The Simple Imperative Language

$$
\begin{aligned}
intexp ::= &\ \mathsf{0} \mid \mathsf{1} \mid \ldots \\
\mid &\ var \\
\mid &\ \mathsf{-}intexp \mid intexp\mathsf{+}intexp \mid intexp\mathsf{-}intexp \mid \ldots \\
boolexp ::= &\ \mathbf{true} \mid \mathbf{false} \\
\mid &\ intexp = intexp \mid intexp < intexp \mid intexp \leq intexp \mid \ldots \\
\mid &\ \neg boolexp \mid boolexp \wedge boolexp \mid boolexp \vee boolexp \mid \ldots
\end{aligned}
$$

(no quantified terms)

$$
\begin{aligned}
comm ::= &\ var \mathsf{:=} intexp \\
\mid &\ \mathbf{skip} \\
\mid &\ comm \ \mathbf{;} \ comm \\
\mid &\ \mathbf{if}\ boolexp\ \mathbf{then}\ comm\ \mathbf{else}\ comm \\
\mid &\ \mathbf{while}\ boolexp\ \mathbf{do}\ comm
\end{aligned}
$$

(may fail to terminate)

Denotational Semantics of SIL

$$[\![-]\!]_{intexp} \in intexp \rightarrow \mathbf{\Sigma} \rightarrow \mathbf{Z}$$

$$[\![-]\!]_{boolexp} \in boolexp \rightarrow \mathbf{\Sigma} \rightarrow \mathbf{B}$$

$$[\![-]\!]_{comm} \in comm \rightarrow \mathbf{\Sigma} \rightarrow \mathbf{\Sigma}_{\perp}$$

$\mathbf{\Sigma} = var \rightarrow \mathbf{Z}$

(simpler than $[\![-]\!]_{assert}$)

$\mathbf{\Sigma}_{\perp} \stackrel{\text{def}}{=} \mathbf{\Sigma} \cup \{\perp\}$ (divergence)

Denotational Semantics of SIL

$$[\![-]\!]_{intexp} \in intexp \to \Sigma \to \mathbf{Z}$$

$$[\![-]\!]_{boolexp} \in boolexp \to \Sigma \to \mathbf{B}$$

$$[\![-]\!]_{comm} \in comm \to \Sigma \to \Sigma_{\perp}$$

$$\Sigma = var \to \mathbf{Z}$$

(simpler than $[\![-]\!]_{assert}$)

$\Sigma_{\perp} \stackrel{\text{def}}{=} \Sigma \cup \{\perp\}$ (divergence)

$$[\![v := e]\!]_{comm}\sigma = [\sigma \mid v : [\![e]\!]_{intexp}\sigma]$$

$$\begin{aligned} &[\![\mathsf{x} := \mathsf{x} * 6]\!]_{comm}[\mathsf{x} : 7] \\ &= [\mathsf{x} : 7 \mid \mathsf{x} : [\![\mathsf{x} * 6]\!]_{intexp}[\mathsf{x} : 7]] \\ &= [\mathsf{x} : 7 \mid \mathsf{x} : 42] \\ &= [\mathsf{x} : 42] \end{aligned}$$

Denotational Semantics of SIL

$$[\![-]\!]_{intexp} \in intexp \to \mathbf{\Sigma} \to \mathbf{Z} \qquad \Sigma = var \to \mathbf{Z}$$

$$[\![-]\!]_{boolexp} \in boolexp \to \mathbf{\Sigma} \to \mathbf{B} \qquad \text{(simpler than } [\![-]\!]_{assert}\text{)}$$

$$[\![-]\!]_{comm} \in comm \to \mathbf{\Sigma} \to \mathbf{\Sigma}_{\perp} \qquad \mathbf{\Sigma}_{\perp} \stackrel{\text{def}}{=} \mathbf{\Sigma} \cup \{\perp\} \text{ (divergence)}$$

$$[\![v := e]\!]_{comm}\sigma = [\sigma \mid v : [\![e]\!]_{intexp}\sigma]$$

$$\begin{aligned}
&[\![\mathsf{x := x*6}]\!]_{comm}[\mathsf{x} : 7] \\
&= [\mathsf{x} : 7 \mid \mathsf{x} : [\![\mathsf{x*6}]\!]_{intexp}[\mathsf{x} : 7]] \\
&= [\mathsf{x} : 7 \mid \mathsf{x} : 42] \\
&= [\mathsf{x} : 42]
\end{aligned}$$

$$[\![\mathbf{skip}]\!]_{comm}\sigma = \sigma$$

Denotational Semantics of SIL

$$[\![-]\!]_{intexp} \in \; intexp \to \mathbf{\Sigma} \to \mathbf{Z} \qquad \Sigma = var \to \mathbf{Z}$$

$$[\![-]\!]_{boolexp} \in boolexp \to \mathbf{\Sigma} \to \mathbf{B} \qquad \text{(simpler than } [\![-]\!]_{assert}\text{)}$$

