\langle x_1, \ldots, x_n \rangle \sqsubseteq \langle y_1, \ldots, y_n \rangle \iff x_1 \sqsubseteq_1 y_1 \text{ and } \ldots \text{ and } x_n \sqsubseteq_n y_n$$

and limit

$$\bigsqcup_{i=0}^{\infty} \langle x_1^{(i)}, \ldots, x_n^{(i)} \rangle = \langle \bigsqcup_{i=0}^{\infty}{}^{1}\, x_1^{(i)}, \ldots, \bigsqcup_{i=0}^{\infty}{}^{n}\, x_n^{(i)} \rangle$$

If P_k are domains, then $\langle \perp_1, \ldots, \perp_n \rangle$ is the least element of $P_1 \times \ldots \times P_n$.

Then the projections π_k^n are continuous functions, and if f_i are continuous, then so are $f_1 \otimes \ldots \otimes f_n$ and $f_1 \times \ldots \times f_n$.

Sums of Predomains

If $P_1, \ldots, P_n$ are predomains, then $P_1 + \ldots + P_n$ is the predomain over their sum

$$\{ \langle 0, x\rangle \mid x \in P_1 \} \cup \ldots \cup \{ \langle n-1,\, x\rangle \mid x \in P_n \}$$

ordered by the injected partial orders of the components:

$$\langle i, x\rangle \sqsubseteq \langle j, y\rangle \iff i = j \text{ and } x \sqsubseteq_i y.$$

All elements in a chain in $P_1 + \ldots + P_n$ have the same tag, and the limit is

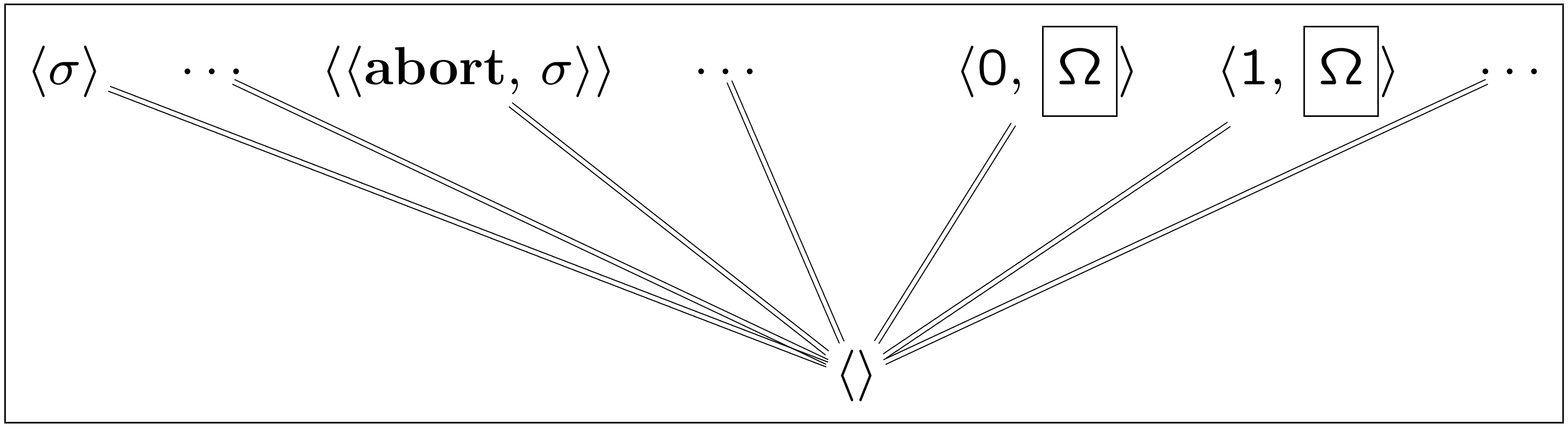
$$\bigsqcup_{i=0}^{\infty} \langle j, x_i\rangle = \langle j,\, \bigsqcup_{i=0}^{\infty}{}^{j} x_i\rangle$$

$P_1 + \ldots + P_n$ is a domain only if $n = 1$ and P_1 is a domain.

