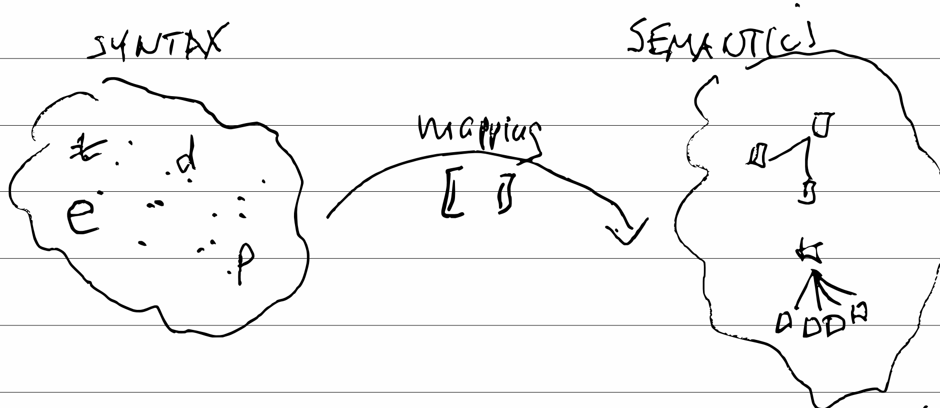
Denotational Semantics

22 Jan 2021

(1) What is denotational semantics
(2) Why would we want a math-y semantics
(3) How do we make one for PCF (λ)?
(4) Is this semantics "good" (in (2)) ?
(5) Does a fully abstract semantics exist?

Scott / Strachey	(1-2)	PRG '67
Plotkin	(3-4)	TCS '77
Milner	(5)	TCS '77

The Very Big Picture



- every phrase is mapped to a semantic elmt
- the mapping is _compositional_

$$[\![e \bullet d]\!] = [\![e]\!] \bullet [\![d]\!]$$

syntactic composition

semantic function

- collection of math. spaces
- .... closed under function space construction $(S \longrightarrow S')$
- mapping is in (Syntax $\longrightarrow$ Semantics)

Compositionality

first impression (Scott)
homomorphism

second impression (Strachey)
environments
stores
continuations
⋮

third impression (Curien et al.)
category theory
as in composition
of arrows

fourth impression (Felleisen)
effect handlers

and many more ...

Why?

Truth

Characterizing truth

Observational (operational) equivalence is the ground truth of a PL.

If

$eval_L$: Syntax $\longrightarrow$ Semantics (specified in <u>any</u> manner)

define $e \cong_L e'$

as for all program contexts C

$$eval(C[e]) = eval(C[e'])$$

Thm

$\cong$ satisfies the following three properties:

(1) trans., refl., symmetric

(2) syntactic congruence ($\forall C$)

(3) can evaluate every program.
($p \cong_L v$)

Thm For any $\mathcal{R}_L$ with these properties,

$e \,\mathcal{R}_L\, e'$ implies $e \cong_L e'$

$\boxed{\text{CANONICITY}}$

It is hard to work with $\cong_L$ if you want to verify programs or write (correct) optimizing compilers. (Π_1')

- always think of <u>all</u> <u>contexts</u>
- simulate the evaluator on them

OXFORD Dreaming:

denotational semantics can completely describe $\cong_L$

Def A denotational semantics is a <u>full</u> <u>description</u> of $\cong_L$:

