Denotational semantics

Semantics and Application to Program Verification

Antoine Miné

École normale supérieure, Paris

year 2013–2014

Course 2

26 February 2014

Operational semantics (last week)

Defined as small execution steps *(transition relation)*
over low-level internal configurations *(states)*

Transitions are chained to define *(maximal)* traces
possibly abstracted as input-output relations *(big-step)*

Denotational semantics (today)

Direct functions from programs to mathematical objects *(denotations)*
by induction on the program syntax *(compositional)*
ignoring intermediate steps and execution details *(no state)*

$\Longrightarrow$ Higher-level, more abstract, more modular.
Tries to decouple a program meaning from its execution.
Focus on the mathematical structures that represent programs.
(founded by Strachey and Scott in the 70s: [Scott-Strachey71])

"Assembly" of semantics vs. "Functional programming" of semantics

Two very different programs

Bubble sort in C

```
int swapped;
do {
  swapped = 0;
  for (int i=1; i<n; i++) {
    if (a[i-1] > a[i]) {
      swap(&a[i-1], &a[i]);
      swapped = 1;
    }
  }
} while (swapped);
```

Quick sort in OCaml

```
let rec sort = function
| [] -> []
| a::rest ->
  let lo, hi =
     List.partition
       (fun y -> y < x) rest
  in
  (sort lo) @ [x] @ (sort hi)
```

- different languages (C / OCaml)
- different algorithms (bubble sort / quick sort)
- different data-types (array / list)

Can we give them the same semantics?

Denotation worlds

- **imperative programs**

  effect of a program: mutate a memory state
  natural denotation: input/output function
  $\mathcal{D} \simeq \textit{memory} \to \textit{memory}$

  challenge: build a whole program denotation
  from denotations of atomic language constructs (modularity)

- **functional programs**

  effect of a program: return a value

  model a program of type `a -> b` as a function $\mathcal{D}_a \to \mathcal{D}_b$,
  of type `(a -> b) -> c` as a function $(\mathcal{D}_a \to \mathcal{D}_b) \to \mathcal{D}_c$, etc.

  challenge: polymorphic or untyped languages

- other paradigms: parallel, probabilistic, etc.

$\Longrightarrow$ very rich theory of mathematical structures
(Scott domains, cartesian closed categories, coherent spaces, event structures, game semantics, etc. We will not present them in this overview!)

Course overview

- **Imperative programs**
  - deterministic programs
  - handling errors
  - handling non-determinism
  - meet-over-all-paths vs. fixpoints
  - modularity
  - linking denotational and operational semantics
- **Higher-order programs**
  - monomorphic typed programs: **PCF**
  - linking denotational and operational semantics: full abstraction
  - untyped $\lambda-$calculus: recursive domain equations
- **Practical session**
  - program the denotational semantics
    of a simple imperative (non-)deterministic language

Deterministic imperative programs

A simple imperative language: IMP

IMP expressions

$$\begin{array}{llll} expr & ::= & X & (variable) \\ & \mid & c & (constant) \\ & \mid & \diamond\, expr & (unary\ operation) \\ & \mid & expr \diamond expr & (binary\ operation) \end{array}$$

- variables in a fixed set $X \in \mathbb{V}$
- constants $\mathbb{I} \stackrel{\text{def}}{=} \mathbb{B} \cup \mathbb{Z}$:
  - booleans $\mathbb{B} \stackrel{\text{def}}{=} \{\text{true}, \text{false}\}$
  - integers $\mathbb{Z}$
- operations $\diamond$:
  - integer operations: $+$, $-$, $\times$, $/$, $<$, $\leq$
  - boolean operations: $\neg$, $\wedge$, $\vee$
  - polymorphic operations: $=$, $\neq$

A simple imperative language: IMP

Statements

stat	::=	**skip**	*(do nothing)*
	|	$X \leftarrow expr$	*(assignment)*
	|	*stat*; *stat*	*(sequence)*
	|	**if** *expr* **then** *stat* **else** *stat*	*(conditional)*
	|	**while** *expr* **do** *stat*	*(loop)*

(inspired from the presentation in [Benton96])

Expression semantics

$\mathsf{E}[\![\, expr \,]\!] : \mathcal{E} \rightharpoonup \mathbb{I}$

- environments $\mathcal{E} \stackrel{\text{def}}{=} \mathbb{V} \to \mathbb{I}$ map variables in $\mathbb{V}$ to values in $\mathbb{I}$
- $\mathsf{E}[\![\, expr \,]\!]$ returns a value in $\mathbb{I}$
- $\rightharpoonup$ denotes partial functions (as opposed to $\to$)
  necessary because some operations are undefined
  - $1 + \text{true}$, $1 \wedge 2$ (type mismatch)
  - $3/0$ (invalid value)
- defined by structural induction on abstract syntax trees

  (next slide)

(when we use the notation $\mathsf{X}[\![\, y \,]\!]$, y is a syntactic object; X serves to distinguish between different semantic functions with different signatures, often varying with the kind of syntactic object y (expression, statement, etc.);
$\mathsf{X}[\![\, y \,]\!] \, z$ is the application of the function $\mathsf{X}[\![\, y \,]\!]$ to the object z)

Expression semantics

$\mathsf{E}[\![\, expr \,]\!] : \mathcal{E} \rightharpoonup \mathbb{I}$

$$
\begin{array}{llll}
\mathsf{E}[\![\, c \,]\!]\,\rho & \stackrel{\text{def}}{=} & c & \in \mathbb{I} \\
\mathsf{E}[\![\, V \,]\!]\,\rho & \stackrel{\text{def}}{=} & \rho(V) & \in \mathbb{I} \\
\mathsf{E}[\![\, -e \,]\!]\,\rho & \stackrel{\text{def}}{=} & -v & \in \mathbb{Z} \quad \text{if } v = \mathsf{E}[\![\, e \,]\!]\,\rho \in \mathbb{Z} \\
\mathsf{E}[\![\, \neg e \,]\!]\,\rho & \stackrel{\text{def}}{=} & \neg v & \in \mathbb{B} \quad \text{if } v = \mathsf{E}[\![\, e \,]\!]\,\rho \in \mathbb{B} \\
\mathsf{E}[\![\, e_1 + e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & v_1 + v_2 & \in \mathbb{Z} \quad \text{if } v_1 = \mathsf{E}[\![\, e_1 \,]\!]\,\rho \in \mathbb{Z}, v_2 = \mathsf{E}[\![\, e_2 \,]\!]\,\rho \in \mathbb{Z} \\
\mathsf{E}[\![\, e_1 - e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & v_1 - v_2 & \in \mathbb{Z} \quad \text{if } v_1 = \mathsf{E}[\![\, e_1 \,]\!]\,\rho \in \mathbb{Z}, v_2 = \mathsf{E}[\![\, e_2 \,]\!]\,\rho \in \mathbb{Z} \\
\mathsf{E}[\![\, e_1 \times e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & v_1 \times v_2 & \in \mathbb{Z} \quad \text{if } v_1 = \mathsf{E}[\![\, e_1 \,]\!]\,\rho \in \mathbb{Z}, v_2 = \mathsf{E}[\![\, e_2 \,]\!]\,\rho \in \mathbb{Z} \\
\mathsf{E}[\![\, e_1 / e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & v_1 / v_2 & \in \mathbb{Z} \quad \text{if } v_1 = \mathsf{E}[\![\, e_1 \,]\!]\,\rho \in \mathbb{Z}, v_2 = \mathsf{E}[\![\, e_2 \,]\!]\,\rho \in \mathbb{Z} \setminus \{0\} \\
\mathsf{E}[\![\, e_1 \wedge e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & v_1 \wedge v_2 & \in \mathbb{B} \quad \text{if } v_1 = \mathsf{E}[\![\, e_1 \,]\!]\,\rho \in \mathbb{B}, v_2 = \mathsf{E}[\![\, e_2 \,]\!]\,\rho \in \mathbb{B} \\
\mathsf{E}[\![\, e_1 \vee e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & v_1 \vee v_2 & \in \mathbb{B} \quad \text{if } v_1 = \mathsf{E}[\![\, e_1 \,]\!]\,\rho \in \mathbb{B}, v_2 = \mathsf{E}[\![\, e_2 \,]\!]\,\rho \in \mathbb{B} \\
\mathsf{E}[\![\, e_1 < e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & v_1 < v_2 & \in \mathbb{B} \quad \text{if } v_1 = \mathsf{E}[\![\, e_1 \,]\!]\,\rho \in \mathbb{Z}, v_2 = \mathsf{E}[\![\, e_2 \,]\!]\,\rho \in \mathbb{Z} \\
\mathsf{E}[\![\, e_1 \leq e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & v_1 \leq v_2 & \in \mathbb{B} \quad \text{if } v_1 = \mathsf{E}[\![\, e_1 \,]\!]\,\rho \in \mathbb{Z}, v_2 = \mathsf{E}[\![\, e_2 \,]\!]\,\rho \in \mathbb{Z} \\
\mathsf{E}[\![\, e_1 = e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & v_1 = v_2 & \in \mathbb{B} \quad \text{if } v_1 = \mathsf{E}[\![\, e_1 \,]\!]\,\rho \in \mathbb{I}, v_2 = \mathsf{E}[\![\, e_2 \,]\!]\,\rho \in \mathbb{I} \\
\mathsf{E}[\![\, e_1 \neq e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & v_1 \neq v_2 & \in \mathbb{B} \quad \text{if } v_1 = \mathsf{E}[\![\, e_1 \,]\!]\,\rho \in \mathbb{I}, v_2 = \mathsf{E}[\![\, e_2 \,]\!]\,\rho \in \mathbb{I}
\end{array}
$$

undefined otherwise

Statement semantics

$\mathsf{S}[\![\, stat \,]\!] : \mathcal{E} \rightharpoonup \mathcal{E}$

- maps an environment before the statement to an environment after the statement
- partial function due to
  - errors in expressions
  - non-termination
- also defined by structural induction

Statement semantics

$\mathsf{S}[\![\, stat \,]\!] : \mathcal{E} \rightharpoonup \mathcal{E}$

- skip: do nothing

  $$\mathsf{S}[\![\, \textbf{skip} \,]\!]\, \rho \stackrel{\text{def}}{=} \rho$$

- assignment: evaluate expression and mutate environment

  $$\mathsf{S}[\![\, X \leftarrow e \,]\!]\, \rho \stackrel{\text{def}}{=} \rho[X \mapsto v] \quad \text{if } \mathsf{E}[\![\, e \,]\!]\, \rho = v$$

