# Denotational Semantics

October 19, 2018

# Syntax of the Imp Language

$$
\begin{array}{lrcl}
\textit{(intexp)} & e & ::= & 0 \mid 1 \mid \dots \\
 & & \mid & x \\
 & & \mid & \text{-}e \mid e\text{+}e \mid e\text{-}e \mid \dots \\
 & & & \\
\textit{(boolexp)} & b & ::= & \textbf{true} \mid \textbf{false} \\
 & & \mid & e\text{=}e \mid e < e \mid e < e \mid \dots \\
 & & \mid & \neg b \mid b \wedge b \mid b \vee b \mid \dots \\
 & & & \qquad \textit{no quantified terms} \\
 & & & \\
\textit{(comm)} & c & ::= & x := e \\
 & & \mid & \textbf{skip} \\
 & & \mid & c;c \\
 & & \mid & \textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ c \\
 & & \mid & \textbf{while}\ b\ \textbf{do}\ c
\end{array}
$$

# Denotational Semantics

$$\begin{aligned}
[\![-]\!]_{intexp} &\in intexp \to \Sigma \to \mathbf{Z} & \qquad \Sigma &\stackrel{\text{def}}{=} var \to \mathbf{Z} \\
[\![-]\!]_{boolexp} &\in boolexp \to \Sigma \to \mathbf{B} \\
[\![-]\!]_{comm} &\in comm \to \Sigma \to \Sigma_{\perp} & \qquad \Sigma_{\perp} &\stackrel{\text{def}}{=} \Sigma \cup \{\perp\}
\end{aligned}$$

$$[\![x := e]\!]_{comm}\, \sigma = \sigma\{x \rightsquigarrow [\![e]\!]_{intexp}\, \sigma\}$$

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

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

$$[\![c;c']\!]_{comm}\, \sigma = \begin{cases} \perp & \text{if } [\![c]\!]_{comm}\, \sigma = \perp \\ ([\![c']\!]_{comm} \circ [\![c]\!]_{comm})\, \sigma & \text{otherwise} \end{cases}$$

$$
\begin{aligned}
[\![-]\!]_{intexp} &\in intexp \to \Sigma \to \mathbf{Z} & \qquad \Sigma &\stackrel{\text{def}}{=} var \to \mathbf{Z} \\
[\![-]\!]_{boolexp} &\in boolexp \to \Sigma \to \mathbf{B} \\
[\![-]\!]_{comm} &\in comm \to \Sigma \to \Sigma_\perp & \qquad \Sigma_\perp &\stackrel{\text{def}}{=} \Sigma \cup \{\perp\}
\end{aligned}
$$

$$[\![x := e]\!]_{comm}\,\sigma = \sigma\{x \leadsto [\![e]\!]_{intexp}\,\sigma\}$$

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

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

$$
[\![c;c']\!]_{comm}\,\sigma \quad = \begin{cases} \perp & \text{if } [\![c]\!]_{comm}\,\sigma = \perp \\ ([\![c']\!]_{comm} \circ [\![c]\!]_{comm})\,\sigma & \text{otherwise} \end{cases}
$$

$$
\begin{aligned}
[\![-]\!]_{intexp} &\in intexp \to \Sigma \to \mathbf{Z} & \qquad \Sigma &\stackrel{\text{def}}{=} var \to \mathbf{Z} \\
[\![-]\!]_{boolexp} &\in boolexp \to \Sigma \to \mathbf{B} & & \\
[\![-]\!]_{comm} &\in comm \to \Sigma \to \Sigma_\perp & \qquad \Sigma_\perp &\stackrel{\text{def}}{=} \Sigma \cup \{\perp\}
\end{aligned}
$$

$$[\![x := e]\!]_{comm}\,\sigma = \sigma\{x \leadsto [\![e]\!]_{intexp}\,\sigma\}$$

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

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

$$[\![c;c']\!]_{comm}\,\sigma \quad = \begin{cases} \perp & \text{if } [\![c]\!]_{comm}\,\sigma = \perp \\ ([\![c']\!]_{comm} \circ [\![c]\!]_{comm})\,\sigma & \text{otherwise} \end{cases}$$

# Denotational Semantics

$$
\begin{array}{lll}
[\![-]\!]_{intexp} & \in intexp \to \Sigma \to \mathbf{Z} & \qquad \Sigma \stackrel{\text{def}}{=} var \to \mathbf{Z} \\
[\![-]\!]_{boolexp} & \in boolexp \to \Sigma \to \mathbf{B} & \\
[\![-]\!]_{comm} & \in comm \to \Sigma \to \Sigma_{\perp} & \qquad \Sigma_{\perp} \stackrel{\text{def}}{=} \Sigma \cup \{\perp\}
\end{array}
$$

$$[\![x := e]\!]_{comm}\,\sigma = \sigma\{x \rightsquigarrow [\![e]\!]_{intexp}\,\sigma\}$$

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

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

$$[\![c;c']\!]_{comm}\,\sigma \quad = \begin{cases} \perp & \text{if } [\![c]\!]_{comm}\,\sigma = \perp \\ ([\![c']\!]_{comm} \circ [\![c]\!]_{comm})\,\sigma & \text{otherwise} \end{cases}$$

# Semantics of Sequential Composition

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

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

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

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

# Semantics of Conditionals

$$[\![\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2]\!]_{comm}\,\sigma = \begin{cases} [\![c_1]\!]_{comm}\,\sigma & \text{if } [\![b]\!]_{boolexp}\,\sigma = \textbf{true} \\ [\![c_2]\!]_{comm}\,\sigma & \text{if } [\![b]\!]_{boolexp}\,\sigma = \textbf{false} \end{cases}$$

