Denotational semantics

Semantics and Application to Program Verification

Antoine Miné

École normale supérieure, Paris

year 2015–2016

Course 4

4 March 2016

Introduction

Operational semantics (state and trace) (last two weeks)

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

Transitions are chained to define maximal traces.

Denotational semantics (today)

Direct functions from programs to mathematical objects, *(denotations)*
defined 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" semantics vs. "Functional programming" semantics.

often: semantics for practical verification vs. semantics for computer theorists

Two very different programs

Bubble sort in C

```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

```ocaml
let rec sort = function
| [] -> []
| x::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 programming principles (loop / recursion)
- 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
  domain $\mathcal{D} \simeq \textit{memory} \to \textit{memory}$

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

- **functional programs**

  effect of a program: return a value (without any side-effect)
  model a program of type `a -> b` as a function in $\mathcal{D}_a \to \mathcal{D}_b$

  challenge: choose $\mathcal{D}$ to allow 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**
  - IMP: deterministic programs
  - NIMP: handling non-determinism
  - linking denotational and operational semantics
- **Higher-order programs**
  - PCF : monomorphic typed programs
  - linking denotational and operational semantics: full abstraction
  - untyped $\lambda-$calculus: recursive domain equations
- **Practical session** (room INFO 4)
  - program the denotational semantics
    of a simple imperative (non-)deterministic language
    (IMP, NIMP)

Deterministic imperative programs

A simple imperative language: IMP

IMP expressions

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

- 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

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

- environments $\mathcal{E} \overset{\text{def}}{=} \mathbb{V} \to \mathbb{I}$ map variables in $\mathbb{V}$ to values in $\mathbb{I}$
- $\mathrm{E}[\![\, expr \,]\!]$ 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 $\mathrm{X}[\![\, y \,]\!]$, 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.);
$\mathrm{X}[\![\, y \,]\!]\, z$ is the application of the function $\mathrm{X}[\![\, y \,]\!]$ to the object z

Expression semantics

$\mathsf{E}\llbracket\, expr \,\rrbracket : \mathcal{E} \rightharpoonup \mathbb{I}$

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

undefined otherwise

Statement semantics

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

- 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

  $$\mathsf{S}[\![\, \mathbf{skip} \,]\!]\, \rho \overset{\text{def}}{=} \rho$$

- assignment: evaluate expression and mutate environment

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

- sequence: function composition

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

- conditional

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

$f[x \mapsto y]$ 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:

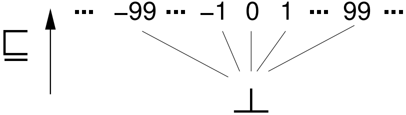
$$\mathsf{S}[\![\,\textbf{while}\ e\ \textbf{do}\ s\,]\!]\,\rho = \begin{cases} \rho & \text{if } \mathsf{E}[\![\,e\,]\!]\,\rho = \text{false} \\ \mathsf{S}[\![\,\textbf{while}\ e\ \textbf{do}\ s\,]\!]\,(\mathsf{S}[\![\,s\,]\!]\,\rho) & \text{if } \mathsf{E}[\![\,e\,]\!]\,\rho = \text{true} \\ \text{undefined}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

This is a recursive definition; we must prove that:

- the equation has solution(s);
- in case there are several solutions, there is a single “right” one;

$\Longrightarrow$ we use fixpoints of operators over partially ordered sets.

Flat orders and partial functions



Flat ordering $(\mathbb{I}_\perp, \sqsubseteq)$ on $\mathbb{I}$:

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

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

Similarly for $\mathcal{E}_\perp \overset{\text{def}}{=} \mathcal{E} \cup \{\perp\}$.

Note that $(\mathcal{E} \rightharpoonup \mathcal{E}) \simeq (\mathcal{E} \to \mathcal{E}_\perp)$
$\implies$ we will now use total functions only.