The injections ι_k^n are continuous functions, and if f_i are continuous, then so are $f_1 \oplus \ldots \oplus f_n$ and $f_1 + \ldots + f_n$.

Recursive Isomorphism for the Domain of Outputs

$\Omega \quad \cong$



$$\Omega \cong (\Sigma + \Sigma + \mathbf{Z} \times \Omega)_\perp$$

$$\exists \left\{ \begin{array}{l} \phi \in \Omega \to (\Sigma + \Sigma + \mathbf{Z} \times \Omega)_\perp \\ \psi \in (\Sigma + \Sigma + \mathbf{Z} \times \Omega)_\perp \to \Omega \end{array} \right\} \text{ such that } \begin{array}{l} \psi \cdot \phi = I_\Omega \\ \phi \cdot \psi = I_{(\Sigma + \Sigma + \mathbf{Z} \times \Omega)_\perp} \end{array}$$

$$\begin{array}{rcllcl} \iota_{\mathsf{term}} & = & \psi \cdot \iota_\uparrow \cdot \iota_0 & \in & \Sigma \to \Omega \\ \iota_{\mathsf{abort}} & = & \psi \cdot \iota_\uparrow \cdot \iota_1 & \in & \Sigma \to \Omega \\ \iota_{\mathsf{out}} & = & \psi \cdot \iota_\uparrow \cdot \iota_2 & \in & \mathbf{Z} \times \Omega \to \Omega \end{array}$$

Intermediate Input: the Domain of Resumptions

Syntax:

$$comm ::= \mathtt{?}\,var$$

Domain of program behaviors $\Omega \ni \omega$:

- $\omega = \bot \Rightarrow$ the program runs forever without output or input
- $\omega = \iota_{\mathsf{term}}\,\sigma \Rightarrow$ the program terminates normally in state σ
- $\omega = \iota_{\mathsf{abort}}\,\sigma \Rightarrow$ the program fails in state σ
- $\omega = \iota_{\mathsf{out}}\,\langle n, \omega' \rangle \Rightarrow$ the program outputs n and then has behavior ω'
- for $g \in \mathbf{Z} \to \Omega$: $\omega = \iota_{\mathsf{in}}\,g \Rightarrow$ if the program inputs n, it has behavior $g\,n$.

$$\Omega \cong (\Sigma + \Sigma + (\mathbf{Z} \times \Omega) + (\mathbf{Z} \to \Omega))_\bot$$

$$\iota_{\mathsf{in}} = \psi \cdot \iota_\uparrow \cdot \iota_3 \in (\mathbf{Z} \to \Omega) \to \Omega$$

Semantics of Intermediate Input

$$[\![?v]\!]_{comm}\,\sigma \;=\; \iota_{\mathsf{in}}\,(\lambda n \in \mathbf{Z}.\,\iota_{\mathsf{term}}\,[\sigma \,|\, v : n])$$

$$\begin{aligned}
f_*\bot &= \bot \\
f_*(\iota_{\mathsf{term}}\,\sigma) &= f\,\sigma \\
f_*(\iota_{\mathsf{abort}}\,\sigma) &= \iota_{\mathsf{abort}}\,\sigma \\
f_*(\iota_{\mathsf{out}}\,\langle n,\, \omega\rangle) &= \iota_{\mathsf{out}}\,\langle n,\, f_*\,\omega\rangle \\
f_*(\iota_{\mathsf{in}}\,g) &= \iota_{\mathsf{in}}\,(\lambda n \in \mathbf{Z}.\, f_*\,(g\,n))
\end{aligned}$$

$$\begin{aligned}
f_\dagger\bot &= \bot \\
f_\dagger(\iota_{\mathsf{term}}\,\sigma) &= \iota_{\mathsf{term}}\,(f\,\sigma) \\
f_\dagger(\iota_{\mathsf{abort}}\,\sigma) &= \iota_{\mathsf{abort}}\,(f\,\sigma) \\
f_\dagger(\iota_{\mathsf{out}}\,\langle n,\, \omega\rangle) &= \iota_{\mathsf{out}}\,\langle n,\, f_\dagger\,\omega\rangle \\
f_\dagger(\iota_{\mathsf{in}}\,g) &= \iota_{\mathsf{in}}\,(f_\dagger \cdot g)
\end{aligned}$$