Examples:

$$\begin{aligned}
&[\![\textbf{if } x < 0 \textbf{ then } x = -x \textbf{ else skip}]\!]_{comm}\,\{(x,-3)\} \\
&\quad = [\![x = -x]\!]_{comm}\,\{(x,-3)\} \qquad \text{since } [\![x < 0]\!]_{boolexp}\,\{(x,-3)\} = \textbf{true} \\
&\quad = \{(x,-3)\}\{x \rightsquigarrow [\![-x]\!]_{intexp}\,\{(x,-3)\}\} \\
&\quad = \{(x,3)\}
\end{aligned}$$

$$\begin{aligned}
&[\![\textbf{if } x < 0 \textbf{ then } x = -x \textbf{ else skip}]\!]_{comm}\,\{(x,5)\} \\
&\quad = [\![\textbf{skip}]\!]_{comm}\,\{(x,5)\} \qquad \text{since } [\![x < 0]\!]_{boolexp}\,\{(x,5)\} = \textbf{false} \\
&\quad = \{(x,5)\}
\end{aligned}$$

# Semantics of Loops

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

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

That is,

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

However, the semantic function is *not syntax directed*, as $[\![\textbf{while } b \textbf{ do } c]\!]_{comm}$ itself shows as a sub-term on the right side of the equation.

# Semantics of Loops

Actually we can view $[\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{comm}$ as a solution for this equation:

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

That is, $[\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{comm}$ is a fixed-point of

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

However, not every $F \in (\Sigma \to \Sigma_{\perp}) \to (\Sigma \to \Sigma_{\perp})$ has a fixed-point, and some may have more than one.
Example: for any $\sigma'$, $\lambda\sigma.\,\sigma'$ is a solution for $[\![\textbf{while true do}\ x := x + 1]\!]_{comm}$.

# Semantics of Loops

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

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

However, not every $F \in (\Sigma \to \Sigma_\perp) \to (\Sigma \to \Sigma_\perp)$ has a fixed-point, and some may have more than one.

We need to lay some structures over the set $\Sigma \to \Sigma_\perp$, to ensure that $F$ has at least one fixed-point.

# Partially Ordered Sets

| | | | |
|---|---|---|---|
| A relation $\rho$ is | reflexive on $S$ | iff | $\forall x \in S. x\, \rho\, x$ |
| | transitive | iff | $x\, \rho\, y \wedge y\, \rho\, z \Rightarrow x\, \rho\, z$ |
| | antisymmetric | iff | $x\, \rho\, y \wedge y\, \rho\, x \Rightarrow x = y$ |
| | symmetric | iff | $x\, \rho\, y \Rightarrow y\, \rho\, x$ |

| | |
|---|---|
| $\sqsubseteq$ is a *preorder* on $S$ | iff $\sqsubseteq$ is reflexive on $S$ and tansitive |
| $\sqsubseteq$ is a *partial order* on $S$ | iff $\sqsubseteq$ is a preorder on $S$ and antisymmetric |
| A poset $S$ | $S$ with a partial order $\sqsubseteq$ on $S$ |
| A discretely ordered $S$ | $S$ with $\mathrm{Id}_S$ as a partial order on $S$ |

$f$ is monotone from $S$ to $T$ iff $f \in S \to T$
and $\forall x, y \in S. x \sqsubseteq y \Rightarrow f\, x \sqsubseteq' f\, y$

$y$ is upper bound of $X$ $\quad \forall x \in X. x \sqsubseteq y$
where $y \in S$ and $X \subseteq S$

# Partially Ordered Sets

| | | | |
|---|---|---|---|
| A relation $\rho$ is | reflexive on $S$ | iff | $\forall x \in S. x \,\rho\, x$ |
| | transitive | iff | $x \,\rho\, y \wedge y \,\rho\, z \Rightarrow x \,\rho\, z$ |
| | antisymmetric | iff | $x \,\rho\, y \wedge y \,\rho\, x \Rightarrow x = y$ |
| | symmetric | iff | $x \,\rho\, y \Rightarrow y \,\rho\, x$ |

| | |
|---|---|
| $\sqsubseteq$ is a *preorder* on $S$ | iff $\sqsubseteq$ is reflexive on $S$ and tansitive |
| $\sqsubseteq$ is a *partial order* on $S$ | iff $\sqsubseteq$ is a preorder on $S$ and antisymmetric |
| A poset $S$ | $S$ with a partial order $\sqsubseteq$ on $S$ |
| A discretely ordered $S$ | $S$ with $\mathrm{Id}_S$ as a partial order on $S$ |
| $f$ is monotone from $S$ to $T$ | iff $f \in S \to T$ and $\forall x, y \in S. x \sqsubseteq y \Rightarrow f\,x \sqsubseteq' f\,y$ |
| $y$ is upper bound of $X$ | $\forall x \in X. x \sqsubseteq y$ where $y \in S$ and $X \subseteq S$ |

# Partially Ordered Sets

| | | | |
|---|---|---|---|
| A relation $\rho$ is | reflexive on $S$ | iff | $\forall x \in S. x \,\rho\, x$ |
| | transitive | iff | $x \,\rho\, y \wedge y \,\rho\, z \Rightarrow x \,\rho\, z$ |
| | antisymmetric | iff | $x \,\rho\, y \wedge y \,\rho\, x \Rightarrow x = y$ |
| | symmetric | iff | $x \,\rho\, y \Rightarrow y \,\rho\, x$ |

| | |
|---|---|
| $\sqsubseteq$ is a *preorder* on $S$ | iff $\sqsubseteq$ is reflexive on $S$ and tansitive |
| $\sqsubseteq$ is a *partial order* on $S$ | iff $\sqsubseteq$ is a preorder on $S$ and antisymmetric |
| A poset $S$ | $S$ with a partial order $\sqsubseteq$ on $S$ |
| A discretely ordered $S$ | $S$ with $\mathrm{Id}_S$ as a partial order on $S$ |
| $f$ is monotone from $S$ to $T$ | iff $f \in S \to T$ and $\forall x, y \in S. x \sqsubseteq y \Rightarrow f\,x \sqsubseteq' f\,y$ |
| $y$ is upper bound of $X$ | $\forall x \in X. x \sqsubseteq y$ where $y \in S$ and $X \subseteq S$ |