$$[\![-]\!]_{comm} \in \; comm \to \mathbf{\Sigma} \to \mathbf{\Sigma}_{\perp} \qquad \Sigma_{\perp} \stackrel{\text{def}}{=} \Sigma \cup \{\perp\} \;\; \text{(divergence)}$$

$$[\![v := e]\!]_{comm}\sigma \quad = \quad [\sigma \,|\, v : [\![e]\!]_{intexp}\sigma]$$

$$\begin{aligned} &[\![\mathsf{x := x{*}6}]\!]_{comm}[\mathsf{x} : 7] \\ &= [\mathsf{x} : 7 \,|\, \mathsf{x} : [\![\mathsf{x{*}6}]\!]_{intexp}[\mathsf{x} : 7]] \\ &= [\mathsf{x} : 7 \,|\, \mathsf{x} : 42] \\ &= [\mathsf{x} : 42] \end{aligned}$$

$$[\![\mathbf{skip}]\!]_{comm}\sigma \quad = \quad \sigma$$

$$[\![c \; ; \; c']\!]_{comm}\sigma \quad = \quad [\![c']\!]_{comm}\,([\![c]\!]_{comm}\sigma)$$

Denotational Semantics of SIL

$$[\![-]\!]_{intexp} \in intexp \to \Sigma \to \mathbf{Z} \qquad \Sigma = var \to \mathbf{Z}$$

$$[\![-]\!]_{boolexp} \in boolexp \to \Sigma \to \mathbf{B} \qquad \text{(simpler than } [\![-]\!]_{assert}\text{)}$$

$$[\![-]\!]_{comm} \in comm \to \Sigma \to \Sigma_\perp \qquad \Sigma_\perp \stackrel{\text{def}}{=} \Sigma \cup \{\perp\} \ \text{(divergence)}$$

$$[\![v := e]\!]_{comm}\sigma = [\sigma \mid v : [\![e]\!]_{intexp}\sigma]$$

$$\begin{aligned} &[\![\mathsf{x := x*6}]\!]_{comm}[\mathsf{x : 7}] \\ &= [\mathsf{x : 7} \mid \mathsf{x} : [\![\mathsf{x*6}]\!]_{intexp}[\mathsf{x : 7}]] \\ &= [\mathsf{x : 7} \mid \mathsf{x : 42}] \\ &= [\mathsf{x : 42}] \end{aligned}$$

$$[\![\mathbf{skip}]\!]_{comm}\sigma = \sigma$$

$$[\![c \, ; \, c']\!]_{comm}\sigma \stackrel{\text{NOT!}}{=} [\![c']\!]_{comm}\,(\underbrace{[\![c]\!]_{comm}\sigma}_{=\ \perp \text{ if } c \text{ fails to terminate}})$$

Semantics of Sequential Composition

We can extend $f \in S \to T_\perp$ to $f_{\perp\!\!\perp} \in S_\perp \to T_\perp$:

$$f_{\perp\!\!\perp}\, x \overset{\text{def}}{=} \begin{cases} \perp, & \text{if } x = \perp \\ f\ x, & \text{otherwise} \end{cases}$$

This defines $\quad (-)_{\perp\!\!\perp} \in (S \to T_\perp) \to S_\perp \to T_\perp$

(a special case of the Kleisli monadic operator).

So

$$\begin{aligned} [\![-]\!]_{comm} &\in comm \to \Sigma \to \Sigma_\perp \\ \Rightarrow [\![c']\!]_{comm} &\in \Sigma \to \Sigma_\perp \\ \Rightarrow ([\![c']\!]_{comm})_{\perp\!\!\perp} &\in \Sigma_\perp \to \Sigma_\perp \end{aligned}$$

$$[\![c\ ;\ c']\!]_{comm}\sigma = ([\![c']\!]_{comm})_{\perp\!\!\perp}([\![c]\!]_{comm}\sigma)$$

Semantics of Conditionals

$$[\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!]_{comm}\sigma = \begin{cases} [\![c_0]\!]_{comm}\sigma, & \text{if } [\![b]\!]_{boolexp}\sigma = \textbf{true} \\ [\![c_1]\!]_{comm}\sigma, & \text{if } [\![b]\!]_{boolexp}\sigma = \textbf{false} \end{cases}$$