- sequence: function composition

  $$\mathsf{S}[\![\, s_1 ; s_2 \,]\!] \stackrel{\text{def}}{=} \mathsf{S}[\![\, s_2 \,]\!] \circ \mathsf{S}[\![\, s_1 \,]\!]$$

- conditional

  $$\mathsf{S}[\![\, \textbf{if } e \textbf{ then } s_1 \textbf{ else } s_2 \,]\!]\, \rho \stackrel{\text{def}}{=} \begin{cases} \mathsf{S}[\![\, s_1 \,]\!]\, \rho & \text{if } \mathsf{E}[\![\, e \,]\!]\, \rho = \text{true} \\ \mathsf{S}[\![\, s_2 \,]\!]\, \rho & \text{if } \mathsf{E}[\![\, e \,]\!]\, \rho = \text{false} \\ \text{undefined} & \text{otherwise} \end{cases}$$

($f[x \mapsto y]$ denotes the function that maps x to y, and any $z \neq x$ to $f(z)$)

Statement semantics: loops

How do we handle loops?

the semantics of loops must satisfy:

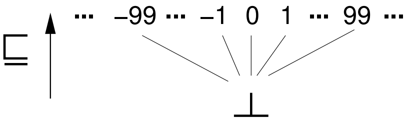
$$\mathsf{S}[\![\, \textbf{while}\ e\ \textbf{do}\ s \,]\!]\, \rho = \begin{cases} \rho & \text{if } \mathsf{E}[\![\, e \,]\!]\, \rho = \text{false} \\ \mathsf{S}[\![\, \textbf{while}\ e\ \textbf{do}\ s \,]\!]\, (\mathsf{S}[\![\, s \,]\!]\, \rho) & \text{if } \mathsf{E}[\![\, e \,]\!]\, \rho = \text{true} \\ \text{undefined} & \text{otherwise} \end{cases}$$

this is a recursive definition, we must prove that:

- the equation has solutions
- choose the right one

$\Longrightarrow$ we use fixpoints on partially ordered sets

Flat orders and partial functions



flat ordering $(\mathbb{I}_\bot, \sqsubseteq)$ on $\mathbb{I}$

- $\mathbb{I}_\bot \stackrel{\text{def}}{=} \mathbb{I} \cup \{\bot\}$ (pointed set)
- $a \sqsubseteq b \stackrel{\text{def}}{\iff} a = \bot \vee a = b$ (partial order)
- every chain is finite, and so has a lub $\sqcup$
  $\implies$ it is a pointed complete partial order (cpo)

$\bot$ denotes the value "undefined" ($\sqsubseteq$ is an information order)

similarly for $\mathcal{E}_\bot \stackrel{\text{def}}{=} \mathcal{E} \cup \{\bot\}$

note that $(\mathcal{E} \rightharpoonup \mathcal{E}) \simeq (\mathcal{E} \to \mathcal{E}_\bot)$

Poset of continuous partial functions

Partial order structure on partial functions $(\mathcal{E}_\perp \xrightarrow{c} \mathcal{E}_\perp, \dot{\sqsubseteq})$

- $\mathcal{E}_\perp \to \mathcal{E}_\perp$ extends $\mathcal{E} \to \mathcal{E}_\perp$
  - domain = co-domain $\implies$ allows composition $\circ$
  - $f \in \mathcal{E} \to \mathcal{E}_\perp$ extended with $f(\perp) \stackrel{\text{def}}{=} \perp$ (strictness)

    $\implies$ if $\mathsf{S}[\![\, s \,]\!]\, x$ is undefined, so is $(\mathsf{S}[\![\, s' \,]\!] \circ \mathsf{S}[\![\, s \,]\!])x$

    such functions are monotonic and continuous

    $(a \sqsubseteq b \implies f(a) \sqsubseteq f(b)$ and $f(\sqcup X) = \sqcup \{ f(x) \mid x \in X \})$

  $\implies$ we restrict $\mathcal{E}_\perp \to \mathcal{E}_\perp$ to continuous functions: $\mathcal{E}_\perp \xrightarrow{c} \mathcal{E}_\perp$

- point-wise order $\dot{\sqsubseteq}$ on functions

  $f \mathrel{\dot{\sqsubseteq}} g \stackrel{\text{def}}{\iff} \forall x\colon f(x) \sqsubseteq g(x)$

- $\mathcal{E}_\perp \xrightarrow{c} \mathcal{E}_\perp$ has a least element: $\dot{\perp} \stackrel{\text{def}}{=} \lambda x.\perp$

- by point-wise lub $\dot{\sqcup}$ of chains, it is also complete $\implies$ a cpo

  $\dot{\sqcup}\, F = \lambda x. \sqcup \{ f(x) \mid f \in F \}$

Fixpoint semantics of loops

to solve the semantic equation, we use a fixpoint of a functional

we use the least fixpoint (most precise for the information order)

$$\mathsf{S}[\![\, \textbf{while}\ e\ \textbf{do}\ s \,]\!] \stackrel{\text{def}}{=} \mathrm{lfp}\, F$$

where : $F : (\mathcal{E}_\perp \to \mathcal{E}_\perp) \to (\mathcal{E}_\perp \to \mathcal{E}_\perp)$

$$F(f)(\rho) = \begin{cases} \rho & \text{if } \mathsf{E}[\![\, e \,]\!]\, \rho = \text{false} \\ f(\mathsf{S}[\![\, s \,]\!]\, \rho) & \text{if } \mathsf{E}[\![\, e \,]\!]\, \rho = \text{true} \\ \perp & \text{otherwise} \end{cases}$$

Theorem

$\mathrm{lfp}\, F$ is well-defined

(remember our equation on $\mathsf{S}[\![\, \textbf{while}\ e\ \textbf{do}\ s \,]\!]$?
it can be rewritten exactly as: $\mathsf{S}[\![\, \textbf{while}\ e\ \textbf{do}\ s \,]\!] = F(\mathsf{S}[\![\, \textbf{while}\ e\ \textbf{do}\ s \,]\!]))$

Fixpoint semantics of loops (proof sketch)

Recall Kleene's theorem:

Kleene's theorem

A continuous function on a cpo has a least fixpoint

Actually, we would prove that $S[\![\mathit{stat}]\!]$ is both well-defined and continuous by induction on the syntax of *stat*:

- base cases: $S[\![\textbf{skip}]\!]$ and $S[\![X \leftarrow e]\!]$ are continuous
- $S[\![\textbf{if } e \textbf{ then } s_1 \textbf{ else } s_2]\!]$: by induction hypothesis
- $S[\![s_1; s_2]\!]$: by induction and because $\circ$ respects continuity
- F is continuous in $(\mathcal{E}_\bot \xrightarrow{c} \mathcal{E}_\bot) \xrightarrow{c} (\mathcal{E}_\bot \xrightarrow{c} \mathcal{E}_\bot)$ by hypotheses and because $\circ$ is continuous
  $\Longrightarrow$ $\mathrm{lfp}\, F$ exists by Kleene's
- $\mathrm{lfp}\, F$ is continuous (simple consequence of Kleene's proof)

Join semantics of loops

Recall another fact about Kleene's fixpoints: $\text{lfp}\, F = \dot{\bigsqcup}_{n\in\mathbb{N}} F^n(\dot{\bot})$

- $F^0(\dot{\bot}) = \dot{\bot}$ is completely undefined (no information)
- $F^1(\dot{\bot})(\rho) = \begin{cases} \rho & \text{if } \mathsf{E}[\![\, e \,]\!]\, \rho = \text{false} \\ \bot & \text{otherwise} \end{cases}$

  environment if the loop is never entered (partial information)
- $F^2(\dot{\bot})(\rho) = \begin{cases} \rho & \text{if } \mathsf{E}[\![\, e \,]\!]\, \rho = \text{false} \\ \mathsf{S}[\![\, s \,]\!]\, \rho & \text{else if } \mathsf{E}[\![\, e \,]\!]\, (\mathsf{S}[\![\, s \,]\!]\, \rho) = \text{false} \\ \bot & \text{otherwise} \end{cases}$

  environment if the loop is iterated at most once
- $F^n(\dot{\bot})(\rho)$

  environment if the loop is iterated at most $n-1$ times
- $\dot{\bigsqcup}_{n\in\mathbb{N}} F^n(\dot{\bot})$

  environment when exiting the loop

  whatever the number of iterations (total information)

Summary

Rewriting the semantics using total functions on cpos:

- $\mathsf{E}[\![\, expr \,]\!] : \mathcal{E}_\perp \xrightarrow{c} \mathbb{I}_\perp$

  returns $\perp$ for an error or if its argument is $\perp$

- $\mathsf{S}[\![\, stat \,]\!] : \mathcal{E}_\perp \xrightarrow{c} \mathcal{E}_\perp$
  - $\mathsf{S}[\![\, \textbf{skip} \,]\!]\,\rho \stackrel{\text{def}}{=} \rho$
  - $\mathsf{S}[\![\, e_1; e_2 \,]\!] \stackrel{\text{def}}{=} \mathsf{S}[\![\, e_2 \,]\!] \circ \mathsf{S}[\![\, e_1 \,]\!]$
  - $\mathsf{S}[\![\, X \leftarrow e \,]\!]\,\rho \stackrel{\text{def}}{=} \begin{cases} \perp & \text{if } \mathsf{E}[\![\, e \,]\!]\,\rho = \perp \\ \rho[X \mapsto \mathsf{E}[\![\, e \,]\!]\,\rho] & \text{otherwise} \end{cases}$
  - $\mathsf{S}[\![\, \textbf{if } e \textbf{ then } s_1 \textbf{ else } s_2 \,]\!]\,\rho \stackrel{\text{def}}{=} \begin{cases} \mathsf{S}[\![\, s_1 \,]\!]\,\rho & \text{if } \mathsf{E}[\![\, e \,]\!]\,\rho = \text{true} \\ \mathsf{S}[\![\, s_2 \,]\!]\,\rho & \text{if } \mathsf{E}[\![\, e \,]\!]\,\rho = \text{false} \\ \perp & \text{otherwise} \end{cases}$
  - $\mathsf{S}[\![\, \textbf{while } e \textbf{ do } s \,]\!] \stackrel{\text{def}}{=} \text{lfp}\, F$

    where $F(f)(\rho) = \begin{cases} \rho & \text{if } \mathsf{E}[\![\, e \,]\!]\,\rho = \text{false} \\ f(\mathsf{S}[\![\, s \,]\!]\,\rho) & \text{if } \mathsf{E}[\![\, e \,]\!]\,\rho = \text{true} \\ \perp & \text{otherwise} \end{cases}$