# Partially Ordered Sets

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

| | |
|---|---|
| $\sqsubseteq$ is a *preorder* on $S$ | iff $\sqsubseteq$ is reflexive on $S$ and tansitive |
| $\sqsubseteq$ is a *partial order* on $S$ | iff $\sqsubseteq$ is a preorder on $S$ and antisymmetric |
| A poset $S$ | $S$ with a partial order $\sqsubseteq$ on $S$ |
| A discretely ordered $S$ | $S$ with $\mathrm{Id}_S$ as a partial order on $S$ |
| $f$ is monotone from $S$ to $T$ | iff $f \in S \to T$ and $\forall x, y \in S. x \sqsubseteq y \Rightarrow f\,x \sqsubseteq' f\,y$ |
| $y$ is upper bound of $X$ | $\forall x \in X. x \sqsubseteq y$ where $y \in S$ and $X \subseteq S$ |

# Partially Ordered Sets

| | | | |
|---|---|---|---|
| A relation $\rho$ is | reflexive on $S$ | iff | $\forall x \in S. x \, \rho \, x$ |
| | transitive | iff | $x \, \rho \, y \wedge y \, \rho \, z \Rightarrow x \, \rho \, z$ |
| | antisymmetric | iff | $x \, \rho \, y \wedge y \, \rho \, x \Rightarrow x = y$ |
| | symmetric | iff | $x \, \rho \, y \Rightarrow y \, \rho \, x$ |

| | |
|---|---|
| $\sqsubseteq$ is a *preorder* on $S$ | iff $\sqsubseteq$ is reflexive on $S$ and tansitive |
| $\sqsubseteq$ is a *partial order* on $S$ | iff $\sqsubseteq$ is a preorder on $S$ and antisymmetric |
| A poset $S$ | $S$ with a partial order $\sqsubseteq$ on $S$ |
| A discretely ordered $S$ | $S$ with $\mathrm{Id}_S$ as a partial order on $S$ |
| $f$ is monotone from $S$ to $T$ | iff $f \in S \to T$ and $\forall x, y \in S. x \sqsubseteq y \Rightarrow f\,x \sqsubseteq' f\,y$ |
| $y$ is upper bound of $X$ | $\forall x \in X. x \sqsubseteq y$ where $y \in S$ and $X \subseteq S$ |

# Partially Ordered Sets

| | | | |
|---|---|---|---|
| A relation $\rho$ is | reflexive on $S$ | iff | $\forall x \in S. x \,\rho\, x$ |
| | transitive | iff | $x \,\rho\, y \wedge y \,\rho\, z \Rightarrow x \,\rho\, z$ |
| | antisymmetric | iff | $x \,\rho\, y \wedge y \,\rho\, x \Rightarrow x = y$ |
| | symmetric | iff | $x \,\rho\, y \Rightarrow y \,\rho\, x$ |

| | |
|---|---|
| $\sqsubseteq$ is a *preorder* on $S$ | iff $\sqsubseteq$ is reflexive on $S$ and tansitive |
| $\sqsubseteq$ is a *partial order* on $S$ | iff $\sqsubseteq$ is a preorder on $S$ and antisymmetric |
| A poset $S$ | $S$ with a partial order $\sqsubseteq$ on $S$ |
| A discretely ordered $S$ | $S$ with $\mathrm{Id}_S$ as a partial order on $S$ |

$f$ is monotone from $S$ to $T$ iff $f \in S \to T$
and $\forall x, y \in S. x \sqsubseteq y \Rightarrow f\,x \sqsubseteq' f\,y$

$y$ is upper bound of $X$ $\quad \forall x \in X. x \sqsubseteq y$
where $y \in S$ and $X \subseteq S$

# Partially Ordered Sets

| | | | |
|---|---|---|---|
| A relation $\rho$ is | reflexive on $S$ | iff | $\forall x \in S. x\,\rho\,x$ |
| | transitive | iff | $x\,\rho\,y \wedge y\,\rho\,z \Rightarrow x\,\rho\,z$ |
| | antisymmetric | iff | $x\,\rho\,y \wedge y\,\rho\,x \Rightarrow x = y$ |
| | symmetric | iff | $x\,\rho\,y \Rightarrow y\,\rho\,x$ |

| | |
|---|---|
| $\sqsubseteq$ is a *preorder* on $S$ | iff $\sqsubseteq$ is reflexive on $S$ and tansitive |
| $\sqsubseteq$ is a *partial order* on $S$ | iff $\sqsubseteq$ is a preorder on $S$ and antisymmetric |
| A poset $S$ | $S$ with a partial order $\sqsubseteq$ on $S$ |
| A discretely ordered $S$ | $S$ with $\mathrm{Id}_S$ as a partial order on $S$ |
| $f$ is monotone from $S$ to $T$ | iff $f \in S \to T$ and $\forall x, y \in S. x \sqsubseteq y \Rightarrow f\,x \sqsubseteq' f\,y$ |
| $y$ is upper bound of $X$ | $\forall x \in X. x \sqsubseteq y$ where $y \in S$ and $X \subseteq S$ |

# Least Upper Bounds

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

If $S$ is a poset and $X \subseteq S$, there is at most one lub of $X$ ($\sqcup X$) .

$\sqcup \emptyset = \bot$, the least element of $S$ (if exists).

Let $\mathcal{X} \subseteq \mathcal{P}(S)$ such that $\sqcup X$ exists for all $X \in \mathcal{X}$. Then

$$\sqcup\{\sqcup X \mid X \in \mathcal{X}\} = \sqcup\left(\bigcup \mathcal{X}\right)$$

if either of these lub exists.