$$\begin{aligned}
[\![?\mathsf{x}\;;\;!\mathsf{x}]\!]\sigma &= [\![!\mathsf{x}]\!]_*\,([\![?\mathsf{x}]\!]\,\sigma) \\
&= [\![!\mathsf{x}]\!]_*\,(\iota_{\mathsf{in}}\,(\lambda n \in \mathbf{Z}.\,\iota_{\mathsf{term}}\,[\sigma \,|\, \mathsf{x} : n])) \\
&= \iota_{\mathsf{in}}\,(\lambda n \in \mathbf{Z}.\,[\![!\mathsf{x}]\!]_*\,(\iota_{\mathsf{term}}\,[\sigma \,|\, \mathsf{x} : n])) \\
&= \iota_{\mathsf{in}}\,(\lambda n \in \mathbf{Z}.\,[\![!\mathsf{x}]\!]\,[\sigma \,|\, \mathsf{x} : n]) \\
&= \iota_{\mathsf{in}}\,(\lambda n \in \mathbf{Z}.\,\iota_{\mathsf{out}}\,\langle [\![\mathsf{x}]\!]\,[\sigma \,|\, \mathsf{x} : n],\, \iota_{\mathsf{term}}\,[\sigma \,|\, \mathsf{x} : n]\rangle) \\
&= \iota_{\mathsf{in}}\,(\lambda n \in \mathbf{Z}.\,\iota_{\mathsf{out}}\,\langle n,\, \iota_{\mathsf{term}}\,[\sigma \,|\, \mathsf{x} : n]\rangle)
\end{aligned}$$

Continuation Semantics

In an implementation of $c_0 \; ; \; c_1$,
the semantics of c_1 has no bearing on the result if c_0 fails to terminate
$\Rightarrow$ the semantics of c_0 determines whether c_1 will be executed or not.

But from the direct semantics of sequencing

$$[\![c_0 \; ; \; c_1]\!] \, \sigma = [\![c_1]\!]_* \, ([\![c_0]\!] \, \sigma)$$

it looks as if the semantics of c_1 determines the result;
much machinery hidden in $(-)_*$ to rectify this.

The semantics of output

- $\omega = \iota_{\mathsf{out}} \, \langle n, \omega' \rangle \Rightarrow$ the program outputs n and then has behavior ω'

also suggests it would be easier to explain a behavior in terms of what to do next,
or its continuation behavior.

Continuation Semantics cont'd

Idea: let the semantic function take an extra argument $\kappa \in \Sigma \to \Omega$ which describes the behavior of the rest of the program. Then

$$\llbracket - \rrbracket_{comm} \in comm \to (\Sigma \to \Omega) \to \Sigma \to \Omega$$

$$\begin{aligned}
\llbracket \mathbf{skip} \rrbracket\, \kappa\, \sigma &= \kappa\, \sigma \\
\llbracket v := e \rrbracket\, \kappa\, \sigma &= \kappa\, [\sigma \mid v : \llbracket e \rrbracket_{intexp} \sigma] \\
\llbracket \mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{else}\ c' \rrbracket\, \kappa\, \sigma &= \mathit{if}\ \llbracket b \rrbracket_{assert} \sigma\ \mathit{then}\ \llbracket c \rrbracket\, \kappa\, \sigma\ \mathit{else}\ \llbracket c' \rrbracket\, \kappa\, \sigma \\
\llbracket c_0 \,;\, c_1 \rrbracket\, \kappa\, \sigma &= \llbracket c_0 \rrbracket\, (\lambda \sigma' \in \Sigma.\, \llbracket c_1 \rrbracket\, \kappa\, \sigma')\, \sigma \\
&= \llbracket c_0 \rrbracket\, (\llbracket c_1 \rrbracket\, \kappa)\, \sigma \\
\text{i.e. } \llbracket c_0 \,;\, c_1 \rrbracket &= \llbracket c_0 \rrbracket \cdot \llbracket c_1 \rrbracket \\
\llbracket \mathbf{while}\ b\ \mathbf{do}\ c \rrbracket\, \kappa &= \mathbf{Y}_{\Sigma \to \Omega}\, F \text{ where } F\, \kappa'\, \sigma = \mathit{if}\ \llbracket b \rrbracket \sigma\ \mathit{then}\ \llbracket c \rrbracket\, \kappa'\, \sigma\ \mathit{else}\ \kappa\, \sigma \\
\llbracket \mathbf{newvar}\ v := e\ \mathbf{in}\ c \rrbracket\, \kappa\, \sigma &= \llbracket c \rrbracket\, (\lambda \sigma' \in \Sigma.\, \kappa\, [\sigma' \mid v : \sigma v])\, [\sigma \mid v : \llbracket e \rrbracket \sigma]
\end{aligned}$$