Errors

Error vs. non-termination

In our semantics $\mathsf{S}[\![\, \textit{stat} \,]\!]\, \rho = \bot$ can mean:

- either *stat* starting on input ρ loops for ever
- or it stops prematurely with an error

$\Longrightarrow$ we would like to distinguish these two cases

Solution:

- add an error value Ω, distinct from $\bot$
- propagate it in the semantics, bypassing computations

  (no further computation after an error)

Expression semantics with errors

We set $\mathcal{E}_{\perp,\Omega} \stackrel{\text{def}}{=} \mathcal{E} \cup \{\perp, \Omega\}$, $\mathbb{I}_{\perp,\Omega} \stackrel{\text{def}}{=} \mathbb{I} \cup \{\perp, \Omega\}$

$\mathsf{E}[\![\, expr \,]\!] : \mathcal{E}_{\perp,\Omega} \stackrel{c}{\to} \mathbb{I}_{\perp,\Omega}$

$$\mathsf{E}[\![\, e \,]\!]\, \perp \quad \stackrel{\text{def}}{=} \quad \perp$$

$$\mathsf{E}[\![\, e \,]\!]\, \Omega \quad \stackrel{\text{def}}{=} \quad \Omega$$

if $\rho \notin \{\, \Omega, \perp \,\}$ then

$$\begin{array}{llll}
\mathsf{E}[\![\, V \,]\!]\, \rho & \stackrel{\text{def}}{=} & \rho(V) & \in \mathbb{I} \\
\mathsf{E}[\![\, c \,]\!]\, \rho & \stackrel{\text{def}}{=} & c & \in \mathbb{I} \\
\mathsf{E}[\![\, -e \,]\!]\, \rho & \stackrel{\text{def}}{=} & -v & \in \mathbb{Z} \quad \text{if } v = \mathsf{E}[\![\, e \,]\!]\, \rho \in \mathbb{Z} \\
 & & \Omega & \qquad\;\; \text{if } \mathsf{E}[\![\, e \,]\!]\, \rho = \Omega \\
\mathsf{E}[\![\, e_1 + e_2 \,]\!]\, \rho & \stackrel{\text{def}}{=} & v_1 + v_2 & \in \mathbb{Z} \quad \text{if } v_1 = \mathsf{E}[\![\, e_1 \,]\!]\, \rho \in \mathbb{Z}, v_2 = \mathsf{E}[\![\, e_2 \,]\!]\, \rho \in \mathbb{Z} \\
 & & \Omega & \qquad\;\; \text{if } \{\mathsf{E}[\![\, e_1 \,]\!]\, \rho, \mathsf{E}[\![\, e_2 \,]\!]\,\} \not\subseteq \mathbb{Z} \\
\mathsf{E}[\![\, e_1 / e_2 \,]\!]\, \rho & \stackrel{\text{def}}{=} & v_1 / v_2 & \in \mathbb{Z} \quad \text{if } v_1 = \mathsf{E}[\![\, e_1 \,]\!]\, \rho \in \mathbb{Z}, v_2 = \mathsf{E}[\![\, e_2 \,]\!]\, \rho \in \mathbb{Z} \setminus \{0\} \\
 & & \Omega & \qquad\;\; \text{if } \mathsf{E}[\![\, e_1 \,]\!]\, \rho \notin \mathbb{Z} \vee \mathsf{E}[\![\, e_2 \,]\!] \,\notin \mathbb{Z} \setminus \{0\}
\end{array}$$

...

(note that $x = \perp \iff \mathsf{E}[\![\, e \,]\!]\, x = \perp$, $x = \Omega \implies \mathsf{E}[\![\, e \,]\!]\, x = \Omega$)

Statements semantics with errors

$\underline{S[\![\, stat \,]\!] : \mathcal{E}_{\bot,\Omega} \xrightarrow{c} \mathcal{E}_{\bot,\Omega}}$

- $S[\![\, s \,]\!] \bot \stackrel{\text{def}}{=} \bot$
- $S[\![\, s \,]\!] \Omega \stackrel{\text{def}}{=} \Omega$
- $S[\![\, \textbf{skip} \,]\!] \rho \stackrel{\text{def}}{=} \rho$
- $S[\![\, s_1; s_2 \,]\!] \stackrel{\text{def}}{=} S[\![\, s_2 \,]\!] \circ S[\![\, s_1 \,]\!]$
- $S[\![\, X \leftarrow e \,]\!] \rho \stackrel{\text{def}}{=} \begin{cases} \rho[X \mapsto v] & \text{if } v = E[\![\, e \,]\!] \rho \in \mathbb{I} \\ \Omega & \text{if } E[\![\, e \,]\!] \rho \in \Omega \end{cases}$
- $S[\![\, \textbf{if } e \textbf{ then } s_1 \textbf{ else } s_2 \,]\!] \rho \stackrel{\text{def}}{=} \begin{cases} S[\![\, s_1 \,]\!] \rho & \text{if } E[\![\, e \,]\!] \rho = \text{true} \\ S[\![\, s_2 \,]\!] \rho & \text{if } E[\![\, e \,]\!] \rho = \text{false} \\ \Omega & \text{otherwise} \end{cases}$

Statements semantics with errors

- $\mathsf{S}[\![\, \textbf{while } e \textbf{ do } s \,]\!] \stackrel{\text{def}}{=} \text{lfp } F$ where

$$F(f)(\rho) = \begin{cases} \bot & \text{if } \rho = \bot \\ \rho & \text{if } \mathsf{E}[\![\, e \,]\!]\, \rho = \text{false} \\ f(\mathsf{S}[\![\, s \,]\!]\, \rho) & \text{if } \mathsf{E}[\![\, e \,]\!]\, \rho = \text{true} \\ \Omega & \text{otherwise} \end{cases}$$

  using the flat ordering $a \sqsubseteq b \iff a = \bot \vee a = b$
  i.e., Ω is not comparable with elements of $\mathcal{E}$
  $\Longrightarrow$ the loop exits immediately at the first error

Several outcome when computing for $\mathsf{S}[\![\, stat \,]\!]\, \rho$

- $\rho' \in \mathcal{E}$: the program terminates successfully
- Ω: the programs terminates with an error
- $\bot$: the program loops forever

More on errors

We can also:

- distinguish different kinds of errors
- tag errors with their location
- track more errors

  e.g., use of uninitialized variables:

  with $\mathcal{E} \stackrel{\text{def}}{=} \mathbb{V} \to (\mathbb{I} \cup \{\text{uninit}\})$

Non-determinism

Why non-determinism?

It is useful to consider non-deterministic programs, to:

- model partially unknown environments (user input)
- abstract away unknown program parts (libraries)
- abstract away too complex parts (rounding errors in floats)
- abstract a set of programs as a single one (parametric programs)

Kinds of non-determinism

- control non-determinism: *stat* ::= **either** s_1 **or** s_2
- data non-determinism: *expr* ::= **random**()

  (more general, as we can write **if random**() = **random**() **then** s_1 **else** s_2)

Consequence on semantics and verification

the semantics should express **all** the possible executions
we must verify **all** the possible executions

Modified language

We extend **IMP** to **NIMP**, an imperative language with non-determinism

NIMP expressions

expr	$::=$	X	(*variable*)
	$\mid$	c	(*constant*)
	$\mid$	$[c_1, c_2]$	(*constant interval*)
	$\mid$	$\diamond$ *expr*	(*unary operation*)
	$\mid$	*expr* $\diamond$ *expr*	(*binary operation*)

$c_1 \in \mathbb{Z} \cup \{-\infty\}$, $c_2 \in \mathbb{Z} \cup \{+\infty\}$

$[c_1, c_2]$ means: a fresh random value between c_1 and c_2 each time the expression is evaluated

Question: is $[0, 1] = [0, 1]$ true or false?

NIMP has the same statements as **IMP**

Expression semantics

$\mathsf{E}[\![\, expr \,]\!] : \mathcal{E} \to \mathcal{P}(\mathbb{I})$

$$
\begin{array}{lcl}
\mathsf{E}[\![\, V \,]\!]\,\rho & \stackrel{\text{def}}{=} & \{\rho(V)\} \\
\mathsf{E}[\![\, c \,]\!]\,\rho & \stackrel{\text{def}}{=} & \{c\} \\
\mathsf{E}[\![\, [c_1, c_2] \,]\!]\,\rho & \stackrel{\text{def}}{=} & \{\, c \in \mathbb{Z} \mid c_1 \leq c \leq c_2 \,\} \\
\mathsf{E}[\![\, -e \,]\!]\,\rho & \stackrel{\text{def}}{=} & \{\, -v \mid v \in \mathsf{E}[\![\, e \,]\!]\,\rho \cap \mathbb{Z} \,\} \\
\mathsf{E}[\![\, \neg e \,]\!]\,\rho & \stackrel{\text{def}}{=} & \{\, \neg v \mid v \in \mathsf{E}[\![\, e \,]\!]\,\rho \cap \mathbb{B} \,\} \\
\mathsf{E}[\![\, e_1 + e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & \{\, v_1 + v_2 \mid v_1 \in \mathsf{E}[\![\, e_1 \,]\!]\,\rho \cap \mathbb{Z},\, v_2 \in \mathsf{E}[\![\, e_2 \,]\!]\,\rho \cap \mathbb{Z} \,\} \\
\mathsf{E}[\![\, e_1 / e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & \{\, v_1 / v_2 \mid v_1 \in \mathsf{E}[\![\, e_1 \,]\!]\,\rho \cap \mathbb{Z},\, v_2 \in \mathsf{E}[\![\, e_2 \,]\!]\,\rho \cap \mathbb{Z} \setminus \{0\} \,\} \\
\mathsf{E}[\![\, e_1 < e_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & \{\, \text{true} \mid \exists v_1 \in \mathsf{E}[\![\, e_1 \,]\!]\,\rho,\, v_2 \in \mathsf{E}[\![\, e_2 \,]\!]\,\rho\colon v_1 \in \mathbb{Z},\, v_2 \in \mathbb{Z},\, v_1 < v_2 \,\} \,\cup \\
 & & \{\, \text{false} \mid \exists v_1 \in \mathsf{E}[\![\, e_1 \,]\!]\,\rho,\, v_2 \in \mathsf{E}[\![\, e_2 \,]\!]\,\rho\colon v_1 \in \mathbb{Z},\, v_2 \in \mathbb{Z},\, v_1 \geq v_2 \,\}
\end{array}
$$

...