# Domains

A *chain* $C$ is a countably infinite non-decreasing sequence $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$.
We may also use $C$ to represent the set of elements on the chain.

The *limit* of a chain $C$ is the lub of all its elements when it exists.

A chain $C$ is interesting if $(\sqcup C) \notin C$.
(Chains with finitely many distinct elements are uninteresting.)

A poset $D$ is a *predomain* (or *complete partial order* – *cpo*) if every chain of elements in $D$ has a limit in $D$.

A predomain $D$ is a *domain* (or *pointed cpo*) if $D$ has a least element $\perp$.

# Lifting

$D_\perp$ is a *lifting* of the predomain $D$ if:

- $\perp \notin D$, and
- $x \sqsubseteq_{D_\perp} y$ iff either $x = \perp$ or $x \sqsubseteq_D y$

$D_\perp$ is a domain.

Any set $S$ can be viewed as a predomain with *discrete partial order* $\sqsubseteq \stackrel{\text{def}}{=} \mathrm{Id}_S$.

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

# Continuous Functions

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

$$f\,(\sqcup C) = \sqcup'\{f\,x_i \mid x_i \in C\} \text{ for every chain } C \text{ in } D$$

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

There are non-continuous monotone functions:
Suppose $C = x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq \ldots$ is an interesting chain in $D$ with a limit $x$, and $D' = \{\bot, \top\}$ such that $\bot \sqsubseteq' \top$. Then

$$f = \lambda y. \begin{cases} \bot & \text{if } y \in C \\ \top & \text{if } y = x \end{cases}$$

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

# Continuous Functions

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

$$f\,(\sqcup\, C) = \sqcup'\{f\, x_i \mid x_i \in C\} \text{ for every chain } C \text{ in } D$$

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

There are non-continuous monotone functions:
Suppose $C = x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq \ldots$ is an interesting chain in $D$ with a limit $x$, and $D' = \{\bot, \top\}$ such that $\bot \sqsubseteq' \top$. Then

$$f = \lambda y. \begin{cases} \bot & \text{if } y \in C \\ \top & \text{if } y = x \end{cases}$$

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

# Monotone vs Continuous Functions

A monotone function $f \in D \to D'$ is continuous iff for all *interesting* chains $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$, we have $f(\bigsqcup_{i=0}^{\infty} x_i) \sqsubseteq \bigsqcup_{i=0}^{\prime\infty} (f\, x_i)$.

**Proof.**
The right-direction implication is obvious following the definition of continuous functions. We prove the left-direction implication.

- for uninteresting chains $x_0 \sqsubseteq x_1 \sqsubseteq \ldots \sqsubseteq x_n$, $x_n = \bigsqcup_{i=0}^{n} x_i$. Since $f$ is monotone, $f(\bigsqcup_{i=0}^{\infty} x_i) = f\, x_n = \bigsqcup_{i=0}^{\prime\infty} (f\, x_i)$.
- for interesting chains, suppose $x = \bigsqcup_{i=0}^{\infty} x_i$. We know $f\, x_i \sqsubseteq f\, x$ holds for all $i \in \mathbf{N}$, following the monotonicity of $f$. Therefore $\bigsqcup_{i=0}^{\prime\infty} (f\, x_i) \sqsubseteq f\, x = f(\bigsqcup_{i=0}^{\infty} x_i)$. Given assumption $f(\bigsqcup_{i=0}^{\infty} x_i) \sqsubseteq \bigsqcup_{i=0}^{\prime\infty} (f\, x_i)$, we know $f(\bigsqcup_{i=0}^{\infty} x_i) = \bigsqcup_{i=0}^{\prime\infty} (f\, x_i)$.

# The (Pre)domain of Continuous Functions

*pointwise ordering* of functions in $P \to P'$, where $P'$ is a predomain:

$$f \sqsubseteq_{\to} g \stackrel{\text{def}}{=} \forall x \in P.\, f\,x \sqsubseteq_{P'} g\,x$$

**Proposition:**
If both $P$ and $P'$ are predomains, then the set $[P \to P']$ of *continuous* functions in $P \to P'$ with partial order $\sqsubseteq_{\to}$ is a predomain, such that for any chain $f_0 \sqsubseteq_{\to} f_1 \sqsubseteq_{\to} \ldots$, we have

$$\bigsqcup_i f_i = \lambda x \in P.\, \bigsqcup'_i (f_i\,x)\,.$$

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

# The (Pre)domain of Continuous Functions: Proof

To prove $[P \to P']$ is a predomain, we need to prove

1. Every chain $f_0 \sqsubseteq_\to f_1 \sqsubseteq_\to \dots$ in $[P \to P']$ has a limit $f$; and
2. $f$ is also in $[P \to P']$.

**Proof:**
Let $f = \lambda x \in P. \bigsqcup'_i (f_i\, x)$. $f$ is well defined, i.e. $\bigsqcup'_i (f_i\, x)$ exists, because $P'$ is a predomain, and $f_0\, x \sqsubseteq_{P'} f_1\, x \sqsubseteq_{P'} \dots$ since $f_0 \sqsubseteq_\to f_1 \sqsubseteq_\to \dots$.

Then we prove

1.1 It is an upper bound of $f_0 \sqsubseteq_\to f_1 \sqsubseteq_\to \dots$. in $[P \to P']$ has a limit $f$;

1.2 It is the least upper bound;

2 It is continuous, thus it is also in $[P \to P']$.

Proof of **1.1:** $f$ is an upper bound.
$f_i \sqsubseteq_\rightarrow f$ because $\forall x \in P. f_i\, x \sqsubseteq_{P'} (\bigsqcup'_i (f_i\, x)) = f\, x$. Therefore $f$ is an upper bound of $f_0 \sqsubseteq_\rightarrow f_1 \sqsubseteq_\rightarrow \ldots$.