$$\llbracket c \rrbracket^{cont}_{comm}\, \kappa\, \sigma = \kappa_{\perp\!\!\perp}\, (\llbracket c \rrbracket^{direct}_{comm}\, \sigma), \text{ in particular } \llbracket c \rrbracket^{direct}_{comm} = \llbracket c \rrbracket^{cont}_{comm}\, \iota_{\uparrow}$$

Continuation Semantics

Idea: let the semantic function take an extra argument $\kappa \in K$ (where $K \overset{\text{def}}{=} \Sigma \to \Omega$) which is its continuation: it describes the behavior of the rest of the program, produces an answer in Ω when applied to an initial state in Σ.

$$[\![-]\!]_{comm} \in comm \to (\Sigma \to \Omega) \to \Sigma \to \Omega$$

i.e. the semantics of a command maps continuations to continuations:

$$\begin{aligned}
[\![-]\!]_{comm} &\in comm \to K \to K \\
[\![\mathbf{skip}]\!]\,\kappa &= \lambda\sigma \in \Sigma.\, \kappa\,\sigma \\
&= \kappa \\
\text{i.e. } [\![\mathbf{skip}]\!] &= I_K \\
[\![v := e]\!]\,\kappa &= \lambda\sigma \in \Sigma.\, \kappa\,[\sigma \mid v : [\![e]\!]_{intexp}\sigma] \\
[\![c_0\ ;\ c_1]\!]\,\kappa &= \lambda\sigma \in \Sigma.\, [\![c_0]\!]\,(\lambda\sigma' \in \Sigma.\, [\![c_1]\!]\,\kappa\,\sigma')\,\sigma \\
&= \lambda\sigma \in \Sigma.\, [\![c_0]\!]\,([\![c_1]\!]\,\kappa)\,\sigma \\
&= [\![c_0]\!]\,([\![c_1]\!]\,\kappa) \\
\text{i.e. } [\![c_0\ ;\ c_1]\!] &= [\![c_0]\!] \cdot [\![c_1]\!]
\end{aligned}$$

More Continuation Semantics

$$[\![\textbf{if } b \textbf{ then } c \textbf{ else } c']\!]\,\kappa = \lambda\sigma \in \Sigma.\ \textit{if }\ [\![b]\!]_{assert}\sigma\ \textit{then }\ [\![c]\!]\,\kappa\,\sigma\ \textit{else }\ [\![c']\!]\,\kappa\,\sigma$$