- we output a set of values, to account for non-determinism
- we can have $\mathsf{E}[\![\, e \,]\!]\,\rho = \emptyset$ due to errors

  (no need for a special Ω nor $\perp$ element)

Statement semantic domain

Semantic domain:

- statements can output *sets* of statements
  $\Longrightarrow$ use $\mathcal{E} \to \mathcal{P}(\mathcal{E})$
- to allow composition, extend it to $\mathcal{P}(\mathcal{E}) \to \mathcal{P}(\mathcal{E})$
- non-termination and errors can be modeled by $\emptyset$
  (no need for a special Ω nor $\perp$ element)

Note:

we could use $\mathcal{P}(\mathbb{I} \cup \{\Omega\})$ and $\mathcal{P}(\mathcal{E} \cup \{\Omega\})$ to distinguish again non-termination from errors

we won't, to lighten the presentation, but this is not difficult

Statement semantics

$\mathsf{S}[\![\, stat \,]\!] : \mathcal{P}(\mathcal{E}) \to \mathcal{P}(\mathcal{E})$

- $\mathsf{S}[\![\, \textbf{skip} \,]\!]\, R \stackrel{\text{def}}{=} R$
- $\mathsf{S}[\![\, s_1; s_2 \,]\!] \stackrel{\text{def}}{=} \mathsf{S}[\![\, s_2 \,]\!] \circ \mathsf{S}[\![\, s_1 \,]\!]$
- $\mathsf{S}[\![\, X \leftarrow e \,]\!]\, R \stackrel{\text{def}}{=} \{\, \rho[X \mapsto v] \mid \rho \in R,\ v \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\}$
  - pick an environment ρ
  - pick an expression value v in $\mathsf{E}[\![\, e \,]\!]\, \rho$
  - generate an updated environment $\rho[X \mapsto v]$
- $\mathsf{S}[\![\, \textbf{if } e \textbf{ then } s_1 \textbf{ else } s_2 \,]\!]\, R \stackrel{\text{def}}{=}$
  $\mathsf{S}[\![\, s_1 \,]\!]\, \{\, \rho \in R \mid \text{true} \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\} \cup$
  $\mathsf{S}[\![\, s_2 \,]\!]\, \{\, \rho \in R \mid \text{false} \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\}$
  - filter environments according to the value of e
  - execute both branch independently
  - join them with $\cup$

Statement semantics

- $S[\![\,\textbf{while}\ e\ \textbf{do}\ s\,]\!]\, R \stackrel{\text{def}}{=} \{\, \rho \in \text{lfp}\, F \mid \text{false} \in E[\![\, e \,]\!]\, \rho \,\}$

  where $F(X) \stackrel{\text{def}}{=} R \cup S[\![\, s \,]\!]\, \{\, \rho \in X \mid \text{true} \in E[\![\, e \,]\!]\, \rho \,\}$

Justification: lfp F exists

- $(\mathcal{P}(\mathcal{E}), \subseteq, \cup, \cap, \emptyset, \mathcal{E})$ forms a complete lattice
- all semantic functions and F are monotonic and continuous

  in fact, they are strict complete join morphisms

  $S[\![\, s \,]\!]\, (\cup_{i\in\Delta} X_i) = \cup_{i\in\Delta}\, S[\![\, s \,]\!]\, X_i$ and $S[\![\, s \,]\!]\, \emptyset = \emptyset$

  which we write as $S[\![\, s \,]\!] \in \mathcal{P}(\mathcal{E}) \xrightarrow{\cup} \mathcal{P}(\mathcal{E})$

  it is really the *image function* of a function in $\mathcal{E} \to \mathcal{P}(\mathcal{E})$

  $S[\![\, s \,]\!]\, X = \cup\, \{\, S[\![\, s \,]\!]\, \{x\} \mid x \in X \,\}$
- we can apply both Kleene's and Tarksi's fixpoint theorems

Join semantics of loops

- $\mathsf{S}[\![\, \textbf{while } e \textbf{ do } s \,]\!]\, R \stackrel{\text{def}}{=} \{\, \rho \in \operatorname{lfp} F \mid \text{false} \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\}$
  where $F(X) \stackrel{\text{def}}{=} R \cup \mathsf{S}[\![\, s \,]\!]\, \{\, \rho \in X \mid \text{true} \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\}$

(F applies a loop iteration to X and adds back the environments R before the loop)

Recall that $\operatorname{lfp} F = \cup_{n \in \mathbb{N}} F^n(\emptyset)$

- $F^0(\emptyset) = \emptyset$
- $F^1(\emptyset) = R$
  environments before entering the loop
- $F^2(\emptyset) = R \cup \mathsf{S}[\![\, s \,]\!]\, \{\, \rho \in R \mid \text{true} \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\}$
  environments after zero or one loop iteration
- $F^n(\emptyset)$: environments after at most $n-1$ loop iterations
  (just before testing the condition to determine if we should iterate a $n-$th time)
- $\cup_{n \in \mathbb{N}} F^n(\emptyset)$: loop invariant

"Angelic" non-determinism and termination

If *stat* is deterministic (no $[c_1, c_2]$ in expressions)
the semantics is equivalent to our semantics on $\mathcal{E}_\perp \xrightarrow{c} \mathcal{E}_\perp$

Justification: $(\{ E \subseteq \mathcal{E} \mid |E| \leq 1 \}, \subseteq, \cup, \emptyset)$ is isomorphic to $(\mathcal{E}_\perp, \sqsubseteq, \sqcup, \perp)$

In general, we can have several outputs for $\mathsf{S}[\![\, stat \,]\!] \{\rho\} \subseteq \mathcal{E} \cup \{\Omega\}$:

- $\emptyset$: the program never terminates at all
- $\{\Omega\}$: the program never terminates correctly
- $R \subseteq \mathcal{E} \setminus \{\Omega\}$: when the program terminates, it terminates correctly, in an environment in R

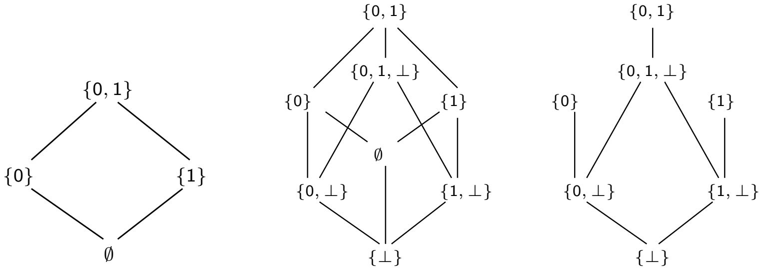
$\Longrightarrow$ we cannot express that a program always terminates!

This is called the "Angelic" semantics, useful for partial correctness

Side-note on non-determinism and termination

Other (more complex) ways to mix non-termination and non-determinism exist

Based on distinguishing $\emptyset$ and $\bot$, and on different order relations $\sqsubseteq$



powerset order
angelic semantics

mixed order
natural semantics

Egli-Milner order
natural semantics

(this is a complex subject, we will say no more)

Path semantics

Syntax and semantics of control paths

Atomic statements

$$\begin{array}{lcll} atomic & ::= & X \leftarrow expr & (assignment) \\ & | & expr? & (boolean\ filter) \end{array}$$

control path: finite sequence of atomic statements

Semantics: $\Pi[\![\, atomic^* \,]\!] : \mathcal{P}(\mathcal{E}) \xrightarrow{\cup} \mathcal{P}(\mathcal{E})$

- $\Pi[\![\, X \leftarrow e \,]\!]R \stackrel{\text{def}}{=} \{\, \rho[X \mapsto v] \mid \rho \in R,\ v \in \mathsf{E}[\![\, e \,]\!]\,\rho \,\}$
- $\Pi[\![\, e? \,]\!]R \stackrel{\text{def}}{=} \{\, \rho \in R \mid \text{true} \in \mathsf{E}[\![\, e \,]\!]\,\rho \,\}$
- $\Pi[\![\, \varepsilon \,]\!]R \stackrel{\text{def}}{=} R$ (empty sequence)
- $\Pi[\![\, a_1; a_2 \,]\!] \stackrel{\text{def}}{=} \Pi[\![\, a_2 \,]\!] \circ \Pi[\![\, a_1 \,]\!]$ (sequence concatenation)
  (well defined as $\circ$ is associative)
- extended to sets of paths: $\Pi[\![\, P \,]\!]R \stackrel{\text{def}}{=} \cup \{\, \Pi[\![\, p \,]\!]R \mid p \in P \,\}$

Control paths of a program

From programs to control paths: $\pi : stat \to \mathcal{P}(atomic^*)$

defined by induction:

$\pi(\textbf{skip}) \stackrel{\text{def}}{=} \varepsilon$

$\pi(X \leftarrow e) \stackrel{\text{def}}{=} \{\, X \leftarrow e \,\}$

$\pi(\textbf{if } e \bowtie 0 \textbf{ then } s_1 \textbf{ else } s_2) \stackrel{\text{def}}{=} (\{\, e? \,\}; \pi(s_1)) \cup (\{\, \neg e? \,\}; \pi(s_2))$ *(branch unzipping)*

$\pi(\textbf{while } e \bowtie 0 \textbf{ do } s) \stackrel{\text{def}}{=} (\cup_{n \in \mathbb{N}} (\{\, e? \,\}; \pi(s))^n); \{\, \neg e? \,\}$ *(loop unrolling)*

$\pi(s_1;\, s_2) \stackrel{\text{def}}{=} \pi(s_1); \pi(s_2)$

(where the concatenation ; is extended to sets of paths)

- π reduces programs to linear sequences of atomic instructions
- $\pi(s)$ is infinite whenever s has loops but each path in $\pi(s)$ has finite length
- some paths may be unfeasible

  ($\forall R$: $\Pi[\![\, p \,]\!] R = \emptyset$, e.g., when unrolling a bounded loop too many times)

Semantic equivalences

Theorem

we have $\sqcap[\![\,\pi(s)\,]\!] = \mathsf{S}[\![\,s\,]\!]$

Proof:
not difficult by structural induction on s
relies on the fact that $\mathsf{S}[\![\,s\,]\!]$ is a strict complete $\cup-$morphism

Terminology:

- $\sqcap[\![\,\pi(s)\,]\!]$ is called the meet-over-all-paths semantics
- $\mathsf{S}[\![\,s\,]\!]$ is called the fixpoint semantics

Note:
In static analysis, $\mathsf{S}[\![\,s\,]\!] : \mathcal{P}(\mathcal{E}) \to \mathcal{P}(\mathcal{E})$ is replaced with $\mathsf{S}^\sharp[\![\,s\,]\!] : \mathcal{E}^\sharp \to \mathcal{E}^\sharp$ on some abstract poset $(\mathcal{E}^\sharp, \sqsubseteq^\sharp)$.
$\mathsf{S}^\sharp[\![\,s\,]\!]$ may not be a complete $\sqcup^\sharp$ morphism (aka distributive), in which case $\sqcap^\sharp[\![\,\pi(s)\,]\!]$ is more precise than $\mathsf{S}^\sharp[\![\,s\,]\!]$, but much more difficult to compute as $\pi(s)$ is often infinite!