Proof of **1.2:** $f$ is the least upper bound.
If $g$ is another upper bound, then $\forall x \in P. f_i\, x \sqsubseteq_{P'} g\, x$ holds for all $i$. Therefore $\forall x \in P. f_i\, x \sqsubseteq_{P'} \bigsqcup'_i (f_i\, x) = f\, x \sqsubseteq_{P'} g\, x$, i.e. $f \sqsubseteq_\rightarrow g$.

Proof of **2:** $f$ is continuous, that is, for any chain $x_0 \sqsubseteq x_1 \ldots$ in $P$, $f(\bigsqcup_j x_j) = \bigsqcup'_j (f\, x_j)$.
We know
$f(\bigsqcup_j x_j) =^1 \bigsqcup'_i (f_i(\bigsqcup_j x_j)) =^2 \bigsqcup'_i (\bigsqcup'_j (f_i\, x_j)) =^3 \bigsqcup'_j (\bigsqcup'_i (f_i\, x_j)) =^4 \bigsqcup'_j (f\, x_j)$

1. Definition of $f$
2. $f_i$ is continuous
3. property of lub
4. Definition of $f$

# Examples: Continuous Functions

For predomains $P$, $P'$ and $P''$,

- If $f \in P \to P'$ is a constant function, then $f \in [P \to P']$.
- $\mathrm{Id}_P \ \in \ [P \to P]$.
- If $f \in [P \to P']$ and $g \in [P' \to P'']$, then $g \circ f \in [P \to P'']$.
- If $f \in [P \to P']$, then $(- \circ f) \in [[P' \to P''] \to [P \to P'']]$.
- If $f \in [P \to P']$, then $(- \circ f) \in [[P' \to P''] \to [P \to P'']]$.
- If $g \in [P' \to P'']$, then $(g \circ -) \in [[P \to P'] \to [P \to P'']]$.

# Strict Functions and Lifting

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

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

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

is the *lifting* of $f$ to $P_{\perp} \to P'_{\perp'}$. If P' is a domain, then the strict function

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

is the *source lifting* of $f$ to $P_{\perp} \to P'$.

If $f$ is continuous, so are $f_{\perp}$ and $f_{\perp\!\!\!\perp}$.
$(-)_{\perp}$ and $(-)_{\perp\!\!\!\perp}$ are also continuous.

# Least Fixed-Point

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

**Theorem [Least Fixed-Point of a Continuous Function, a.k.a. Kleene Fixed-Point Theorem]**

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 $f$. (Note $f^0 = \mathrm{Id}_D$ and $f^{n+1} = f \circ (f^n)$)

**Proof.**

$x$ is well-defined because $\bot \sqsubseteq f\,\bot \sqsubseteq f^2\,\bot \sqsubseteq \ldots$ is a chain. (why?)

$x$ is a fixed-point because

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

For any fixed-point $y$ of $f$, $\bot \sqsubseteq y \Rightarrow f\,\bot \sqsubseteq f\,y = y$.
By induction, we have $\forall i \in \mathbf{N}.\ f^i\,\bot \sqsubseteq y$. So $y$ is an upper bound of the chain $\bot \sqsubseteq f\,\bot \sqsubseteq \ldots$ Since $x$ is a lub, so $x \sqsubseteq y$.

# The Least Fixed-Point Operator

Let

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

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

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

# Get Back to Semantics of Loops

Recall our first attempt:

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

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

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

We pick the least fixed-point:

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

# Semantics of Loops: Intuition

$$\begin{array}{lll} w_0 & \stackrel{\text{def}}{=} & \textbf{while true do skip} \\ w_{i+1} & \stackrel{\text{def}}{=} & \textbf{if } b \textbf{ then } (c\,;w_i) \textbf{ else skip} \end{array} \qquad \begin{array}{ll} [\![w_0]\!]_{comm} & = \lambda\sigma.\,\bot \\ [\![w_{i+1}]\!]_{comm} & = F\,[\![w_i]\!]_{comm} \end{array}$$

Suppose the loop **while** $b$ **do** $c$ at state $\sigma$ evaluates the condition ($b$) $n$ times before it terminates. Then it behaves like $w_i$ for all $i \geq n$.

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

If the loop never terminates:

$$[\![\textbf{while } b \textbf{ do } c]\!]_{comm}\,\sigma = \bot = [\![w_i]\!]_{comm}\,\sigma \quad \text{(for all } i\text{)}$$

Therefore

$$\forall\sigma \in \Sigma.\ [\![\textbf{while } b \textbf{ do } c]\!]_{comm}\,\sigma = \bigsqcup_{i=0}^{\infty}([\![w_i]\!]_{comm}\,\sigma)$$

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

# Variable Declarations

Syntax

$$c \ ::= \ \textbf{newvar}\ x := e\ \textbf{in}\ c$$

Semantics:

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

($\textbf{newvar}\ x := e\ \textbf{in}\ c$) binds $x$ in $c$, but not in $e$:

$$fv(\textbf{newvar}\ x := e\ \textbf{in}\ c) = (fv(c) - \{x\}) \cup fv(e)$$

# Free Variables and Assigned Variables

Free variables:

$$
\begin{aligned}
fv_{comm}(x := e) &= \{x\} \cup fv_{intexp}(e) \\
fv_{comm}(\mathbf{skip}) &= \emptyset \\
fv_{comm}(c\,;c') &= fv_{comm}(c) \cup fv_{comm}(c') \\
fv_{comm}(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1) &= fv_{boolexp}(b) \cup fv_{comm}(c_0) \cup fv_{comm}(c_1) \\
fv_{comm}(\mathbf{while}\ b\ \mathbf{do}\ c) &= fv_{boolexp}(b) \cup fv_{comm}(c) \\
fv_{comm}(\mathbf{newvar}\ x := e\ \mathbf{in}\ c) &= (fv_{comm}(c) - \{x\}) \cup fv_{intexp}(e)
\end{aligned}
$$