$$\text{SYN} \xrightarrow{eval} \text{SEM}$$

iff

$$e \cong_L e' \iff eval(e) = eval(e')$$

<u>or</u>

$$\forall C: eval(C[e]) = eval(C[e'])$$

$$\iff$$

$$eval(e) \underbrace{=}_{\text{math. equality}} eval(e')$$

The "organization" of terms

semantic spaces is such

that we can use

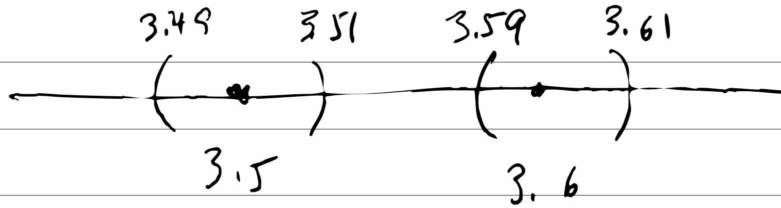
200-year old tools

to analyze SEM, val, and SYN

- algebra
- geometry
- topology

Ex: on $\mathbb{R}$, intervals separate points

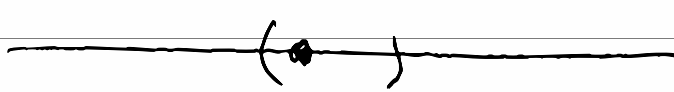
- 3.5 vs 3.6 ✓



"open sets": disjoint

BUT

- $3.\overline{9}$ vs 4.0 ✗



All two open sets containing one also contain the other.

How?

PCF (the typed variant of λ)

$$e = x \mid \lambda x{:}t.e \mid e\,e \mid \mathrm{fix} \mid$$
$$n \mid \mathrm{add1}\, e \mid \mathrm{sub1}\, e \mid$$
$$\mathrm{if0}\; e\; e\; e$$

$$t = \mathrm{int} \mid$$
$$t \to t$$

$$
\begin{aligned}
\text{add} \;=\; & \text{fix}\ (\lambda A : \text{int} \to \text{int}. \\
& \quad \lambda x : \text{int}. \\
& \quad\quad \lambda y : \text{int}. \\
& \quad\quad\quad \underline{\text{if0}}\ \ y \\
& \quad\quad\quad\quad x \\
& \quad\quad\quad\quad \text{add1}\ (A\ x\ (\text{sub1}\ y)))
\end{aligned}
$$

$$\text{mul} = \ldots\ \text{add} \ldots$$

$$\text{equal} = \ldots 0 \ldots 1 \ldots$$

Now we can compute every partial recursive function. $\boxed{\text{TM !}}$

Let's start w/ the meaning of types: $t = \text{int} \mid t \to t$

$[\![\text{int}]\!] = \mathbb{Z}$

$[\![\text{int} \to \text{int}]\!] = \mathbb{Z} \to \mathbb{Z} = \mathbb{Z}^{\mathbb{Z}}$

"keyword" math. functions

So: $\lambda x : \text{int} . \ x$

"means" $\{ (0,0), (1,1), (-1,-1), \ldots \}$

$\boxed{\text{denotes}}$

Problem ↯

$\underline{\text{fix}}\ (\underline{\lambda} f : \text{int} \to \text{int}.\ \underline{\lambda} x.\ \underline{f}\ x)$

or

```
(define (f {x : int}) : int
  (f x))
```

Its type is $\underline{\text{int}} \longrightarrow \underline{\text{int}}$.

It always runs forever.

So:

$\{ (0, ?), (+1, ?), \ldots \}$

$\boxed{\text{what goes here?}}$

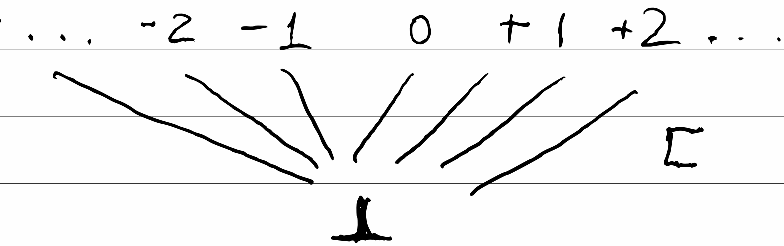
Solution Think of denotations as information about the outcome of a program.

new element: $\perp$

pronounced: "There is no informat." or "bottom"

$\perp$ is less information than $0, 1, 2, \ldots$

So:



When infinite loops are made explicit on the math-y side, we can deal with it explicitly.

Ex 1: $f(\perp) = \perp$

mathematical equality

means if f's argument goes into an infinite loop, it itself returns no information (i.e. is in an infinite loop).