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 $f(\perp) \overset{\text{def}}{=} \perp$ (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

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

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

- 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 actually the least fixpoint. (most precise for the information order)

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

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

$\operatorname{lfp} F$ 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

To use the theorem we prove that $\mathsf{S}[\![\, \mathit{stat} \,]\!]$ is continuous (and is well-defined) by induction on the syntax of *stat*:

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

  moreover, $\mathrm{lfp}\, F$ is continuous (simple consequence of Kleene's proof)
  $\Longrightarrow$ $\mathsf{S}[\![\, \textbf{while } e \textbf{ do } s \,]\!]$ is continuous

Join semantics of loops

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

- $F^0(\dot{\bot}) = \dot{\bot}$ 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)

Error vs. non-termination

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

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

Note : we could distinguish between the two cases by :

- adding an error value Ω, distinct from $\bot$
- propagating it in the semantics, bypassing computations (no further computation after an error)

Summary

Rewriting the semantics using total functions on cpos with $\bot$:

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

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

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

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

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)
- handle a set of programs as a single one (parametric programs)

Kinds of non-determinism

- data non-determinism: $expr ::= \textbf{random}()$
- control non-determinism: $stat ::= \textbf{either } s_1 \textbf{ or } s_2$

  but we can write "$\textbf{either } s_1 \textbf{ or } s_2$" as "$\textbf{if random}() = 0 \textbf{ then } s_1 \textbf{ else } s_2$"

Consequence on semantics and verification

we want to verify **all** the possible executions

$\Longrightarrow$ the semantics should express **all** the possible executions

Modified language

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

NIMP language

$$
\begin{array}{lcll}
\mathit{expr} & ::= & X & \textit{(variable)} \\
& | & c & \textit{(constant)} \\
& | & [c_1, c_2] & \textit{(constant interval)} \\
& | & \diamond\, \mathit{expr} & \textit{(unary operation)} \\
& | & \mathit{expr} \diamond \mathit{expr} & \textit{(binary operation)}
\end{array}
$$

NIMP has the same statements as **IMP**

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

$[c_1, c_2]$ means: return 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?

Expression semantics

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

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

$\ldots$

- 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:

- a statement can output a set of environments
  $\Longrightarrow$ 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)

Statement semantics

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

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

Statement semantics

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

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

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

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

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

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

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

Join semantics of loops

- $\mathsf{S}[\![\, \textbf{while}\ e\ \textbf{do}\ s \,]\!]\, R \stackrel{\text{def}}{=} \{\, \rho \in \text{lfp}\ F \mid \text{false} \in \mathsf{E}[\![\, e \,]\!]\, \rho \,\}$
  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 $\text{lfp}\, F = \cup_{n \in \mathbb{N}}\, F^n(\emptyset)$

- $F^0(\emptyset) = \emptyset$
- $F^1(\emptyset) = R$
  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: $(\{\, E \subseteq \mathcal{E} \,|\, |E| \leq 1 \,\}, \subseteq, \cup, \emptyset)$ is isomorphic to $(\mathcal{E}_\perp, \sqsubseteq, \sqcup, \perp)$

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

- $\emptyset$: 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

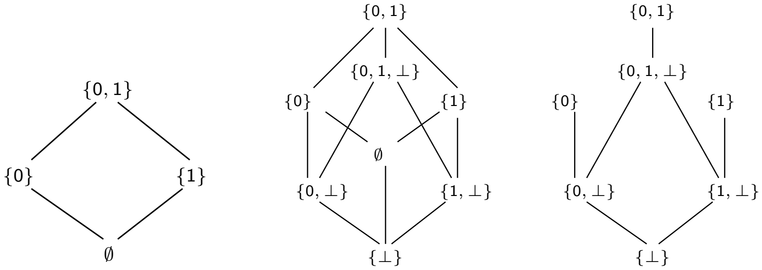
$\Longrightarrow$ we cannot express that a program always terminates!

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

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)

Link between operational and denotational semantics

Motivation

Are the operational and denotational semantics consistent with each other?