Example:

$$[\![\textbf{if } \mathsf{x{<}0} \textbf{ then } \mathsf{x{:=}{-}x} \textbf{ else skip}]\!]_{comm}[\mathsf{x} : -3]$$

$$= [\![\mathsf{x{:=}{-}x}]\!]_{comm}[\mathsf{x} : -3], \qquad \text{since } [\![\mathsf{x{<}0}]\!]_{boolexp}[\mathsf{x} : -3] = \textbf{true}$$

$$= [\mathsf{x} : -3 \mid \mathsf{x} : [\![\mathsf{{-}x}]\!]_{intexp}[\mathsf{x} : -3]]$$

$$= [\mathsf{x} : 3]$$

$$[\![\textbf{if } \mathsf{x{<}0} \textbf{ then } \mathsf{x{:=}{-}x} \textbf{ else skip}]\!]_{comm}[\mathsf{x} : 5]$$

$$= [\![\textbf{skip}]\!]_{comm}[\mathsf{x} : 5], \qquad \text{since } [\![\mathsf{x{<}0}]\!]_{boolexp}[\mathsf{x} : 5] = \textbf{false}$$

$$= [\mathsf{x} : 5]$$

Problems with the Semantics of Loops

Idea: define the meaning of **while** b **do** c as that of

$$\textbf{if } b \textbf{ then } (c \,;\, \textbf{while } b \textbf{ do } c) \textbf{ else skip}$$

But the equation

$$\begin{aligned}
&[\![\textbf{while } b \textbf{ do } c]\!]_{comm}\sigma \\
&= [\![\textbf{if } b \textbf{ then } (c \,;\, \textbf{while } b \textbf{ do } c) \textbf{ else skip}]\!]_{comm}\sigma \\
&= \begin{cases} ([\![\textbf{while } b \textbf{ do } c]\!]_{comm})_{\perp\!\!\perp}([\![c]\!]_{comm}\sigma), & \text{if } [\![b]\!]_{boolexp}\sigma = \textbf{true} \\ \sigma, & \text{otherwise} \end{cases}
\end{aligned}$$

is not syntax directed and sometimes has infinitely many solutions:

$[\![\textbf{while true do}\; \mathtt{x:=x+1}]\!]_{comm} = \lambda\sigma : \Sigma.\, \sigma'$ is a solution for any σ'.

Partially Ordered Sets

A relation ρ is	reflexive on S	iff	$\forall x \in S.\, x\rho x$
	transitive	iff	$x\rho y \;\&\; y\rho z \Rightarrow x\rho z$
	antisymmetric	iff	$x\rho y \;\&\; y\rho x \Rightarrow x = y$
	symmetric	iff	$x\rho y \Rightarrow y\rho x$

$\sqsubseteq$ is reflexive on P & transitive	$\Rightarrow \sqsubseteq$ is a preorder on P
$\sqsubseteq$ is a preorder on P & antisymmetric	$\Rightarrow \sqsubseteq$ is a partial order on P
P with a partial order $\sqsubseteq$ on P	$\Rightarrow$a poset P
P with I_P as a partial order on P	$\Rightarrow$a discretely ordered P
$f \in P \rightarrow P'$ & $\forall x, y \in P.\, (x \sqsubseteq y \Rightarrow fx \sqsubseteq' fy)$	$\Rightarrow f$ is monotone from P to P'
$y \in P :\ \forall X \subseteq P. \forall x \in X.\, x \sqsubseteq y$	$\Rightarrow y$ is an upper bound of X

Least Upper Bounds

y is a lub of $X \subseteq P$ if y is an upper bound of X
and $\forall z \in P.\ (z$ is an upper bound of $X \Rightarrow y \sqsubseteq z)$

If P is a poset and $X \subseteq P$, there is at most one lub $\bigsqcup X$ of X.

$\bigsqcup\{\} = \bot$ — the least element of P (when it exists).

Let $\mathcal{X} \subseteq \mathcal{P}\ P$ such that $\bigsqcup X$ exists for every $X \in \mathcal{X}$. Then

$$\bigsqcup\{\bigsqcup X \mid X \in \mathcal{X}\} = \bigsqcup \bigcup \mathcal{X}$$

if either of these lubs exists. In particular

$$\bigsqcup_{i=0}^{\infty} \bigsqcup_{j=0}^{\infty} x_{ij} = \bigsqcup\{x_{ij} \mid i \in \mathbf{N} \text{ and } j \in \mathbf{N}\} = \bigsqcup_{j=0}^{\infty} \bigsqcup_{i=0}^{\infty} x_{ij}$$

if $\bigsqcup_{i=0}^{\infty} x_{ij}$ exist for all j, or $\bigsqcup_{j=0}^{\infty} x_{ij}$ exist for all i.

Domains

A chain is a countably infinite non-decreasing sequence $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$

The limit of a chain C is its lub $\bigsqcup C$ when it exists.

A chain C is interesting if $\bigsqcup C \notin C$.

(Chains with finitely many distinct elements are uninteresting.)

A poset P is a predomain (or complete partial order — cpo)
if P contains the limits of all its chains.

A predomain P is a domain (or pointed cpo) if P has a least element $\bot$.