Ex2: $g(\perp) = 1$

means "even if" g's argument is an infinite loop, it returns 1

Task 2

$$[\![\text{int}]\!] = \mathbb{Z}_\perp$$

$$[\![\text{int} \to \text{int}]\!] = \mathbb{Z}_\perp \longrightarrow \mathbb{Z}_\perp$$

Now

$$\underline{\text{fix}}\,(\underline{\lambda} f : \text{int} \to \text{int}.\ \underline{\lambda} x.\ f\,x)$$

denotes $\{ (0,\perp);\ (+1,\perp);\ (-1,\perp);\ \ldots \}$

$$[\![(\text{int} \to \text{int}) \to \text{int}]\!]$$

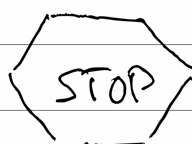
$$= (\mathbb{Z}_\perp \longrightarrow \mathbb{Z}_\perp) \longrightarrow \mathbb{Z}_\perp$$

$$= \mathbb{Z}_\perp^{(\mathbb{Z}_\perp \to \mathbb{Z}_\perp)}$$

Problem 44

This space contains

$$H_{13} = \{ (f, 0) \mid f(13) = \bot \} \cup \{ (f, 1) \mid f(13) \neq \bot \}$$



What is H_{13}?

Why is its presence a problem?

$$F13 \stackrel{df}{=} \mathrm{fix}\,(\underline{\lambda} f : \mathrm{int} \to \mathrm{int}.$$
$$(\underline{\lambda} x : \mathrm{int}.$$
$$\underline{\mathrm{if0}}\ \ \mathrm{equal}(x, 13)$$
$$f(x)$$
$$21))$$

$$N \stackrel{df}{=} \underline{\lambda} h : (\mathrm{int} \to \mathrm{int}) \to \mathrm{int}.$$
$$[\mathrm{fix}\,(\lambda g : \_\_.$$
$$\lambda f.$$
$$\underline{\mathrm{if0}}\ \ h(f)$$
$$42$$
$$g(f))$$
$$F13]$$

$$X \stackrel{df}{=} \underline{\lambda} h : \dots . \quad F13(13)$$

$$N \cong_{PCF} X$$

<u>But</u> $$[\![N]\!] \neq [\![X]\!]$$

$$(\mathbb{Z}_\perp \to \mathbb{Z}_\perp) \to \mathbb{Z}_\perp$$

<u>So</u>

$$[\![N]\!](H_{13}) = 42$$

$$[\![X]\!](H_{13}) = \perp$$

because

$$[\![F13]\!](13) = \perp$$

Solution

We exploit the ordering and allow only monotone functions in $\longrightarrow$!

Def $f : A \longrightarrow B$

f is monotone iff for all a, a' in A, $a < a'$ implies $f(a) < f(a')$

$\hookrightarrow$ Obvious if A and B are $\mathbb{Z}$

Def Otherwise two functions f, g in $[\![t]\!] \longrightarrow [\![t']\!]$ are ordered iff for all $x \in [\![t]\!]$, $f(x) \sqsubseteq g(x)$

INDUCTIVE POINTWISE

Now we can try again:

$$[\![\text{int}]\!] = \mathbb{Z}_\perp$$

$$[\![\tau \to \tau']\!] = [\![\tau]\!] \to_m [\![\tau']\!]$$

syntactic composition

semantic composition

$H_{13}(f) = 0 \quad \therefore \; f(13) = \perp$

$H_{13}(g) = 1 \quad \therefore \; g(13) \neq \perp$

$\therefore \quad f \sqsubseteq g$ or f incomparable to g

$\therefore \quad H_{13}(f)$ is incomparable to $H_{13}(g)$

$\therefore$ <u>ERGO.</u>

$$H_{13} \notin (\mathbb{Z}_\perp \to_m \mathbb{Z}_\perp) \to_m \mathbb{Z}_\perp$$

We got rid of this "trash".