Note that:

- systems are actually described operationally
  (previous courses)
- 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

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

$\ell \in \mathcal{L}$: 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 $\sigma \to \sigma'$

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

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

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

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

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

Defines the small-step semantics of a statement

(the semantics of expressions is still in denotational form)

Special states

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

- initial states: $I \stackrel{\text{def}}{=} \{\, (\ell_e, \rho) \mid \rho \in \mathcal{E} \,\}$
  note that $\sigma \to \sigma' \implies \sigma' \notin I$
- blocking states: $B \stackrel{\text{def}}{=} \{\, \sigma \in \Sigma \mid \forall \sigma' \colon \in \Sigma,\ \sigma \not\to \sigma' \,\}$
  - correct termination: $OK \stackrel{\text{def}}{=} \{\, (\ell_x, \rho) \mid \rho \in \mathcal{E} \,\}$
    note that $OK \subseteq B$
  - error: $ERR \stackrel{\text{def}}{=} B \cap \{\, (\ell, \rho) \mid \ell \neq \ell_x, \rho \in \mathcal{E} \,\}$

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

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:

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

- infinite traces:

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

From traces to big-step semantics

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

many variants exist:

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

Exercise: compute the semantics of "**while** $X > 0$ **do** $X \leftarrow X - [0,1]$"

From big-step to denotational semantics

The angelic denotational and big-step semantics are isomorphic

(isomorphism between relations and strict complete join morphisms)

$\mathsf{S}[\![\, s \,]\!] = \alpha(\mathsf{A}[\![\, s \,]\!])$ where

- $\alpha(X) \overset{\text{def}}{=} \lambda R.\{\, \rho' \mid \rho \in R,\ ((\ell_e, \rho), (\ell_x, \rho')) \in X \,\}$ *(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

$\Longrightarrow$ our operational and denotational semantics match

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

(it forgets about infinite computations)

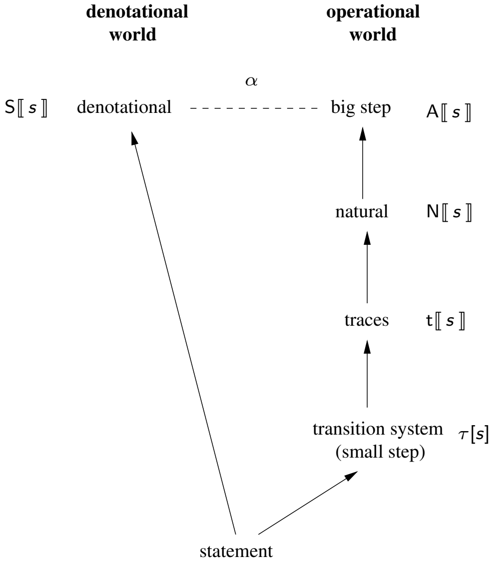
Thesis

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:

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

This also holds for the angelic denotational semantics:

- $\mathsf{S}[\![\, s \,]\!] = \alpha(\mathrm{lfp}\, F)$ *(α converts relations to functions)*
  where $F(X) \overset{\text{def}}{=} (B \times B) \cup \{\, (\sigma, \sigma'') \,|\, \exists \sigma' \colon \sigma \to \sigma' \wedge (\sigma', \sigma'') \in X \,\}$

and many others: natural, denotational, big-step, 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)

$$
\begin{array}{rcll}
\textit{type} & ::= & \mathbf{int} & (\textit{integers}) \\
 & | & \mathbf{bool} & (\textit{booleans}) \\
 & | & \textit{type} \to \textit{type} & (\textit{functions}) \\
\textit{term} & ::= & X & (\textit{variable } X \in \mathbb{V}) \\
 & | & c & (\textit{constant}) \\
 & | & \lambda X^{\textit{type}}.\textit{term} & (\textit{abstraction}) \\
 & | & \textit{term term} & (\textit{application}) \\
 & | & \mathbf{Y}^{\textit{type}}\ \textit{term} & (\textit{recursion}) \\
 & | & \Omega^{\textit{type}} & (\textit{failure})
\end{array}
$$

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 $\mathsf{T}[\![\, \mathit{term} \,]\!]$?