In semantic domains, $\sqsubseteq$ is an order based on information content:

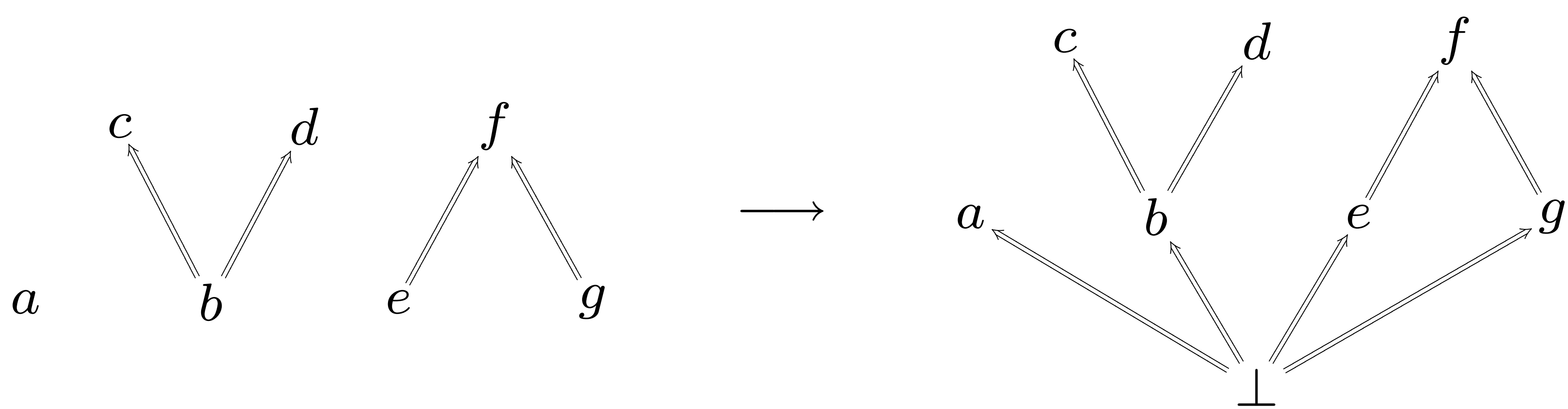
$x \sqsubseteq y$ (x approximates y, y is a refinement of x)

if x yields the same results as y in all contexts when it terminates,
but may diverge in more contexts.

Lifting

Any set S can be viewed as a predomain with discrete partial order $\sqsubseteq = I_S$.

The lifting $P_\perp$ of a predomain P is the domain $D = P \cup \{\perp\}$ where $\perp \notin P$, and $x \sqsubseteq_D y$ if $x = \perp$ or $x \sqsubseteq_P y$.



D is a flat domain if $D - \{\perp\}$ is discretely ordered by $\sqsubseteq$.

Continuous Functions

If P and P' are predomains, $f \in P \to P'$ is a continuous function from P to P' if it maps limits to limits:

$$f(\bigsqcup\{\, x_i \mid x_i \in C \,\}) = \bigsqcup'\{\, f\ x_i \mid x_i \in C \,\} \text{ for every chain } C \subseteq P$$

Continuous functions are monotone: consider chains $x \sqsubseteq y \sqsubseteq y \ldots$

There are non-continuous monotone functions:

Let $P \supseteq$ the interesting chain $C = (x_0 \sqsubseteq x_1 \sqsubseteq \ldots)$ with a limit x in P, and $P' = \{\bot, \top\}$ with $\bot \sqsubseteq' \top$. Then

$$f = \{\, [x_i, \bot] \mid x_i \in C \,\} \cup \{[x, \top]\}$$

is monotone but not continuous: $\sqcup'\{\, f\ x_i \mid x_i \in C \,\} = \bot \neq \top = f(\sqcup C)$

Monotone vs Continuous Functions

If $f \in P \to P'$ is monotone, then f is continuous

iff $f(\bigsqcup_i x_i) \sqsubseteq \bigsqcup'_i (f\ x_i)$ for all interesting chains x_i $(i \in \mathbf{N})$ in P.

Proof

[1ex] For uninteresting chains:

$$\text{if } \bigsqcup_i x_i = x_n, \text{ then } \bigsqcup_i{}' (f x_i) = f x_n = f(\bigsqcup_i x_i).$$

[1ex] For interesting chains: prove the opposite approximation:

$$(\forall i \in \mathbf{N}.\, x_i \sqsubseteq \bigsqcup_j x_j) \Rightarrow (\forall i \in \mathbf{N}.\, f x_i \sqsubseteq f(\bigsqcup_j x_j))$$
$$\Rightarrow \bigsqcup_i{}' (f x_i) \sqsubseteq f(\bigsqcup_i x_i)$$

The (Pre)domain of Continuous Functions

Pointwise ordering on functions in $P \to P'$ where P' is a predomain:

$$f \sqsubseteq_{\to} g \iff \forall x \in P.\, f\; x \sqsubseteq' g\; x$$

Proposition:

If both P and P' are predomains, then the set $[P \to P']$ of continuous functions from P to P' with partial order $\sqsubseteq_{\to}$ is a predomain with

$$\bigsqcup f_i = \lambda x \in P.\, \bigsqcup{}' (f_i x)$$

If P' is a domain, then $[P \to P']$ is a domain with $\bot_{\to} = \lambda x \in P.\, \bot'$

The (Pre)domain of Continuous Functions: Proof

To prove $[P \to P']$ is a predomain:

Let f_i be a chain in $[P \to P']$, and $f = \lambda x \in P. \bigsqcup' f_i x$.