Assigned variables:

$$
\begin{aligned}
fa(x := e) &= \{x\} \\
fa(\mathbf{skip}) &= \emptyset \\
fa(c\,;c') &= fa(c) \cup fa(c') \\
fa(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1) &= fa(c_0) \cup fa(c_1) \\
fa(\mathbf{while}\ b\ \mathbf{do}\ c) &= fa(c) \\
fa(\mathbf{newvar}\ x := e\ \mathbf{in}\ c) &= fa(c) - \{x\}
\end{aligned}
$$

# Free Variables and Assigned Variables

Free variables:

$$\begin{aligned}
fv_{comm}(x := e) &= \{x\} \cup fv_{intexp}(e) \\
fv_{comm}(\mathbf{skip}) &= \emptyset \\
fv_{comm}(c\,;c') &= fv_{comm}(c) \cup fv_{comm}(c') \\
fv_{comm}(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1) &= fv_{boolexp}(b) \cup fv_{comm}(c_0) \cup fv_{comm}(c_1) \\
fv_{comm}(\mathbf{while}\ b\ \mathbf{do}\ c) &= fv_{boolexp}(b) \cup fv_{comm}(c) \\
fv_{comm}(\mathbf{newvar}\ x := e\ \mathbf{in}\ c) &= (fv_{comm}(c) - \{x\}) \cup fv_{intexp}(e)
\end{aligned}$$

Assigned variables:

$$\begin{aligned}
fa(x := e) &= \{x\} \\
fa(\mathbf{skip}) &= \emptyset \\
fa(c\,;c') &= fa(c) \cup fa(c') \\
fa(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1) &= fa(c_0) \cup fa(c_1) \\
fa(\mathbf{while}\ b\ \mathbf{do}\ c) &= fa(c) \\
fa(\mathbf{newvar}\ x := e\ \mathbf{in}\ c) &= fa(c) - \{x\}
\end{aligned}$$

# Coincidence Theorem for Commands

The meaning of a command now depends not only on the mapping of its free variables:

$[\![c]\!]_{comm}\,\sigma\,x = \sigma\,x \quad$ if $[\![c]\!]_{comm}\,\sigma \neq \perp$ and $x \notin fv(c)$

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

**Coincidence Theorem:**

- If $\sigma\,x = \sigma'\,x$ for all $x \in fv(c)$, then $[\![c]\!]_{comm}\,\sigma = \perp = [\![c]\!]_{comm}\,\sigma'$, or $\forall x \in fv(c).\, [\![c]\!]_{comm}\,\sigma\,x = [\![c]\!]_{comm}\,\sigma'\,x$.
- If $[\![c]\!]_{comm}\,\sigma \neq \perp$, then $[\![c]\!]_{comm}\,\sigma\,x = \sigma\,x$ for all $x \notin fv(c)$.

**Renaming Theorem:**

If $x' \notin fv(c) - \{x\}$, then

$[\![\textbf{newvar}\ x := e\ \textbf{in}\ c]\!]_{comm}\,\sigma = [\![\textbf{newvar}\ x' := e\ \textbf{in}\ c[x'/x]]\!]_{comm}\,\sigma$

# Coincidence Theorem for Commands

The meaning of a command now depends not only on the mapping of its free variables:

$[\![c]\!]_{comm}\,\sigma\,x = \sigma\,x \quad \text{if } [\![c]\!]_{comm}\,\sigma \neq \perp \text{ and } x \notin fv(c)$

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

**Coincidence Theorem:**

- If $\sigma\,x = \sigma'\,x$ for all $x \in fv(c)$, then $[\![c]\!]_{comm}\,\sigma = \perp = [\![c]\!]_{comm}\,\sigma'$, or $\forall x \in fv(c).\ [\![c]\!]_{comm}\,\sigma\,x = [\![c]\!]_{comm}\,\sigma'\,x$.
- If $[\![c]\!]_{comm}\,\sigma \neq \perp$, then $[\![c]\!]_{comm}\,\sigma\,x = \sigma\,x$ for all $x \notin fv(c)$.

**Renaming Theorem:**

If $x' \notin fv(c) - \{x\}$, then

$[\![\textbf{newvar}\ x := e\ \textbf{in}\ c]\!]_{comm}\,\sigma = [\![\textbf{newvar}\ x' := e\ \textbf{in}\ c[x'/x]]\!]_{comm}\,\sigma$

# 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

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

# 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.

A *context* $C$ is a command with a *hole* $\bullet$.
A command $c$ can be *placed in the hole* of $C$, yielding $C[c]$ (not substitution — name capture is allowed).

Example:
If $C = \textbf{newvar}\ x := 1\ \textbf{in}\ \bullet\ \textbf{;}\ y := x;$, then
$C[x := x + 1] = \textbf{newvar}\ x := 1\ \textbf{in}\ x := x + 1\ \textbf{;}\ y := x;$

Let $O$ be an observation, and $\mathcal{O}$ be a set of observations, i.e.
$O \in \mathcal{O} \subseteq \mathit{comm} \rightarrow \mathit{outcomes}$.
Also we use $\mathcal{C}$ for the set of all contexts.

A semantic function $[\![-]\!]$ is *sound (with respect to $\mathcal{O}$)* iff

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

# Soundness and Full Abstractness

A semantic function $[\![-]\!]$ is *sound (with respect to $\mathcal{O}$)* iff

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

A semantic function $[\![-]\!]$ is *fully abstract (with respect to $\mathcal{O}$)* 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. $[\![-]\!]$ is the "most abstract" sound semantics.