Application: program transformation

We want to prove that $\mathsf{S}[\![\, s \,]\!] = \mathsf{S}[\![\, s' \,]\!]$
when s' is obtained from s by some program transformation $\rightsquigarrow$

It is sometime easier to prove that $\Pi[\![\, \pi(s) \,]\!] = \Pi[\![\, \pi(s') \,]\!]$

e.g.: loop unrolling

while e **do** s $\rightsquigarrow$ **if** e **then** $(s;$ **while** e **do** $s)$ **else skip**

Application: parallel programs

Statement extension

$$\begin{array}{lll} \textit{stat} & ::= & \dots \\ & | & \textit{stat} \,||\, \textit{stat} \quad (\textit{parallel composition}) \end{array}$$

Intuitive semantics:

$s_1 \,||\, s_2$ interleaves the executions of s_1 and s_2
and returns when both are finite

we consider assignments and tests to be atomic

many interleavings are possible $\Longrightarrow$ consider them all!
(non-deterministic control)

Application: parallel programs

Modeling interleaving: using control paths

we extent $\pi : stat \to \mathcal{P}(atomic^*)$ with

$$\pi(s_1 \mid\mid s_2) = \cup \{ \, mix(p_1, p_2) \mid p_1 \in \pi(s_1), \; p_2 \in \pi(s_2) \, \}$$

where *mix* is defined by induction on paths length:

- $mix(p, \varepsilon) \stackrel{\text{def}}{=} mix(\varepsilon, p) \stackrel{\text{def}}{=} p$
- $mix((p; a), (q; b)) \stackrel{\text{def}}{=} (mix((p; a), q); b) \cup (mix(p; (q, b)); a)$

  (where $a, b \in atomic$, $p, q \in atomic^*$, and ";" is extended to sets of paths)

$\Pi[\![\pi(s)]\!]$ is well-defined
but there is no longer a corresponding denotational semantics $\mathsf{S}[\![s]\!]$!

(this is a difficult problem to solve)

Modularity

Contexts

Contexts: statements with holes

ctx	$::=$	**skip**	(*do nothing*)
	$\mid$	$X \leftarrow expr$	(*assignment*)
	$\mid$	$ctx; ctx$	(*sequence*)
	$\mid$	**if** $expr$ **then** ctx **else** ctx	(*conditional*)
	$\mid$	**while** $expr$ **do** ctx	(*loop*)
	$\mid$	$\square$	(*hole*)

Substitution: $ctx[\square \mapsto stat] \in stat$, defined by induction (filling holes)

- $c[\square \mapsto s] \stackrel{\text{def}}{=} c$ for assignments and skip contexts
- $(c_1; c_2)[\square \mapsto s] \stackrel{\text{def}}{=} c_1[\square \mapsto s]; c_2[\square \mapsto s]$
- $(\textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2)[\square \mapsto s] \stackrel{\text{def}}{=} \textbf{if } e \textbf{ then } c_1[\square \mapsto s] \textbf{ else } c_2[\square \mapsto s]$
- $(\textbf{while } e \textbf{ do } c)[\square \mapsto s] \stackrel{\text{def}}{=} \textbf{while } e \textbf{ do } c[\square \mapsto s]$
- $\square[\square \mapsto s] \stackrel{\text{def}}{=} s$

Semantics of statements with holes

Context semantics: $\mathsf{C}[\![\, ctx \,]\!] : (\mathcal{P}(\mathcal{E}) \xrightarrow{\cup} \mathcal{P}(\mathcal{E})) \xrightarrow{\cup} \mathcal{P}(\mathcal{E}) \xrightarrow{\cup} \mathcal{P}(\mathcal{E})$

$\simeq$ semantics of statements
but parameterized by the semantics of the hole

$$\mathsf{C}[\![\, \textbf{skip} \,]\!](H)(R) \stackrel{\text{def}}{=} R$$

$$\mathsf{C}[\![\, s_1; s_2 \,]\!](H) \stackrel{\text{def}}{=} \mathsf{C}[\![\, s_2 \,]\!](H) \circ \mathsf{C}[\![\, s_1 \,]\!](H)$$

$$\mathsf{C}[\![\, X \leftarrow e \,]\!](H)(R) \stackrel{\text{def}}{=} \{\, \rho[X \mapsto v] \mid \rho \in R,\ v \in \mathsf{E}[\![\, e \,]\!]\,\rho \,\}$$

$$\begin{aligned} &\mathsf{C}[\![\, \textbf{if } e \textbf{ then } s_1 \textbf{ else } s_2 \,]\!](H)(R) \stackrel{\text{def}}{=} \\ &\quad \mathsf{C}[\![\, s_1 \,]\!](H)(\{\, \rho \in R \mid \text{true} \in \mathsf{E}[\![\, e \,]\!]\,\rho \,\}) \cup \\ &\quad \mathsf{C}[\![\, s_2 \,]\!](H)(\{\, \rho \in R \mid \text{false} \in \mathsf{E}[\![\, e \,]\!]\,\rho \,\}) \end{aligned}$$

$$\begin{aligned} &\mathsf{C}[\![\, \textbf{while } e \textbf{ do } s \,]\!](H)(R) \stackrel{\text{def}}{=} \{\, \rho \in \text{lfp}\, F \mid \text{false} \in \mathsf{E}[\![\, e \,]\!]\,\rho \,\} \\ &\quad \text{where } F(X) \stackrel{\text{def}}{=} R \cup \mathsf{C}[\![\, s \,]\!](H)(\{\, \rho \in X \mid \text{true} \in \mathsf{E}[\![\, e \,]\!]\,\rho \,\}) \end{aligned}$$

$$\mathsf{C}[\![\, \square \,]\!](H)(R) \stackrel{\text{def}}{=} H(R)$$

(H is passed down recursively in $\mathsf{C}[\![\, c \,]\!]$, and used when encountering $\square$)

Substitution vs. context semantics

Theorem

$$C[\![c]\!](S[\![s]\!]) = S[\![c[\square \mapsto s]]\!]$$

$\Longrightarrow$ we can exploit this to perform modular reasoning

- extract a program part s, s.t. $prog = c[\square \mapsto s]$
- compute its semantics in isolation: $S[\![s]\!]$
- use it as $C[\![c]\!](S[\![s]\!])$ to get $S[\![prog]\!]$

useful if s is repeated often in $prog$ as $|c| + |s| \ll |prog|$

Proof: easy by structural induction on c

Application: first order procedures

Statements

$$\begin{array}{lcll} \textit{stat} & ::= & \textbf{skip} & \\ & \mid & \textit{stat}; \textit{stat} & \\ & \mid & \dots & \\ & \mid & f() & \text{(procedure call } f \in \mathcal{F}\text{)} \end{array}$$

$\mathcal{F}$: set of procedure names
$body : \mathcal{F} \to stat$: procedure definition

Assume: no local variables, no recursivity

- substitution semantics:
  $S[\![f()]\!] \stackrel{\text{def}}{=} S[\![body(f)]\!]$, $\simeq$ procedure inlining
- modular semantics:
  $f \mapsto S[\![f()]\!]$ tabulated "bottom-up" on the call graph
  (leaf procedures first)

Side-note on local variables

How do we handle local variables?

Assume distinct sets of variables:

- global variables: $\mathbb{V}_G$
- local variables: $\mathbb{V}_f$ for each procedure $f \in \mathcal{F}$

We need procedure-local environments (scopes) and operators:

- $\forall f \in \mathcal{F}: \mathcal{E}_f \stackrel{\text{def}}{=} (\mathbb{V}_G \cup \mathbb{F}_f) \to \mathbb{I}$
- $\mathsf{S}[\![\, body(f) \,]\!] : \mathcal{P}(\mathcal{E}_f) \stackrel{\cup}{\to} \mathcal{P}(\mathcal{E}_f)$

- going into the scope of f:
  $\rho_{\to f} \stackrel{\text{def}}{=} \lambda X \in \mathbb{V}_G \cup \mathbb{V}_f . \rho(X)$ if $X \in \mathbb{V}_G$, uninit otherwise
- leaving the scope of f:
  $\rho \bowtie_f \rho' \stackrel{\text{def}}{=} \lambda X \in dom(\rho) . \rho'(X)$ if $X \in \mathbb{V}_G$, $\rho(X)$ otherwise

Then: $\mathsf{S}[\![\, f() \,]\!] R \stackrel{\text{def}}{=} \{\, \rho \bowtie_f \rho' \mid \rho \in R,\ \rho' \in \mathsf{S}[\![\, body(f) \,]\!] \{\rho_{\to f}\} \,\}$

Side-note on recursive functions

Context semantics:

$$S[\![\, stat \,]\!] : (\mathcal{F} \to (\mathcal{P}(\mathcal{E}) \stackrel{\cup}{\to} \mathcal{P}(\mathcal{E}))) \stackrel{\cup}{\to} \mathcal{P}(\mathcal{E}) \stackrel{\cup}{\to} \mathcal{P}(\mathcal{E})$$

Assuming the semantics $H(f)$ of each function f is known, we define:

$$\begin{array}{lcl}
S[\![\, \textbf{skip} \,]\!] \, (H)(R) & \stackrel{\text{def}}{=} & R \\
S[\![\, s_1; s_2 \,]\!] \, (H) & \stackrel{\text{def}}{=} & S[\![\, s_2 \,]\!] \, (H) \circ S[\![\, s_1 \,]\!] \, (H) \\
S[\![\, X \leftarrow e \,]\!] \, (H)(R) & \stackrel{\text{def}}{=} & \{ \, \rho[X \mapsto v] \mid \rho \in R, \; v \in E[\![\, e \,]\!] \, \rho \, \} \\
\cdots & & \\
S[\![\, f() \,]\!] \, (H)(R) & \stackrel{\text{def}}{=} & H(f)(R)
\end{array}$$

We must solve the equation $\forall f \in \mathcal{F}$: $H(f) = S[\![\, body(f) \,]\!] \, (H)$

$\Longrightarrow$ again, a fixpoint!

we choose $H = \text{lfp} \, \mathcal{H}$ where $\mathcal{H}(F)(f) \stackrel{\text{def}}{=} S[\![\, body(f) \,]\!] \, (F)$

Question: what interpretation for $\dot{\cup}_{n \in \mathbb{N}} \, \mathcal{H}^n(\dot{\bot})$?