($\bigsqcup' f_i x$ exists because $f_0 x \sqsubseteq' f_1 x \sqsubseteq' \ldots$ since $f_0 \sqsubseteq_{\to} f_1 \sqsubseteq_{\to} \ldots$
and P' is a predomain)

$f_i \sqsubseteq_{\to} f$ since $\forall x \in P.\ f_i x \sqsubseteq' f x$; hence f is an upper bound of $\{f_i\}$.

If g is such that $\forall i \in \mathbf{N}.\ f_i \sqsubseteq_{\to} g$, then $\forall x \in P.\ f_i x \sqsubseteq' g x$,

hence $\forall x \in P.\ f x \sqsubseteq' g x$, i.e. $f \sqsubseteq_{\to} g$.

$\Rightarrow$ f is the limit of f_i... but is f continuous so it is in $[P \to P']$?

Yes: If x_j is a chain in P, then

$$f(\bigsqcup_j x_j) = \bigsqcup_i{}' f_i(\bigsqcup_j x_j) = \bigsqcup_i{}' \bigsqcup_j{}' f_i x_j = \bigsqcup_j{}' \bigsqcup_i{}' f_i x_j = \bigsqcup_j{}' f x_j$$

Some Continuous Functions

For predomains P, P', P'',

- if $f \in P \to P'$ is a constant function, then $f \in [P \to P']$
- $I_P \in [P \to P]$
- if $f \in [P \to P']$ and $g \in [P' \to P'']$, then $g \cdot f \in [P \to P'']$
- if $f \in [P \to P']$, then $(- \cdot f) \in [[P' \to P''] \to [P \to P'']]$
- if $f \in [P' \to P'']$, then $(f \cdot -) \in [[P \to P'] \to [P \to P'']]$

Strict Functions and Lifting

If D and D' are domains, $f \in D \to D'$ is strict if $f\bot = \bot'$.

If P and P' are predomains and $f \in P \to P'$, then the strict function

$$f_\bot \stackrel{\text{def}}{=} \lambda x \in P_\bot. \begin{cases} f x, & \text{if } x \in P \\ \bot', & \text{if } x = \bot \end{cases}$$

is the lifting of f to $P_\bot \to P'_\bot$; if P' is a domain, then the strict function

$$f_{\bot\!\!\!\bot} \stackrel{\text{def}}{=} \lambda x \in P_\bot. \begin{cases} f x, & \text{if } x \in P \\ \bot', & \text{if } x = \bot \end{cases}$$

is the source lifting of f to $P_\bot \to P'$.

If f is continuous, so are $f_\bot$ and $f_{\bot\!\!\!\bot}$.

$(-)_\bot$ and $(-)_{\bot\!\!\!\bot}$ are also continuous.

Least Fixed-Point

If $f \in S \to S$, then $x \in S$ is a fixed-point of f if $x = fx$.

Theorem [Least Fixed-Point of a Continuous Function]

If D is a domain and $f \in [D \to D]$,
then $x \stackrel{\text{def}}{=} \bigsqcup_{i=0}^{\infty} f^i \bot$ is the least fixed-point of f.

Proof:

x exists because $\bot \sqsubseteq f\bot \sqsubseteq \ldots f^i\bot \sqsubseteq f^{i+1}\bot \sqsubseteq \ldots$ is a chain.

x is a fixed-point because

$$fx = f(\bigsqcup_{i=0}^{\infty} f^i\bot) = \bigsqcup_{i=0}^{\infty} f(f^i\bot) = \bigsqcup_{i=1}^{\infty} f^i\bot = \bigsqcup_{i=0}^{\infty} f^i\bot = x$$

For any fixed-point y of f, $\bot \sqsubseteq y \;\Rightarrow\; f\bot \sqsubseteq fy = y$,
by induction $\forall i \in \mathbf{N}.\ f^i\bot \sqsubseteq y$, therefore $x = \bigsqcup(f^i\bot) \sqsubseteq y$.