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

Solution: use the type information

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

Domain definition by induction on the syntax of types

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

Order on semantic domains

Order: all domains are cpos

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

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

  - $\mathcal{D}_{t_1} \overset{c}{\to} \mathcal{D}_{t_2}$ 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: $\mathsf{T}[\![\, m \,]\!] : \mathcal{E} \to \mathcal{D}_{typ(m)}$

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

- 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) $\perp$
- 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: $\Sigma \overset{\text{def}}{=} \textit{term}$
- transitions correspond to reductions:

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

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

$$m \Downarrow = m' \quad \overset{\text{def}}{\Longleftrightarrow} \quad m \to^* m' \land \nexists m'' : m' \to m''$$

(**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: $(\lambda X^{\mathbf{int}}.0) \neq (\lambda X^{\mathbf{int}}.\mathit{minus}\ 1\ 1)$, 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 $\approx$:

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

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

Full abstraction

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

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

$\forall s_1, s_2 \in \mathit{stat} \colon \mathsf{S}[\![\, s_1 \,]\!] = \mathsf{S}[\![\, s_2 \,]\!] \iff \forall c \colon \mathsf{A}[\![\, c[s_1] \,]\!] = \mathsf{A}[\![\, c[s_2] \,]\!]$

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 $\boldsymbol{\lambda} X^{t_1 \to t_2}.m$, $\mathsf{T}[\![\, m_1 \,]\!] = \mathsf{T}[\![\, m_2 \,]\!]$ imposes $\mathsf{T}[\![\, m_1 \,]\!]\, f = \mathsf{T}[\![\, m_2 \,]\!]\, f$ 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 $\mathsf{T}[\![\, m_1 \,]\!]\, f \neq \mathsf{T}[\![\, m_2 \,]\!]\, f$ 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 $\mathcal{D}_{t_1 \to t_2}$ (next slide)
- 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*)
	$\mid$	$\boldsymbol{\lambda} X.term$	(*abstraction*)
	$\mid$	*term term*	(*application*)
	$\mid$	Ω	(*failure*)

- we can write truly polymorphic functions: e.g., $\boldsymbol{\lambda} X.X$

  (in **PCF** we would have to choose a type: **int** → **int** or **bool** → **bool** or (**int** → **int**) → (**int** → **int**) or ...)

- no need for a recursion combinator **Y**

  (we can define $\mathbf{Y} \stackrel{\text{def}}{=} \boldsymbol{\lambda} F.(\boldsymbol{\lambda} X.F\ (X\ X))(\boldsymbol{\lambda} X.F\ (X\ X))$, not typable in **PCF**)

- operational semantics based on reduction, similarly to **PCF**

- denotational semantics also similar to **PCF**, but...

Domain equations

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

- 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 $(\mathcal{X} \to \mathcal{Y})_\perp$ 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:

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

Problem: no solution in set theory

($\mathcal{D} \to \mathcal{D}$ *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:

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

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:
$$\begin{tikzcd} \mathcal{D}_0 \arrow[r, shift right, "\gamma_0"'] & \mathcal{D}_1 \arrow[l, shift right, "\alpha_0"'] \arrow[r, shift right, "\gamma_1"'] & \cdots \arrow[l, shift right, "\alpha_1"'] \end{tikzcd}$$

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

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

Inverse limits

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

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 $F(\mathcal{D}) \overset{\text{def}}{=} (\mathbb{Z} \cup \mathbb{B} \cup (\mathcal{D} \xrightarrow{c} \mathcal{D}))_\perp$

$(\gamma_i(f) \overset{\text{def}}{=} \lambda x.f$

$\alpha_i(x) \overset{\text{def}}{=} \cancel{x}$ if $x \in \mathbb{Z} \cup \mathbb{B} \cup \{\perp\}$ and $\alpha_i(f) \overset{\text{def}}{=} f(\perp)$ if $f \in \mathcal{D}_i \xrightarrow{c} \mathcal{D}_i)$

$\Longrightarrow$ 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 justifications:

- 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 $f(a) = \bot$ for some $a \neq \bot$, then $f(\bot) = \bot$
- 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

Domain equations for data-types

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

Examples:

- integer lists:

  $$\mathcal{D} = (\{\textit{empty}\} \cup (\mathbb{Z} \times \mathcal{D}))_\perp$$

- pairs:

  $$\mathcal{D} = (\mathbb{Z} \cup (\mathcal{D} \times \mathcal{D}))_\perp$$

  (allows arbitrary nested pairs, and also contains trees and lists)

- records:

  $$\mathcal{D} = (\mathbb{Z} \cup (\mathbb{N} \rightarrow \mathcal{D}))_\perp$$

  (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.