Side-note on function returns

How do we handle early return?

Example: (**if** $x > 0$ **then return**); $x \leftarrow -x$

Solution: maintain two environment sets, D and R:

- D: environments at current point (direct flow)
- R: collected environments at all **return** encountered (return flow)

Semantics $S_c[\![s]\!] : (\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\mathcal{E})) \to (\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\mathcal{E}))$

- sequential statements update the direct flow only:
  $S_c[\![X \leftarrow e]\!] (D, R) \stackrel{\text{def}}{=} (S[\![X \leftarrow e]\!] D, R)$
- returns shift and accumulate the direct flow into the return flow:
  $S_c[\![\textbf{return}]\!] (D, R) \stackrel{\text{def}}{=} (\emptyset, D \cup R)$ (empty direct flow after the return)
- at a normal function end, collect both flows
  if $(D', R') = S_c[\![body(f)]\!] (D, R)$
  then $S_c[\![f()]\!] \stackrel{\text{def}}{=} (D' \cup R', R)$ (the original return flow is restored)
- at control-flow joins, merge both flows (end of tests and loop iterations)
  $(D, R) \sqcup (D', R') \stackrel{\text{def}}{=} (D \cup D', R \cup R')$

$\Longrightarrow$ related to the notion of continuation

Side-note on unstructured jumps

How do we handle unstructured jumps ("gotos")?

Example: (**if** $x > 0$ **then goto** A); ... ; **label** A; ...

Solution: again, continuations!

$S_c[\![s]\!] : (\mathcal{P}(\mathcal{E}) \times (\mathcal{C} \to \mathcal{P}(\mathcal{E}))) \to (\mathcal{P}(\mathcal{E}) \times (\mathcal{C} \to \mathcal{P}(\mathcal{E})))$
where $\mathcal{C}$ is a finite set of goto labels (or break or return points)

- $S_c[\![\textbf{goto}\, A]\!] (D, C) \stackrel{\text{def}}{=} (\emptyset, C[A \mapsto C(A) \cup D])$
- $S_c[\![\textbf{label}\, A]\!] (D, C) \stackrel{\text{def}}{=} (D \cup C(A), C)$

Problem: backward gotos, can be used to simulate loops

Example: **label** A; ... (**if** $x > 0$ **then goto** A); ...

Solution: as for loops, use a fixpoint

e.g., assuming that jumps are local to functions, we iterate each function call

$$S_c[\![f()]\!] (D, C) \stackrel{\text{def}}{=} (\text{fst}\,(\text{lfp}\, \lambda(X, Y).S_c[\![body(f)]\!] (D, Y)),\ C)$$

(at each iteration, the new continuation Y is reinjected, the direct flow restarts at D; after stabilization, the direct flow is returned and the original continuation is restored)

Link between operational and denotational semantics

Motivation

Are the operational and denotational semantics consistent with each other?

Note that:

- systems are actually described operationally
- the denotational semantics is a more abstract representation
  (more suitable for some reasoning on the system)

$\Longrightarrow$ the denotational semantics must be proven faithful
(in some sense) to the operational model to be of any use

Transition systems for our non-deterministic language

Labelled syntax

$$
\begin{array}{lcl}
{}^{\ell}\mathit{stat}^{\ell} & ::= & {}^{\ell}\textbf{skip}^{\ell} \\
 & | & {}^{\ell}X \leftarrow \mathit{expr}^{\ell} \\
 & | & {}^{\ell}\textbf{if } \mathit{expr} \textbf{ then } {}^{\ell}\mathit{stat} \textbf{ else } {}^{\ell}\mathit{stat}^{\ell} \\
 & | & {}^{\ell}\textbf{while } {}^{\ell}\mathit{expr} \textbf{ do } {}^{\ell}\mathit{stat}^{\ell} \\
 & | & {}^{\ell}\mathit{stat};{}^{\ell}\mathit{stat}^{\ell}
\end{array}
$$

$\ell \in \mathcal{L}$: control labels

- statements are decorated with unique control labels $\ell \in \mathcal{L}$
- program configurations in $\Sigma \stackrel{\text{def}}{=} \mathcal{L} \times \mathcal{E}$
  (lower-level than $\mathcal{E}$: we must track program locations)
- transition relation $\tau \subseteq \Sigma \times \Sigma$
  models atomic execution steps

Transition systems for our language

τ is defined by induction on the syntax of statements
$(\sigma, \sigma') \in \tau$ is denoted as $\sigma \to \sigma'$

$$\tau[^{\ell 1}\textbf{skip}^{\ell 2}] \stackrel{\text{def}}{=} \{ (\ell 1, \rho) \to (\ell 2, \rho) \mid \rho \in \mathcal{E} \}$$

$$\tau[^{\ell 1}X \leftarrow e^{\ell 2}] \stackrel{\text{def}}{=} \{ (\ell 1, \rho) \to (\ell 2, \rho[X \mapsto v]) \mid \rho \in \mathcal{E},\ v \in \mathsf{E}[\![e]\!] \rho \}$$

$$\begin{array}{l} \tau[^{\ell 1}\textbf{if } e \textbf{ then } {}^{\ell 2}s_1 \textbf{ else } {}^{\ell 3}s_2{}^{\ell 4}] \stackrel{\text{def}}{=} \\ \quad \{ (\ell 1, \rho) \to (\ell 2, \rho) \mid \rho \in \mathcal{E},\ \text{true} \in \mathsf{E}[\![e]\!] \rho \} \cup \\ \quad \{ (\ell 1, \rho) \to (\ell 3, \rho) \mid \rho \in \mathcal{E},\ \text{false} \in \mathsf{E}[\![e]\!] \rho \} \cup \\ \quad \tau[^{\ell 2}s_1{}^{\ell 4}] \cup \tau[^{\ell 3}s_2{}^{\ell 4}] \end{array}$$

$$\begin{array}{l} \tau[^{\ell 1}\textbf{while } {}^{\ell 2}e \textbf{ do } {}^{\ell 3}s^{\ell 4}] \stackrel{\text{def}}{=} \\ \quad \{ (\ell 1, \rho) \to (\ell 2, \rho) \mid \rho \in \mathcal{E} \} \cup \\ \quad \{ (\ell 2, \rho) \to (\ell 3, \rho) \mid \rho \in \mathcal{E},\ \text{true} \in \mathsf{E}[\![e]\!] \rho \} \cup \\ \quad \{ (\ell 2, \rho) \to (\ell 4, \rho) \mid \rho \in \mathcal{E},\ \text{false} \in \mathsf{E}[\![e]\!] \rho \} \cup \tau[^{\ell 3}s^{\ell 2}] \end{array}$$

$$\tau[^{\ell 1}s_1; {}^{\ell 2}s_2{}^{\ell 3}] \stackrel{\text{def}}{=} \tau[^{\ell 1}s_1{}^{\ell 2}] \cup \tau[^{\ell 2}s_2{}^{\ell 3}]$$

Defines the small-step semantics of a statement

Reminder: special states

Given a labelled statement ${}^{\ell_e}s^{\ell_x}$ and its transition system, we define:

- initial states: $I \stackrel{\text{def}}{=} \{\, (\ell_e, \rho) \,|\, \rho \in \mathcal{E} \,\}$

  note that $\sigma \to \sigma' \implies \sigma' \notin I$

- blocking states: $B \stackrel{\text{def}}{=} \{\, \sigma \in \Sigma \,|\, \forall \sigma'\!: \in \Sigma,\ \sigma \not\to \sigma' \,\}$
  - correct termination: $OK \stackrel{\text{def}}{=} \{\, (\ell_x, \rho) \,|\, \rho \in \mathcal{E} \,\}$

    note that $OK \subseteq B$
  - error: $ERR \stackrel{\text{def}}{=} \{\, (\ell, \rho) \,|\, \ell \neq \ell_x, \rho \in \mathcal{E} \,\} \cap B$

$B = ERR \cup OK,\ ERR \cap OK = \emptyset$

Reminder: maximal trace semantics

Trace: in Σ^∞ (finite or infinite sequence of states)

- starting in an initial state I
- following transitions $\to$
- can only end in a blocking state B (traces are maximal)

i.e.: $t[\![s]\!] = t[\![s]\!]^* \cup t[\![s]\!]^\omega$ where

- finite traces:

  $t[\![s]\!]^* \stackrel{\text{def}}{=} \{ (\sigma_0, \ldots, \sigma_n) \mid n \geq 0, \sigma_0 \in I, \sigma_n \in B, \forall i < n : \sigma_i \to \sigma_{i+1} \}$

- infinite traces:

  $t[\![s]\!]^\omega \stackrel{\text{def}}{=} \{ (\sigma_0, \ldots) \mid \sigma_0 \in I, \forall i \in \mathbb{N} : \sigma_i \to \sigma_{i+1} \}$

Reminder: from traces to big-step semantics

Big-step semantics: abstraction of traces
only remembers the input-output relations

many variants exist:

- "angelic" semantics, in $\mathcal{P}(\Sigma \times \Sigma)$:
  $\mathsf{A}[\![\, s \,]\!] \stackrel{\text{def}}{=} \{\, (\sigma, \sigma') \mid \exists (\sigma_0, \ldots, \sigma_n) \in t[\![\, s \,]\!]^{*} : \sigma = \sigma_0, \sigma' = \sigma_n \,\}$
  (only give information on the terminating behaviors;
  can only prove partial correctness)
- natural semantics, in $\mathcal{P}(\Sigma \times \Sigma_{\perp})$:
  $\mathsf{N}[\![\, s \,]\!] \stackrel{\text{def}}{=} \mathsf{A}[\![\, s \,]\!] \cup \{\, (\sigma, \perp) \mid \exists (\sigma_0, \ldots) \in t[\![\, s \,]\!]^{\omega} : \sigma = \sigma_0 \,\}$
  (models the terminating and non-terminating behaviors;
  can prove total correctness)
- "demoniac" semantics, in $\mathcal{P}(\Sigma \times \Sigma)$:
  $\mathsf{D}[\![\, s \,]\!] \stackrel{\text{def}}{=} \mathsf{A}[\![\, s \,]\!] \cup \{\, (\sigma, \sigma') \mid \exists (\sigma_0, \ldots) \in t[\![\, s \,]\!]^{\omega} : \sigma = \sigma_0, \sigma' \in \Sigma \,\}$
  (models non-termination as chaos;
  cannot prove any property of possibly non-terminating executions)