The Least Fixed-Point Operator

Let

$$\mathbf{Y}_D = \lambda f \in [D \to D].\ \bigsqcup_{i=0}^{\infty} f^i \bot$$

Then for each $f \in [D \to D]$, $\mathbf{Y}_D f$ is the least fixed-point of f.

$$\mathbf{Y}_D \in [[D \to D] \to D]$$

Semantics of Loops

The semantic equation

$$
\begin{aligned}
&[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]_{comm}\sigma \\
&\quad = \begin{cases} ([\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]_{comm})_{\perp\!\!\perp}([\![c]\!]_{comm}\sigma), & \text{if } [\![b]\!]_{boolexp}\sigma = \mathbf{true} \\ \sigma, & \text{otherwise} \end{cases}
\end{aligned}
$$

implies that $[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]_{comm}$ is a fixed-point of

$$
F \stackrel{\text{def}}{=} \lambda f \in [\Sigma \to \Sigma_\perp].\lambda\sigma \in \Sigma. \begin{cases} f_{\perp\!\!\perp}([\![c]\!]_{comm}\sigma), & \text{if } [\![b]\!]_{boolexp}\sigma = \mathbf{true} \\ \sigma, & \text{otherwise} \end{cases}
$$

We pick the least fixed-point:

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

Semantics of Loops: Intuition

$$w_0 \stackrel{\text{def}}{=} \textbf{while true do skip} \qquad [\![w_0]\!]_{comm} = \bot$$

$$w_{i+1} \stackrel{\text{def}}{=} \textbf{if } b \textbf{ then } (c \;;\; w_i) \textbf{ else skip} \quad [\![w_{i+1}]\!]_{comm} = F[\![w_i]\!]_{comm}$$

The loop **while** b **do** c behaves like w_i from state σ

if the loop evaluates the condition $n \leq i$ times:

$$[\![w_i]\!]_{comm}\sigma = \begin{cases} [\![\textbf{while } b \textbf{ do } c]\!]_{comm}\sigma, & \text{if } n \leq i \\ \bot, & \text{if } n > i \end{cases}$$

or the loop fails to terminate:

$$[\![\textbf{while } b \textbf{ do } c]\!]_{comm}\sigma = \bot = [\![w_i]\!]_{comm}\sigma.$$

So

$$\forall \sigma \in \Sigma.\ [\![\textbf{while } b \textbf{ do } c]\!]_{comm}\sigma = \bigsqcup_{n=0}^{\infty} [\![w_n]\!]_{comm}\sigma$$

$$\Rightarrow [\![\textbf{while } b \textbf{ do } c]\!]_{comm} = \mathbf{Y}_{[\Sigma \to \Sigma_\bot]} F$$

Variable Declarations

Syntax:

$$comm \;::=\; \textbf{newvar}\ var \mathrel{\textbf{:=}} intexp\ \textbf{in}\ comm$$

Semantics:

$$\begin{aligned}
&[\![\textbf{newvar}\ v \mathrel{\textbf{:=}} e\ \textbf{in}\ c]\!]_{comm}\sigma \\
&\quad \stackrel{\text{def}}{=} ([-\mid v : \sigma v])_{\perp\!\!\perp}([\![c]\!]_{comm}[\sigma \mid v : [\![e]\!]_{intexp}\sigma]) \\
&\quad = \begin{cases} \perp, & \text{if } \sigma' = \perp \\ [\sigma' \mid v : \sigma v], & \text{otherwise} \end{cases} \\
&\qquad\qquad \text{where } \sigma' = [\![c]\!]_{comm}[\sigma \mid v : [\![e]\!]_{intexp}\sigma]
\end{aligned}$$

$\textbf{newvar}\ v \mathrel{\textbf{:=}} e\ \textbf{in}\ c$ binds v in c, but not in e:

$$FV(\textbf{newvar}\ v \mathrel{\textbf{:=}} e\ \textbf{in}\ c) = (FV(c) - \{v\}) \cup FV(e)$$

Problems with Substitutions

Only variables are allowed on the left of assignment

$\Rightarrow$ substitution cannot be defined as for predicate logic:

$$(\texttt{x:=x+1})/\texttt{x} \to \texttt{10} \;=\; \texttt{10:=10+1}$$

We have to require $\delta \in \mathit{var} \to \mathit{var}$; then

$$
\begin{aligned}
(v := e)/\delta &= (\delta v) := (e/(c_{\mathsf{var}} \cdot \delta)) \\
(c_0 \,;\, c_1)/\delta &= (c_0/\delta) \,;\, (c_1/\delta) \\
&\cdots \\
(\mathbf{newvar}\ v := e\ \mathbf{in}\ c)/\delta &= \mathbf{newvar}\ u := (e/(c_{\mathsf{var}} \cdot \delta))\ \mathbf{in}\ (c/[\delta \mid v : u]) \\
&\quad \text{where } u \notin \{\delta w \mid w \in FV(c) - \{v\}\}
\end{aligned}
$$

Assigned Variables

Hence it is useful to know which variables are assigned to:

$$
\begin{array}{rcl}
FA(v := e) & = & \{v\} \\
FA(c_0 \,;\, c_1) & = & FA(c_0) \cup FA(c_1) \\
& \cdots & \\
FA(\mathbf{newvar}\; v := e \;\mathbf{in}\; c) & = & FA(c) - \{v\}
\end{array}
$$

Note that

$$FA(c) \subseteq FV(c)$$

Coincidence Theorem for Commands

The meaning of a command now depends

not only on the mapping of its free variables:

$$[\![c]\!]_{comm}\sigma v = \sigma v$$

if $[\![c]\!]_{comm}\sigma \neq \bot$ and $v \notin FV(c)$

(i.e. all non-free variables get the values they had before c was executed).