Problem

What does fix denote?

$$fix : (\tau \to \tau) \to \tau$$

$$f : \tau \to \tau$$

$$fix(f) : \tau$$

Solution (from logic)

Let's use Tarski method for finding fixpoints.

$$fix(f) = \lim_{i \in \mathbb{N}} f^i(\bot)$$

Then $f(\lim f^i(\bot)) = \lim f^i(\bot)$

To use this theorem, we must find a lim operation on the mathematical space.

$$\bigcup_{i \in \mathbb{N}} f^i(\perp)$$

<u>Ex</u> how to build factorial

$\perp = \{\}$ $\boxed{\perp \text{ elsewhere}}$

$!_0 = \{(0,1)\}$

$!_1 = \{(0,1); (1,1)\}$

$!_2 = \{(0,1); (1,1); (2,2)\}$

$$! = \bigcup_{i \in \mathbb{N}} !_i$$

What we would not want:

$$
\begin{aligned}
!!_0 &= \{\} \\
!!_1 &= \{(0, \perp)\} \\
!!_2 &= \{(0, \perp); (1, 1)\} \\
!!_3 &= \{(0, \perp); (1, 1); (2, 2)\} \\
&\vdots
\end{aligned}
$$

<u>BUT</u> the $\cup$ yields $!$

It would mean a discontinuous jump at the limit.

Insights

1. We build infinite elements from

limits

of series of finite elements.

2. This construction is known from algebraic topology as

ideal completion.

3. Once we have a topology, we can insist on continuity of functions

$$[\![\tau]\!] \longrightarrow_c [\![\tau']\!]$$

4. Continuity implies mono

Types $\quad [\![\text{int}]\!] = \mathbb{Z}$

$$[\![\tau \to \tau']\!] = [\![\tau]\!] \to_c [\![\tau']\!]$$

Terms $\quad [\![n]\!] = n$

$$[\![e \cdot e']\!] = [\![e]\!] ([\![e']\!])$$

$$= \text{apply}([\![e]\!], [\![e']\!])$$

sloppy

$[\![\lambda x{:}\text{int}.\, e]\!]$ = cont. function f such that $f(x) = [\![e]\!]$

$[\![\text{fix}]\!]$ = limit operation

etc.

Technically, these spaces are known as ω-algebraic consistently-complete complete partial order.

PL calls them $\boxed{\text{Scott domains}}$

All good?

Real Problem

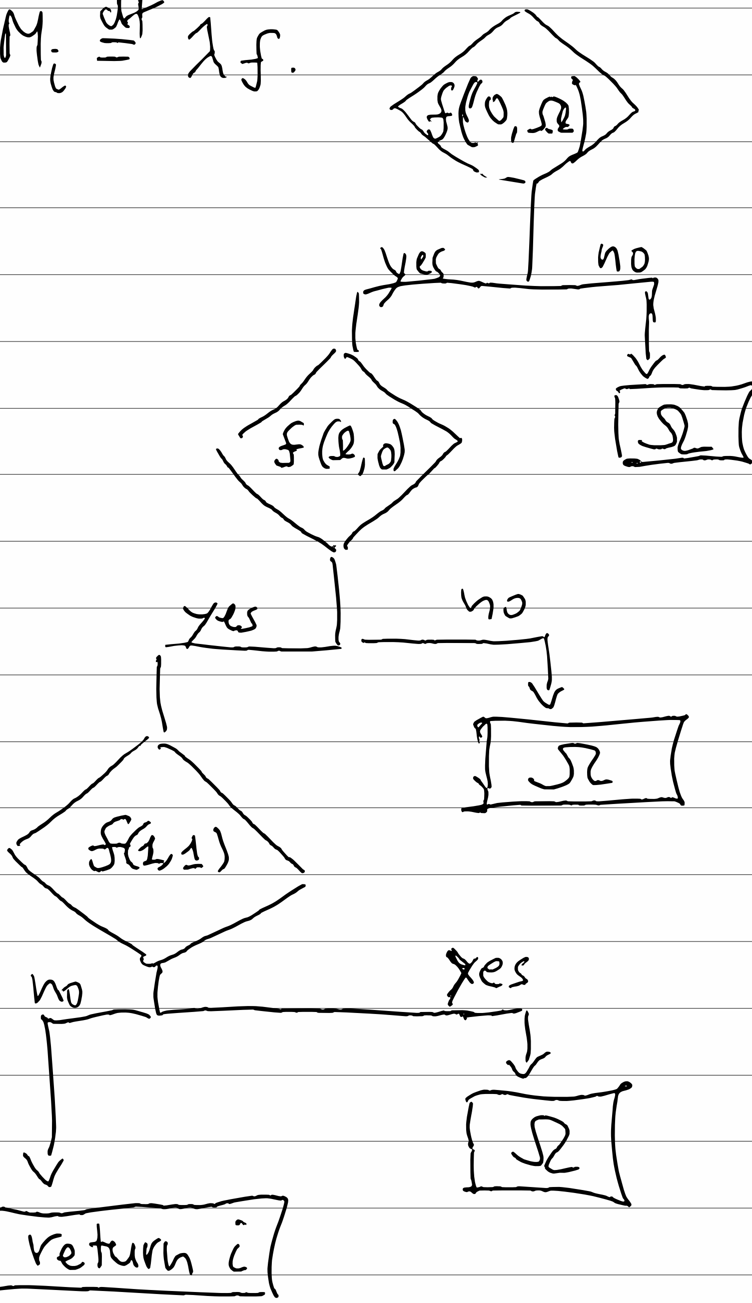
The domains we developed still don't make a good denotational semantics.