Example: $\textbf{while}\ X > 0\ \textbf{do}\ X \leftarrow X - [0, 1]$

From big-step to denotational semantics

The angelic denotational and big-step semantics are isomorphic

$S[\![s]\!] = \alpha(A[\![s]\!])$ where

- $\alpha(X) \stackrel{\text{def}}{=} \lambda R.\{\, \rho' \mid \rho \in R,\ ((\ell_e, \rho), (\ell_x, \rho')) \in X \,\}$ *(image of a relation)*
- $\alpha^{-1}(Y) = \{\, ((\ell_e, \rho), (\ell_x, \rho')) \mid \rho \in \mathcal{E}, \rho' \in Y(\{\rho\}) \,\}$

Proof idea: by induction on the syntax of s (quite long)

$\Longrightarrow$ our operational and denotational semantics match

Also, the denotational semantics is an abstraction of the natural semantics

(it forgets about infinite computations)

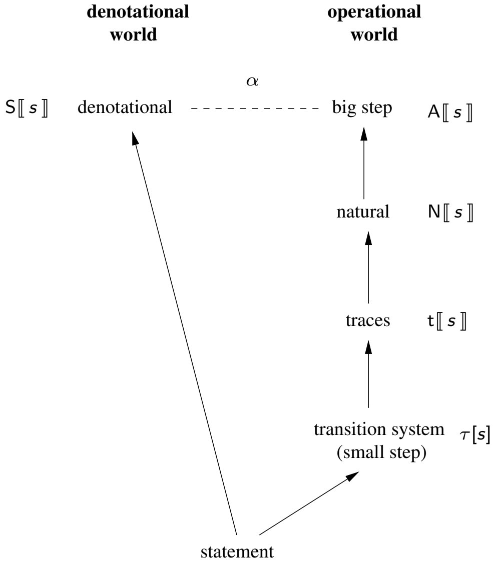
Thesis

All semantics can be compared for equivalence or abstraction

this can be made formal in the **abstract interpretation theory**

(see [Cousot02])

Semantic diagram



Fixpoint formulation

Recall that traces can be expressed as fixpoints:

- $t[\![s]\!]^* = (\text{lfp } F) \cap (I\Sigma^\infty)$ *($\cap(I\Sigma^\infty)$ restricts to traces starting in I)*
  where $F(X) \stackrel{\text{def}}{=} B \cup \{ (\sigma, \sigma_0, \ldots, \sigma_n) \mid \sigma \to \sigma_0 \land (\sigma_0, \ldots, \sigma_n) \in X \}$
- $t[\![s]\!]^\omega = (\text{gfp } F) \cap (I\Sigma^\infty)$
  where $F(X) \stackrel{\text{def}}{=} \{ (\sigma, \sigma_0, \ldots) \mid \sigma \to \sigma_0 \land (\sigma_0, \ldots) \in X \}$

This also holds for the angelic denotational semantics:

- $S[\![s]\!] = \alpha(\text{lfp } F)$ *(α converts relations to functions)*
  where $F(X) \stackrel{\text{def}}{=} (B \times B) \cup \{ (\sigma, \sigma'') \mid \exists \sigma' : \sigma \to \sigma' \land (\sigma', \sigma'') \in X \}$

and many others: natural, denotational, big-stem, denotational,...

Thesis

All semantics can be expressed through fixpoints

(again [Cousot02])

Higher-order programs

Monomorphic typed higher order language

PCF language (introduced by Scott in 1969)

type	::=	**int**	(*integers*)
	|	**bool**	(*booleans*)
	|	$type \to type$	(*functions*)
term	::=	X	(*variable* $X \in \mathbb{V}$)
	|	c	(*constant*)
	|	$\lambda X^{type}.term$	(*abstraction*)
	|	$term\ term$	(*application*)
	|	$\mathbf{Y}^{type}\ term$	(*recursion*)
	|	Ω^{type}	(*failure*)

PCF (programming computable functions) is a $\lambda-$calculus with:

- a monomorphic type system (*unlike ML*)
- explicit type annotations X^{type}, $\mathbf{Y}^{type}$, Ω^{type} (*unlike ML*)
- an explicit recursion combiner **Y** (*unlike untyped $\lambda-$calculus*)
- constants, including $\mathbb{Z}$, $\mathbb{B}$ and a few built-in functions
  (arithmetic and comparisons in $\mathbb{Z}$, if-then-else, etc.)

Semantic domains

What should be the domain of $\mathsf{T}[\![\, term \,]\!]$?

Difficulty: *term* contains heterogeneous objects: constants, functions, second order functions, etc.

Solution: use the type information

each term m can be given a type $typ(m)$
use one semantic domain $\mathcal{D}_t$ per type t
then $\mathsf{T}[\![\, m \,]\!] : \mathcal{E} \to \mathcal{D}_{typ(m)}$ where $\mathcal{E} \stackrel{\text{def}}{=} \mathbb{V} \to (\cup_{t \in type}\, \mathcal{D}_t)$

Domain definition by induction on the syntax of types

- $\mathcal{D}_{\textbf{int}} \stackrel{\text{def}}{=} \mathbb{Z}_\perp$
- $\mathcal{D}_{\textbf{bool}} \stackrel{\text{def}}{=} \mathbb{B}_\perp$
- $\mathcal{D}_{t_1 \to t_2} \stackrel{\text{def}}{=} (\mathcal{D}_{t_1} \stackrel{c}{\to} \mathcal{D}_{t_2})_\perp$

Order on semantic domains

Order: all domains are cpos

- $\mathcal{D}_{\textbf{int}} \stackrel{\text{def}}{=} \mathbb{Z}_\perp$, $\mathcal{D}_{\textbf{bool}} \stackrel{\text{def}}{=} \mathbb{B}_\perp$ use a flat ordering
- $\mathcal{D}_{t_1 \to t_2} \stackrel{\text{def}}{=} (\mathcal{D}_{t_1} \stackrel{c}{\to} \mathcal{D}_{t_2})_\perp$

  with order $f \sqsubseteq g \iff f = \perp \vee (f, g \neq \perp \wedge \forall x: f(x) \sqsubseteq g(x))$

  - $\mathcal{D}_{t_1} \stackrel{c}{\to} \mathcal{D}_{t_2}$ is ordered point-wise
  - each domain has its fresh minimal $\perp$ element

    (to distinguish $\Omega^{\textbf{int} \to \textbf{int}}$ from $\lambda X.^{\textbf{int}}\Omega^{\textbf{int}}$)
  - we restrict $\to$ to continuous functions

    (to be able to take fixpoints)

(see [Scott93])

Denotational semantics

Environments: $\mathcal{E} \stackrel{\text{def}}{=} \mathbb{V} \to (\cup_{t \in type}\, \mathcal{D}_t)$

Semantics: $\mathsf{T}[\![\, m \,]\!] : \mathcal{E} \to \mathcal{D}_{typ(m)}$

$$
\begin{array}{lcl}
\mathsf{T}[\![\, X \,]\!]\,\rho & \stackrel{\text{def}}{=} & \rho(X) \\
\mathsf{T}[\![\, c \,]\!]\,\rho & \stackrel{\text{def}}{=} & c \\
\mathsf{T}[\![\, \boldsymbol{\lambda} X^t.m \,]\!]\,\rho & \stackrel{\text{def}}{=} & \lambda x.\mathsf{T}[\![\, m \,]\!]\,(\rho[X \mapsto x]) \\
\mathsf{T}[\![\, m_1\ m_2 \,]\!]\,\rho & \stackrel{\text{def}}{=} & (\mathsf{T}[\![\, m_1 \,]\!]\,\rho)(\mathsf{T}[\![\, m_2 \,]\!]\,\rho) \\
\mathsf{T}[\![\, \mathbf{Y}^t\ m \,]\!]\,\rho & \stackrel{\text{def}}{=} & \mathrm{lfp}\ (\mathsf{T}[\![\, m \,]\!]\,\rho) \\
\mathsf{T}[\![\, \Omega^t \,]\!]\,\rho & \stackrel{\text{def}}{=} & \bot^t
\end{array}
$$

- program functions $\boldsymbol{\lambda}$ are mapped to mathematical functions λ
- program recursion $\mathbf{Y}$ is mapped to fixpoints lfp
- errors and non-termination are mapped to (typed) $\bot$
- we should prove that $\mathsf{T}[\![\, m \,]\!]$ is indeed continuous (by induction) so that lfp exists, and also that $\mathsf{T}[\![\, m_1 \,]\!]$ is indeed a function (by soundness of typing)

Operational semantics

Operational semantics: based on the $\lambda-$calculus

- states are terms: $\Sigma \stackrel{\text{def}}{=} \textit{term}$
- transition is reduction:

$$
\begin{array}{ll}
(\boldsymbol{\lambda} X^t.m_1)\ m_2 \to m_1[X \mapsto m_2] & (\lambda-\textit{reduction}) \\
\Omega^t \to \Omega^t & (\textit{failure}) \\
\mathbf{Y}^t\ m \to m\ (\mathbf{Y}^t\ m) & (\textit{iteration}) \\
\textit{plus}\ c_1\ c_2 \to (c_1 + c_2) & (\textit{arithmetic}) \\
\textit{if}\ \text{true}\ m_1\ m_2 \to m_1 & (\textit{if-then-else}) \\
\textit{if}\ \text{false}\ m_1\ m_2 \to m_2 & (\textit{if-then-else}) \\
\dfrac{m_1 \to m_1'}{m_1\ m_2 \to m_1'\ m_2} & (\textit{context rule}) \\
\ldots &
\end{array}
$$

- big-step semantics $m \Downarrow$: maximal reductions

$$m \Downarrow = m' \quad \stackrel{\text{def}}{\iff} \quad m \to^* m' \wedge \nexists m'' : m' \to m''$$

(**PCF** *is deterministic*)

Links between operational and denotational semantics

How do we check that operational and denotational semantics match?

check that they have the same view of "semantically equal programs"

- denotational way: we can use $\mathsf{T}[\![\, m_1 \,]\!] = \mathsf{T}[\![\, m_2 \,]\!]$
- we need an operational way to compare functions

  comparing the syntax is too fine grained,

  Example: $(\boldsymbol{\lambda} X^{\textbf{int}}.0) \neq (\boldsymbol{\lambda} X^{\textbf{int}}.\mathit{minus}\ 1\ 1)$, but they have the same denotation

Observational equivalence: observe terms in all contexts

- contexts c: terms with holes $\square$
- $c[m]$ term obtained by substituting m in hole
- *ground* is the set of terms of type **int** or **bool**
- term equivalence $\approx$:

  $m_1 \approx m_2 \stackrel{\text{def}}{\iff} (\forall c\colon c[m_1] \Downarrow = c[m_2] \Downarrow \text{ when } c[m_1] \in \mathit{ground})$

(don't look at a function's syntax, force its full evaluation and look at the value result)

Full abstraction

Full abstraction: $\forall m_1, m_2: m_1 \approx m_2 \iff T[\![m_1]\!] = T[\![m_2]\!]$

Unexpected result: for **PCF**, $\Leftarrow$ holds (adequacy), but not $\Rightarrow$!