Coincidence Theorem:

(a) If $\sigma u = \sigma' u$ for all $u \in FV(c)$, then $[\![c]\!]_{comm}\sigma = \bot = [\![c]\!]_{comm}\sigma'$

or $\forall v \in FV(c).\ [\![c]\!]_{comm}\sigma v = [\![c]\!]_{comm}\sigma' v$.

(b) If $[\![c]\!]_{comm}\sigma \neq \bot$, then $[\![c]\!]_{comm}\sigma v = \sigma v$ for all $v \notin FA(c)$.

More Trouble with Substitutions

Recall that for predicate logic $[\![-]\!]([\![-]\!]_{intexp}\sigma \cdot \delta) = [\![-/\delta]\!]\sigma$.

The corresponding property for commands: $[\![-]\!](\sigma \cdot \delta) = [\![-/\delta]\!]\sigma \cdot \delta$;

fails in general due to aliasing:

$$\begin{array}{rcl} (\texttt{x:=x+1 ; y:=y*2})/[\mathsf{x} : \mathsf{z}|\mathsf{y} : \mathsf{z}] & = & (\texttt{z:=z+1 ; z:=z*2}) \\ [\mathsf{x} : 2 \,|\, \mathsf{y} : 2] & = & [\mathsf{z} : 2] \cdot [\mathsf{x} : \mathsf{z}|\mathsf{y} : \mathsf{z}] \end{array}$$

but

$$\begin{array}{rl} [\![\texttt{x:=x+1 ; y:=y*2}]\!]_{comm}[\mathsf{x} : 2 \,|\, \mathsf{y} : 2] & = [\mathsf{x} : 3 \,|\, \mathsf{y} : 4] \\ ([\![\texttt{z:=z+1 ; z:=z*2}]\!]_{comm}[\mathsf{z} : 2]) \cdot [\mathsf{x} : \mathsf{z}|\mathsf{y} : \mathsf{z}] & = [\mathsf{z} : 6] \cdot [\mathsf{x} : \mathsf{z}|\mathsf{y} : \mathsf{z}] \\ & = [\mathsf{x} : 6 \,|\, \mathsf{y} : 6] \end{array}$$

Substitution Theorem for Commands:

If $\delta \in var \rightarrow var$ and δ is an injection from a set $V \supseteq FV(c)$,

and σ and σ' are such that $\sigma' v = \sigma(\delta v)$ for all $v \in V$,

then $([\![c]\!]_{comm})\sigma' v = ([\![c/\delta]\!]_{comm}\sigma \cdot \delta)v$ for all $v \in V$.

Abstractness of Semantics

Abstract semantics are an attempt to separate the important properties of a language (what computations can it express) from the unimportant (how exactly computations are represented).

The more terms are considered equal by a semantics, the more abstract it is.

A semantic function $[\![-]\!]_1$ is at least as abstract as $[\![-]\!]_0$ if $[\![-]\!]_1$ equates all terms that $[\![-]\!]_0$ does:

$$\forall c.\ [\![c]\!]_0 = [\![c']\!]_0 \Rightarrow [\![c]\!]_1 = [\![c']\!]_1$$

Soundness of Semantics

If there are other means of observing the result of a computation, a semantics may be incorrect if it equates too many terms.

$\mathcal{C}$ = the set of contexts: terms with a hole $\bullet$.
A term c can be placed in the hole of a context C, yielding term $C[c]$
(not subtitution — variable capture is possible)

Example: if $C = \textbf{newvar}\ \texttt{x:=1}\ \textbf{in}\ \bullet$,
then $C[\texttt{x:=x+1}] = \textbf{newvar}\ \texttt{x:=1}\ \textbf{in}\ \texttt{x:=x+1}$.

$\mathcal{O} = \mathit{terms} \rightarrow \mathit{outcomes}$: the set of observations.

A semantic function $[\![-]\!]$ is sound iff

$$\forall c, c'.\ [\![c]\!] = [\![c']\!] \Rightarrow \forall O \in \mathcal{O}.\ \forall C \in \mathcal{C}.\ O(C[c]) = O(C[c']).$$

Fully Abstract Semantics

Recap:

$[\![-]\!]_1$ is at least as abstract as $[\![-]\!]_0$

if $[\![-]\!]_1$ equates all terms that $[\![-]\!]_0$ does:

$$\forall c.\, [\![c]\!]_0 = [\![c']\!]_0 \Rightarrow [\![c]\!]_1 = [\![c']\!]_1$$

$[\![-]\!]$ is sound iff

$$\forall c, c'.\, [\![c]\!] = [\![c']\!] \Rightarrow \forall O \in \mathcal{O}.\, \forall C \in \mathcal{C}.\, O(C[c]) = O(C[c']).$$

A semantics is fully abstract iff

$$\forall c, c'.\, [\![c]\!] = [\![c']\!] \;\Leftrightarrow\; \forall O \in \mathcal{O}.\, \forall C \in \mathcal{C}.\, O(C[c]) = O(C[c'])$$

i.e. iff it is a “most abstract” sound semantics.