$\Rightarrow$ (soundness): $[\![-]\!]$ cannot be too abstract;
$\Leftarrow$: $[\![-]\!]$ cannot be too concrete either

**Proposition:**
If $[\![-]\!]_0$ and $[\![-]\!]_1$ are both fully abstract semantics with respect to $\mathcal{O}$, then $[\![-]\!]_0 = [\![-]\!]_1$, i.e. $\forall c.\, [\![c]\!]_0 = [\![c]\!]_1$.

# Full Abstractness of Semantics for Imp

Let $\quad O_{\sigma,x} \stackrel{\text{def}}{=} \lambda c. \begin{cases} \perp & \text{if } [\![c]\!]_{comm}\,\sigma = \perp \\ \sigma'\,x & \text{if } [\![c]\!]_{comm}\,\sigma = \sigma' \end{cases}$

So $O_{\sigma,x}$ is an observation, and $O_{\sigma,x} \in comm \to \mathbf{Z}_\perp$

Let $\mathcal{O}$ be the set of all such observations, i.e.

$$\mathcal{O} = \{O_{\sigma,x} \mid \sigma \in \Sigma \text{ and } x \in var\}$$

**Proposition:** $[\![-]\!]_{comm}$ is fully abstract with respect to $\mathcal{O}$.

- $[\![-]\!]_{comm}$ is sound: By compositionality, if $[\![c]\!]_{comm} = [\![c']\!]_{comm}$, then for any context $C$, $[\![C[c]]\!]_{comm} = [\![C[c']]\!]_{comm}$ (induction). So $O_{\sigma,x}(C[c]) = O_{\sigma,x}(C[c'])$ for any observation $O_{\sigma,x}$.
- $[\![-]\!]_{comm}$ is most abstract: consider the empty context $C = \bullet$. If $O_{\sigma,x}(c) = O_{\sigma,x}(c')$ holds for all $x \in var$ and $\sigma \in \Sigma$, we know $[\![c]\!]_{comm} = [\![c']\!]_{comm}$.

Suppose we only care about termination of *closed* programs.

Let $O' \stackrel{\text{def}}{=} \lambda c. \begin{cases} \textbf{false} & \text{if } \exists\sigma. [\![c]\!]_{comm}\,\sigma = \bot \\ \textbf{true} & \text{otherwise} \end{cases}$

Note that if $c$ is closed, whether $[\![c]\!]_{comm}\,\sigma$ terminates or not is independent with $\sigma$.

$O'$ is an observation, with type $comm \to \mathbf{B}$

Let $\mathcal{O}' = \{O'\}$. $[\![-]\!]_{comm}$ is fully abstract with respect to $\mathcal{O}'$ if we only consider closed environments, i.e.

$$\begin{aligned} &\forall c, c'. [\![c]\!]_{comm} = [\![c']\!]_{comm} \Leftrightarrow \\ &\qquad \forall O \in \mathcal{O}'. \forall C \in \mathcal{C}. fv(C[c]) \cup fv(C[c']) = \emptyset \\ &\qquad\qquad \Rightarrow O(C[c]) = O(C[c']) \end{aligned}$$

# Observing Termination of Closed Commands (cont'd)

The proof of soundness ($\Rightarrow$) is the same as before. We prove the semantics is most abstract with respect to $\mathcal{O}'$ ($\Leftarrow$).

Suppose $[\![c]\!]_{comm} \neq [\![c']\!]_{comm}$, we could construct a context $C$ such that $O'(C[c]) \neq O'(C[c])$.

Suppose $[\![c]\!]_{comm}\,\sigma \neq [\![c']\!]_{comm}\,\sigma$ for some $\sigma$. Let $\{x_i \mid i \in [1, n]\} \stackrel{\text{def}}{=} fv(c) \cup fv(c')$, and $k_i$ be constants such that $k_i = \sigma\, x_i$.

Then by the Coincidence Theorem, for any $\sigma'$ and $\sigma''$,

$$\begin{aligned}[\![c]\!]_{comm}\ &(\sigma'\{x_1 \rightsquigarrow k_1, \ldots, x_n \rightsquigarrow k_n\}) \\ &\neq [\![c']\!]_{comm}\ (\sigma''\{x_1 \rightsquigarrow k_1, \ldots, x_n \rightsquigarrow k_n\})\end{aligned}$$

# Observing Termination of Closed Commands (cont'd)

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

$$C \stackrel{\text{def}}{=} \textbf{newvar}\ x_1 = k_1\ \textbf{in}\ \ldots\ \textbf{newvar}\ x_n = k_n\ \textbf{in}\ \bullet$$

First we show, for any $\sigma'$ and $\sigma''$, it is impossible to have $[\![C[c]]\!]_{comm}\,\sigma' = [\![C[c']]\!]_{comm}\,\sigma'' = \perp$.

This is because

$$[\![C[c]]\!]_{comm}\,\sigma' = f_{\perp\!\!\perp}\ ([\![c]\!]_{comm}\ (\sigma'\{x_1 \rightsquigarrow k_1, \ldots, x_n \rightsquigarrow k_n\}))$$
$$\text{where } f = (-)\{x_1 \rightsquigarrow \sigma'\,x_1, \ldots, x_n \rightsquigarrow \sigma'\,x_n\}$$

So $[\![C[c]]\!]_{comm}\,\sigma' = [\![C[c']]\!]_{comm}\,\sigma'' = \perp$ only if
$[\![c]\!]_{comm}\ (\sigma'\{x_1 \rightsquigarrow k_1, \ldots, x_n \rightsquigarrow k_n\}) =$
$[\![c']\!]_{comm}\ (\sigma''\{x_1 \rightsquigarrow k_1, \ldots, x_n \rightsquigarrow k_n\})$.
This cannot be true, as we show in the previous slide.