(full abstraction concept introduced by Milner in 1975, proof by Plotkin 1977)

Compare with: **IMP**, **NIMP** are fully abstract

$\forall s_1, s_2 \in stat: S[\![s_1]\!] = S[\![s_2]\!] \iff \forall c: A[\![c[s_1]]\!] = A[\![c[s_2]]\!]$

Intuitive explanation:

Domains such as $\mathcal{D}_{t_1 \to t_2}$ contain many functions, most of them do not correspond to *any* program (this is expected: many functions are not computable).

The problem is that, if m_1, m_2 have the form $\lambda X^{t_1 \to t_2}.m$, $T[\![m_1]\!] = T[\![m_2]\!]$ imposes $T[\![m_1]\!] f = T[\![m_2]\!] f$ for all $f \in \mathcal{D}_{t_1 \to t_2}$, including many f that are not computable.

It is actually possible to construct m_1, m_2 where $T[\![m_1]\!] f \neq T[\![m_2]\!] f$ only for some non-program functions f, so that $m_1 \approx m_2$ actually holds

Two solutions come to mind:

- enrich the language to express more functions in $\mathcal{D}_{t_1 \to t_2}$
- restrict $\mathcal{D}_{t_1 \to t_2}$ to contain less non-program objects

Fruitful but complex research topic...

Full abstraction

Example: the parallel or function *por*

$$por(a)(b) \stackrel{\text{def}}{=} \begin{cases} \text{true} & \text{if } a = \text{true} \vee b = \text{true} \\ \text{false} & \text{if } a = \text{false} \wedge b = \text{false} \\ \bot & \text{otherwise} \end{cases}$$

por can observe a and b concurrently, and return as soon as one returns true
compare with sequential *or*, where $\forall b$: $or(\bot)(b) = \bot$

We have the following non-obvious result:

- *por* cannot be defined in **PCF**
  (*por* is a parallel construct, **PCF** is a sequential language)
- **PCF**+*por* is fully abstract

(see [Ong95], [Winskel97] for references on the subject)

Recursive domain equations

Untyped higher order language

$\lambda-$calculus *(with arithmetic)*

term	$::=$	X	(*variable* $X \in \mathbb{V}$)
	$\mid$	c	(*constants*)
	$\mid$	$\lambda X.term$	(*abstraction*)
	$\mid$	*term term*	(*application*)
	$\mid$	Ω	(*failure*)

- we can write truly polymorphic functions:
  e.g., $\lambda X.X$
  (in **PCF** we would have to choose a type: **int** $\to$ **int** or **bool** $\to$ **bool** or (**int** $\to$ **int**) $\to$ (**int** $\to$ **int**) or ...)
- no need for a recursion combinator **Y**
  (we can define $\mathbf{Y} \stackrel{\text{def}}{=} \lambda F.(\lambda X.F\ (X\ X))(\lambda X.F\ (X\ X))$, not typable in **PCF**)
- operational semantics based on reduction similar to **PCF**
- denotational semantics also similar to **PCF**, but...

Domain equations

How to choose the domain of denotations $\mathsf{T}[\![\, m \,]\!]$?

- we need a unique domain $\mathcal{D}$ for all terms
  (*no type information to help us*)
- $\lambda X.X$ is a function
  $\Longrightarrow$ it should have denotation in $(\mathcal{X} \rightarrow \mathcal{Y})_\perp$ for some $\mathcal{X}, \mathcal{Y} \subseteq \mathcal{D}$
- $\lambda X.X$ is polymorphic; it accepts any term as argument
  $\Longrightarrow \mathcal{D} \subseteq \mathcal{X}, \mathcal{Y}$

We have a domain equation to solve:

$$\mathcal{D} \simeq (\mathbb{Z} \cup \mathbb{B} \cup (\mathcal{D} \rightarrow \mathcal{D}))_\perp$$

Problem: no solution in set theory

($\mathcal{D} \rightarrow \mathcal{D}$ *has a strictly larger cardinal than* $\mathcal{D}$)

Inverse limits

Given a fixpoint domain equation $\mathcal{D} = F(\mathcal{D})$
we construct an infinite sequence of domains:

- $\mathcal{D}_0 \stackrel{\text{def}}{=} \{\bot\}$
- $\mathcal{D}_{i+1} \stackrel{\text{def}}{=} F(\mathcal{D}_i)$

We require the existence of continuous retractions:

- $\gamma_i : \mathcal{D}_i \xrightarrow{c} \mathcal{D}_{i+1}$ *(embedding)*
- $\alpha_i : \mathcal{D}_{i+1} \xrightarrow{c} \mathcal{D}_i$ *(projection)*
- $\alpha_i \circ \gamma_i = \lambda x.x$ *($\mathcal{D}_i \simeq$ a subset of $\mathcal{D}_{i+1}$)*
- $\gamma_i \circ \alpha_i \sqsubseteq \lambda x.x$ *($\mathcal{D}_{i+1}$ can be approximated by $\mathcal{D}_i$)*

This is denoted: $\mathcal{D}_0 \underset{\gamma_0}{\overset{\alpha_0}{\leftrightarrows}} \mathcal{D}_1 \underset{\gamma_1}{\overset{\alpha_1}{\leftrightarrows}} \cdots$

Inverse limit: $\mathcal{D}_\infty \stackrel{\text{def}}{=} \{\, (a_0, a_1, \ldots) \mid \forall i: a_i \in \mathcal{D}_i \wedge a_i = \alpha(a_{i+1}) \,\}$

(infinite sequences of elements; able to represent an element of any $\mathcal{D}_i$)

Inverse limits

Inverse limits: $\mathcal{D}_\infty \stackrel{\text{def}}{=} \{ (a_0, a_1, \ldots) \mid \forall i: a_i \in \mathcal{D}_i \wedge a_i = \alpha(a_{i+1}) \}$

Theorem

$\mathcal{D}_\infty$ is a cpo and $F(\mathcal{D}_\infty)$ is isomorphic to $\mathcal{D}_\infty$

Application to $\lambda-$calculus

If we restrict ourself to continuous functions
retractions can be computed for $F(\mathcal{D}) \stackrel{\text{def}}{=} (\mathbb{Z} \cup \mathbb{B} \cup (\mathcal{D} \stackrel{c}{\to} \mathcal{D}))_\perp$

$\Longrightarrow$ we found our semantic domain!

(pioneered by [Scott-Strachey71], see [Abramsky-Jung94] for a reference)

Restrictions of function spaces

The restriction to continuous functions seems merely technical but there are some valid justification:

- all the denotations in **IMP**, **NIMP**, **PCF** were continuous

  (*this appeared naturally, not as an a priori restriction*)

- intuitively, computable functions should at least be monotonic

  recall that $\sqsubseteq$ is an information order

  a function cannot give a more precise result with less information

  e.g.: if $f(a) = \perp$ for some $a \neq \perp$, then $f(\perp) = \perp$

- continuity is also reasonable

  given a problem on an infinite data set S

  computers can only process finite parts S_i of S

  continuity ensures that the solution of S is contained in that of all S_i

  e.g.: if $0 \sqsubseteq 1 \sqsubseteq \cdots \sqsubseteq \omega$ and $\forall i < \omega$: $f(i) = 0$, then $f(\omega)$ should also be 0

Data-types

Solution domains of recursive equations can also give the semantics of a variety of inductive or polymorphic data-types

Examples:

- integer lists:
  $\mathcal{D} = (\{empty\} \cup (\mathbb{Z} \times \mathcal{D}))_\perp$
- pairs:
  $\mathcal{D} = (\mathbb{Z} \cup (\mathcal{D} \times \mathcal{D}))_\perp$
  (allows arbitrary nested pairs, and also contains trees and lists)
- records:
  $\mathcal{D} = (\mathbb{Z} \cup (\mathbb{N} \to \mathcal{D}))_\perp$
  (fields are named by integer position)
- sum types:
  $\mathcal{D} = (\mathbb{Z} \cup (\{1\} \times \mathcal{D}) \cup (\{2\} \times \mathcal{D}))_\perp$
  (we "tag" each case of the sum with an integer)

Bibliography

Courses and references on denotational semantics:

[Benton96] **P. N. Benton**. *Semantics of programming languages* In University of Cambridge, 1996.

[Winskel97] **G. Winskel**. *Lecture notes on denotational semantics.* In University of Cambridge, 1997.

[Schmidt86] **D. Schmidt**. *Denotational semantics. A methodology for language development.* In Allyn and Bacon, 1986.

[Abramsky-Jung94] **S. Abramsky and A. Jung**. *Domain theory.* In Handbook of Logic in Computer Science, Clarendon Press, Oxford, 1994.

Bibliography

Research articles and surveys:

[Scott-Strachey71] **D. Scott and C. Strachey**. *Toward a mathematical semantics for computer languages.* In Oxford Programming Research Group Technical Monograph. PRG-6. 1971.

[Scott93] **D. Scott**. *A type-theoretical alternative to ISWIM, CUCH, OWHY.* In TCS, 121(1–2):411–440, 1993.

[Cousot02] **P. Cousot**. *Constructive design of a hierarchy of semantics of a transition system by abstract interpretation.* In TCS, 277(1–2):47–103, 2002.

[Ong95] **C.-H. L. Ong**. *Correspondence between operational and denotational semantics: the full abstraction problem for PCF* In Oxford University, 1995.