Full Abstractness of Semantics for SIL

Consider observations $O_{\sigma,v} \in \mathcal{O} \stackrel{\text{def}}{=} comm \to \mathbf{Z}_\perp$

observing the value of variable v after executing from state σ:

$$O_{\sigma,v}(c) = \left\{ \begin{matrix} \perp, & \text{if } [\![c]\!]_{comm}\sigma = \perp \\ [\![c]\!]_{comm}\sigma v, & \text{otherwise} \end{matrix} \right\} = ((-)\, v)_\perp([\![c]\!]_{comm}\sigma)$$

$[\![-]\!]_{comm}$ is fully abstract (with respect to observations $\mathcal{O}$):

- $[\![-]\!]_{comm}$ is sound: By compositionality, if $[\![c]\!]_{comm} = [\![c']\!]_{comm}$, then $[\![C[c]]\!]_{comm} = [\![C[c']]\!]_{comm}$ for any context C (induction); hence $O(C[c]) = O(C[c'])$ for any observation O.

- $[\![-]\!]_{comm}$ is most abstract: Consider the empty context $C = \bullet$; if $O_{\sigma,v}(c) = O_{\sigma,v}(c')$ for all $v \in var$, $\sigma \in \Sigma$, then $[\![c]\!] = [\![c']\!]$.

Observing Termination of Closed Commands

Suffices to observe if closed commands terminate:

If $[\![c]\!]_{comm} \neq [\![c']\!]_{comm}$, construct a context that distinguishes c and c'.

Suppose $[\![c]\!]_{comm}\sigma \neq [\![c']\!]_{comm}\sigma$ for some σ.

Let $\{v_i \mid i \in 1 \textbf{ to } n\} \stackrel{\text{def}}{=} FV(c) \cup FV(c')$,

and κ_i be constants such that $[\![\kappa_i]\!]_{intexp}\sigma' = \sigma v_i$.

Then by the Coincidence Theorem

$$[\![c]\!]_{comm}[\sigma' | v_i : \kappa_i{}^{i \in 1 \textbf{ to } n}] \neq [\![c']\!]_{comm}[\sigma' | v_i : \kappa_i{}^{i \in 1 \textbf{ to } n}]$$

for any state σ'.

Observing Termination Cont'd

Consider then the context C closing both c and c':

$$C \stackrel{\text{def}}{=} \textbf{newvar}\ v_1 := \kappa_1\ \textbf{in}\ \ldots \textbf{newvar}\ v_n := \kappa_n\ \textbf{in}\ \bullet$$

$C[c]$ and $C[c']$ may not both diverge from any initial state σ', since

$$[\![C[c]]\!]_{comm}\sigma' = ([-|v_i : \sigma' v_i\ {}^{i \in 1\ \text{to}\ n}])_{\perp\!\!\perp}\ [\![c]\!]_{comm}[\sigma'|v_i : \kappa_i\ {}^{i \in 1\ \text{to}\ n}]$$

and $C[c] = \perp = C[c']$ is only possible if

$$[\![c]\!]_{comm}[\sigma'|v_i : \kappa_i\ {}^{i \in 1\ \text{to}\ n}] = \perp = [\![c']\!]_{comm}[\sigma'|v_i : \kappa_i\ {}^{i \in 1\ \text{to}\ n}],$$

but by assumption and Coincidence the initial state $[\sigma'|v_i : \kappa_i\ {}^{i \in 1\ \text{to}\ n}]$ distinguishes c and c'.

Observing Termination Cont'd

If only one of $C[c]$ and $C[c']$ terminates,

then the restricted observations on C distinguishes between them.

If both $C[c]$ and $C[c']$ terminate,

then $[\![c]\!]_{comm}\sigma \neq \bot \neq [\![c']\!]_{comm}\sigma$,

hence $[\![c]\!]\sigma v = [\![\kappa]\!]\sigma' \neq [\![c']\!]\sigma v$ for some v.

Then for context

$$D \stackrel{\text{def}}{=} C[(\bullet\ ;\ \textbf{while}\ v\text{=}\kappa\ \textbf{do skip})]$$

we have $[\![D[c]]\!]_{comm}\sigma' = \bot \neq [\![D[c']]\!]_{comm}\sigma'$,

$\Rightarrow\ O_{\sigma,v}(D[c]) \neq O_{\sigma,v}(D[c'])$.