$$
\begin{aligned}
M_i \overset{\text{df}}{=} \lambda f : (\mathrm{int} \to \mathrm{int} \to \mathrm{int})\ & \\
& \mathrm{if0}\ f(0, \Omega) \\
& \quad \mathrm{if0}\ f(\Omega, 0) \\
& \qquad \mathrm{if0}\ f(1, 1) \\
& \qquad\quad i \\
& \qquad\quad \Omega \\
& \qquad \Omega \\
& \quad \Omega
\end{aligned}
$$

$$\boxed{\Omega \overset{\text{df}}{=} \mathrm{fix}(\lambda x. x)}$$

Claim $\boxed{M_0 \cong_{PCF} M_1 \cong M_2 \ldots}$

$M_i \stackrel{df}{=} \lambda f.$



Problem

$$\mathrm{por} \in \mathbb{Z}_\bot \to_c (\mathbb{Z}_\bot \to_c \mathbb{Z}_\bot)$$

$$\mathrm{por}(0, \bot) = 0$$

$$\mathrm{por}(\bot, 0) = 0$$

$$\mathrm{por}(1, 1) = 1$$

$$\mathrm{por}(x, y) = \bot$$

"por" is short for "pick a 0 in parallel from one of the two arguments".

$$[\![M_0]\!](\text{por}) = 0$$

$$[\![M_1]\!](\text{por}) = 1$$

$$\therefore$$

$$[\![M_0]\!] \neq [\![M_1]\!]$$

In sum:

The continuous Scott domains for PCF contain

(1) finite deterministic

(2) infinite deterministic

parallelism.

This makes them non-good.

Research Problem

Deterministic parallelism is basically a useless feature. PCF is sequential.

PCF

is there a good sequential denotational model w/

is there a sequential variant of PCF that has a model w/

built in a finitary manner (algebraic topology)

Milner's insight

1. We can construct a "good" model.

2. This particular model is <u>not</u> built from a finitary basis

   ARGH! No matter.

3. Any other "good" model is isomorphic to this one.

$\boxed{\text{"good" is called fully abstract.}}$

How is Milner's Model Built

Every term denotes the set of terms that are equivalent under β.

$$[\![(\lambda x{:}\tau.e)\ e']\!]$$

$$\Cap$$

$$D$$

$$\Cup$$

$$[\![e[x \leftarrow e']]\!]$$

That's a bit of a lie.

In the "olden" days, PL constructed models using

$$\begin{cases} K^t x y \mathrel{\hat{=}} x \\ S^t x y z \mathrel{\hat{=}} x(z)(y(z)) \end{cases}$$

and compiled λ-terms into $\{S, K\}$ terms.

STOP!

What's a theme?