$$\begin{aligned}
[\![\textbf{while } b \textbf{ do } c]\!]\,\kappa &= [\![\textbf{if } b \textbf{ then } (c\ ;\ \textbf{while } b \textbf{ do } c) \textbf{ else skip}]\!]\,\kappa \\
&= \lambda\sigma \in \Sigma.\ \textit{if }\ [\![b]\!]_{assert}\sigma\ \textit{then }\ ([\![c]\!] \cdot [\![\textbf{while } b \textbf{ do } c]\!])\,\kappa\,\sigma\ \textit{else }\ \kappa\,\sigma \\
&= \lambda\sigma \in \Sigma.\ \textit{if }\ [\![b]\!]_{assert}\sigma\ \textit{then }\ [\![c]\!]\,([\![\textbf{while } b \textbf{ do } c]\!]\,\kappa)\,\sigma\ \textit{else }\ \kappa\,\sigma \\
&= F\,([\![\textbf{while } b \textbf{ do } c]\!]\,\kappa),\ \text{where} \\
&\ F\,\kappa' \ = \ \lambda\sigma \in \Sigma.\ \textit{if }\ [\![b]\!]_{assert}\sigma\ \textit{then }\ [\![c]\!]\,\kappa'\,\sigma\ \textit{else }\ \kappa\,\sigma
\end{aligned}$$

$$[\![\textbf{while } b \textbf{ do } c]\!]\,\kappa = \mathbf{Y}_{\Sigma\to\Omega}\,F\ \text{where}\ F\,\kappa'\,\sigma = \textit{if }\ [\![b]\!]\sigma\ \textit{then }\ [\![c]\!]\,\kappa'\,\sigma\ \textit{else }\ \kappa\,\sigma$$

$$[\![\textbf{newvar } v := e \textbf{ in } c]\!]\,\kappa \ = \ \lambda\sigma \in \Sigma.\ [\![c]\!]\,(\lambda\sigma' \in \Sigma.\,\kappa\,[\sigma' \mid v : \sigma v])\,[\sigma \mid v : [\![e]\!]\sigma]$$

Relationship Between Direct and Continuation Semantics

The connection is that

$$\begin{aligned} [\![c]\!]^{cont}_{comm}\,\kappa\,\sigma &= \kappa_{\perp\!\!\perp}\,([\![c]\!]^{direct}_{comm}\,\sigma) \\ \text{i.e. } [\![c]\!]^{cont}_{comm}\,\kappa &= \kappa_{\perp\!\!\perp}\cdot[\![c]\!]^{direct}_{comm} \end{aligned}$$

which can be shown by structural induction on $comm$, e.g.

$$[\![\mathbf{skip}]\!]^{cont}\,\kappa = I_K\,\kappa = \kappa \qquad \kappa_{\perp\!\!\perp}\cdot[\![\mathbf{skip}]\!]^{direct} = \kappa_{\perp\!\!\perp}\cdot I_\Sigma = \kappa$$

$$\begin{aligned} [\![c\ ;\ c']\!]^{cont}\,\kappa &= [\![c]\!]^{cont}\,([\![c']\!]^{cont}\,\kappa) = [\![c]\!]^{cont}\,(\kappa_{\perp\!\!\perp}\cdot[\![c']\!]^{direct}) \\ &= (\kappa_{\perp\!\!\perp}\cdot[\![c']\!]^{direct})_{\perp\!\!\perp}\cdot[\![c]\!]^{direct} \\ &= \kappa_{\perp\!\!\perp}\cdot([\![c']\!]^{direct})_{\perp\!\!\perp}\cdot[\![c]\!]^{direct} = \kappa_{\perp\!\!\perp}\cdot[\![c\ ;\ c']\!]^{direct} \end{aligned}$$

When the “final” (or “top-level”) continuation is the injection $\iota_\uparrow \in \Sigma \to \Sigma_\perp$, then

$$\begin{aligned} [\![c]\!]^{cont}_{comm}\,\iota_\uparrow &= (\iota_\uparrow)_{\perp\!\!\perp}\cdot[\![c]\!]^{direct}_{comm} \\ \text{i.e. } [\![c]\!]^{direct}_{comm} &= [\![c]\!]^{cont}_{comm}\,\iota_\uparrow \end{aligned}$$

Continuation Semantics of Extensions

For input and output,

$$\begin{aligned}
[\![\,!e]\!]\,\kappa &= \lambda\sigma \in \Sigma.\, \iota_{\mathsf{out}}\,\langle [\![e]\!]_{\mathit{intexp}}\sigma,\ \kappa\,\sigma\rangle \\
[\![?v]\!]\,\kappa &= \lambda\sigma \in \Sigma.\, \iota_{\mathsf{in}}\,(\lambda n \in \mathbf{Z}.\, \kappa\,[\sigma \,|\, v : n])
\end{aligned}$$