# Observing Termination of Closed Commands (cont'd)

- Only one of $C[c]$ and $C[c']$ terminates. Then $O'(C[c]) \neq O'(C[c'])$. We are done.

- Both $C[c]$ and $C[c']$ terminate. So $[\![c]\!]_{comm}\,\sigma \neq \bot \neq [\![c']\!]_{comm}\,\sigma$.
  Since $[\![c]\!]_{comm}\,\sigma \neq [\![c']\!]_{comm}\,\sigma$, there exist $x$ and $k$ such that $[\![c]\!]_{comm}\,\sigma\, x = k \neq [\![c']\!]_{comm}\,\sigma\, x$.

  We construct another context $C'$:

  $$C' \stackrel{\text{def}}{=} C[\bullet; \mathbf{while}\ x = k\ \mathbf{do\ skip}]\,,$$

  so $C'[c]$ diverges, but $C'[c']$ doesn't. Therefore $O'(C'[c]) = \mathbf{false} \neq \mathbf{true} = O'(C'[c'])$.

# Extension: The **fail** Command

Syntax: $c \; ::= \; \textbf{fail}$

To give semantics to **fail**, we need to extend our semantic domains, just like we lift $\Sigma$ to $\Sigma_\perp$ to give semantics to diverging programs.

We define $\hat{\Sigma} \stackrel{\text{def}}{=} \Sigma \cup \{\textbf{abort}\} \times \Sigma$, and $\hat{\Sigma}_\perp \stackrel{\text{def}}{=} (\hat{\Sigma})_\perp$.

Now $[\![c]\!]_{comm} \in \Sigma \to \hat{\Sigma}_\perp$.

Semantics:

$$
\begin{aligned}
[\![\textbf{fail}]\!]_{comm}\,\sigma &\stackrel{\text{def}}{=} (\textbf{abort}, \sigma) \\
[\![c\,;c']\!]_{comm}\,\sigma &\stackrel{\text{def}}{=} ([\![c']\!]_{comm})_*\,([\![c]\!]_{comm}\,\sigma)
\end{aligned}
$$

where $f_*$ is a lifting of $f \in \Sigma \to \hat{\Sigma}_\perp$ to $\hat{\Sigma}_\perp \to \hat{\Sigma}_\perp$.

# Semantics with **fail** Command

Semantics:

$$
\begin{aligned}
[\![\mathbf{fail}]\!]_{comm}\,\sigma &\stackrel{\text{def}}{=} (\mathbf{abort}, \sigma)\\
[\![c\,;c']\!]_{comm}\,\sigma &\stackrel{\text{def}}{=} ([\![c']\!]_{comm})_*\,([\![c]\!]_{comm}\,\sigma)
\end{aligned}
$$

where $f_*$ is a lifting of $f \in \Sigma \to \hat{\Sigma}_\perp$ to $\hat{\Sigma}_\perp \to \hat{\Sigma}_\perp$:

$$
\begin{aligned}
f_*\,\perp &\stackrel{\text{def}}{=} \perp\\
f_*\,(\mathbf{abort}, \sigma) &\stackrel{\text{def}}{=} (\mathbf{abort}, \sigma)\\
f_*\,\sigma &\stackrel{\text{def}}{=} f\,\sigma
\end{aligned}
$$

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

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

How to define semantics of **newvar** $x := e$ **in** $c$?

# Local Declarations with Failure: Problem

Recall the semantics of local declarations:

$$\begin{aligned}&[\![\textbf{newvar}\ x := e\ \textbf{in}\ c]\!]_{comm}\,\sigma\\ &\quad\stackrel{\text{def}}{=} ((-)\{x \rightsquigarrow \sigma\, x\})_{\perp\!\!\perp}([\![c]\!]_{comm}(\sigma\{x \rightsquigarrow [\![e]\!]_{intexp}\,\sigma\}))\end{aligned}$$

The naive generalization in the presence of failure:

$$\begin{aligned}&[\![\textbf{newvar}\ x := e\ \textbf{in}\ c]\!]_{comm}\,\sigma\\ &\quad\stackrel{\text{def}}{=} ((-)\{x \rightsquigarrow \sigma\, x\})_{*}\,([\![c]\!]_{comm}(\sigma\{x \rightsquigarrow [\![e]\!]_{intexp}\,\sigma\}))\end{aligned}$$

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

$$[\![\textbf{newvar}\ x := 1\ \textbf{in fail}]\!]_{comm}\,\sigma \;=\; \langle \textbf{abort}, \sigma\{x \rightsquigarrow 1\}\rangle$$

so names of local variables can be exported out of scope.

# Local Declarations with Failure

Naive semantics means renaming does not preserve meaning:

$$x := 0\,\mathbf{;}\,[\![\mathbf{newvar}\ x := 1\ \mathbf{in}\ \mathbf{fail}]\!]_{comm}\,\sigma = (\mathbf{abort}, \sigma\{x \rightsquigarrow 1\})$$
$$x := 0\,\mathbf{;}\,[\![\mathbf{newvar}\ y := 1\ \mathbf{in}\ \mathbf{fail}]\!]_{comm}\,\sigma = (\mathbf{abort}, \sigma\{x \rightsquigarrow 0, y \rightsquigarrow 1\})$$

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

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

$$\begin{aligned} f_\dagger\ \perp &= \perp \\ f_\dagger\ (\mathbf{abort}, \sigma) &= (\mathbf{abort}, f\,\sigma) \\ f_\dagger\ \sigma &= f\,\sigma \end{aligned}$$

Then $[\![\mathbf{newvar}\ x := e\ \mathbf{in}\ c]\!]_{comm}\,\sigma$
$$\stackrel{\text{def}}{=} ((-)\{x \rightsquigarrow \sigma\,x\})_\dagger\ ([\![c]\!]_{comm}(\sigma\{x \rightsquigarrow [\![e]\!]_{intexp}\,\sigma\}))$$