The relationship between direct and continuation semantics is then

$$\begin{aligned}
[\![c]\!]^{\mathit{cont}}_{\mathit{comm}}\,\kappa\,\sigma &= \kappa_*\,([\![c]\!]^{\mathit{direct}}_{\mathit{comm}}\,\sigma) \\
\text{or } [\![c]\!]^{\mathit{cont}}_{\mathit{comm}}\,\kappa &= \kappa_* \cdot [\![c]\!]^{\mathit{direct}}_{\mathit{comm}}
\end{aligned}$$

Failure ignores the given continuation and directly produces a result
⇒ one might expect

$$[\![\mathbf{fail}]\!]\,\kappa\,\sigma = \iota_{\mathsf{term}}\,\sigma$$

but this does not work: local variables are not reset to their original bindings.

Continuation Semantics of Failure

So we have to introduce a second, abortive continuation,
which the semantics of failure invokes and of local declarations augments:

$$
\begin{array}{rcl}
\llbracket - \rrbracket_{comm} & \in & comm \to K \to K \to K \\
\llbracket \mathbf{skip} \rrbracket \, \kappa_t \, \kappa_f & = & \kappa_t \\
\llbracket v := e \rrbracket \, \kappa_t \, \kappa_f & = & \lambda\sigma \in \Sigma.\, \kappa_t \, [\sigma \mid v : \llbracket e \rrbracket_{intexp} \sigma] \\
\llbracket c_0 \,;\, c_1 \rrbracket \, \kappa_t \, \kappa_f & = & \llbracket c_0 \rrbracket \, (\llbracket c_1 \rrbracket \, \kappa_t \, \kappa_f) \, \kappa_f \\
\llbracket \mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{else}\ c' \rrbracket \, \kappa_t \, \kappa_f & = & \lambda\sigma \in \Sigma.\, \mathit{if}\ \llbracket b \rrbracket_{assert} \sigma\ \mathit{then}\ \llbracket c \rrbracket \, \kappa_t \, \kappa_f \, \sigma\ \mathit{else}\ \llbracket c' \rrbracket \, \kappa_t \, \kappa_f \, \sigma \\
\llbracket \mathbf{while}\ b\ \mathbf{do}\ c \rrbracket \, \kappa_t \, \kappa_f & = & \mathbf{Y}_{\Sigma \to \Omega} \, F \\
 & & \text{where } F \, \kappa' \, \sigma = \mathit{if}\ \llbracket b \rrbracket \sigma\ \mathit{then}\ \llbracket c \rrbracket \, \kappa' \, \kappa_f \, \sigma\ \mathit{else}\ \kappa_t \, \sigma \\
\llbracket \mathbf{newvar}\ v := e\ \mathbf{in}\ c \rrbracket \, \kappa_t \, \kappa_f & = & \lambda\sigma \in \Sigma.\, \llbracket c \rrbracket \, (\lambda\sigma' \in \Sigma.\, \kappa_t \, [\sigma' \mid v : \sigma v]) \\
 & & \qquad\qquad (\lambda\sigma' \in \Sigma.\, \kappa_f \, [\sigma' \mid v : \sigma v]) \, [\sigma \mid v : \llbracket e \rrbracket \sigma] \\
\llbracket\, !e \rrbracket \, \kappa_t \, \kappa_f & = & \lambda\sigma \in \Sigma.\, \iota_{\mathsf{out}} \, \langle \llbracket e \rrbracket_{intexp} \sigma,\ \kappa_t \, \sigma \rangle \\
\llbracket ?v \rrbracket \, \kappa_t \, \kappa_f & = & \lambda\sigma \in \Sigma.\, \iota_{\mathsf{in}} \, (\lambda n \in \mathbf{Z}.\, \kappa_t \, [\sigma \mid v : n]) \\
\llbracket \mathbf{fail} \rrbracket \, \kappa_t \, \kappa_f & = & \kappa_f
\end{array}
$$