Operational Semantics

26 Jan 2021

(1) Did we not reject operational semantics?
(2) What is a well-defined operational sem.?
(3) How do we make one for PCF?
(4) Why is there more than one λ?
(5) Is it equivalent to denotational sem.?
(6) Why are there many operational semantics for PCF?

Landin	(1)	v. 1963 – 1965
Plotkin	(2-4)	TCS '74
Plotkin	(5)	TCS '78
Plotkin	(6)	TR Aarhus '80
Kahn	(6)	TR INRIA '80

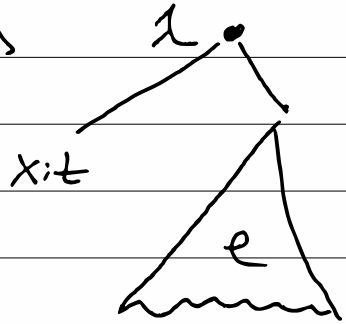
Pre-historic Operational Sem.

ISWIM ⇝ think PCF

Concrete Syntax : : lambda {x:t} e

⇓

Abstract Syntax : trees



describe w/ ISWIM
"structure" def.

↳ think Fundamentals I:

An AST (abstract syntax tree) is one of:
— (make-lambda Sym Typ AST)

⋮

Semantics: the SECD Machine

defines a partial function

$$val : PCF^{int} \longrightarrow \mathbb{Z}$$

as follows:

1. load program into machine(c)
2. run machine until it stops
3. unload resulting integer

S, E, C, D are registers:

S : local stack of values
E : environments
C : string (sequence) of ASTs
D : dump aka function call stack

all described w/ ISWIM

function calls on the SECD machine

1. $\langle S, E, (e\,e') \cdot C, D \rangle$

$\mapsto \langle S, E, e \cdot e' \cdot ap \cdot C, D \rangle$

2. $\langle S, E, (\lambda x{:}t.\, e) \cdot C, D \rangle$

$\mapsto \langle (\lambda x{:}t.\, e) \cdot S, E, C, D \rangle$

3. $\langle i \cdot (\lambda x{:}t.\, e) \cdot S, E, ap \cdot C, D \rangle$

$\longmapsto \langle S, E[x \leftarrow i], e, \langle S, E, C \rangle \cdot D \rangle$

initial states: $\langle \varepsilon, \emptyset, PCF^{int}, \varepsilon \rangle$

final states: $\langle i, \emptyset, \varepsilon, \varepsilon \rangle$

$\equiv$

1973:

Algol '60

ISWIM / PCF

λ-calculus

normal forms

head normal form

weak head normal form

normal-order vs applicative-order strategy

What is evaluation order?

What is a well-defined op sem?

1. We don't need a PL to define them.
   → Use inductively defined sets, products of such sets, etc.

2. We don't need "rules" and ordered rules to write down instructions.
   → Use relations instead.

3. We don't need a PL to define 'eval'.
   → Use transitive closures instead.
   Use algebraic closures in general.

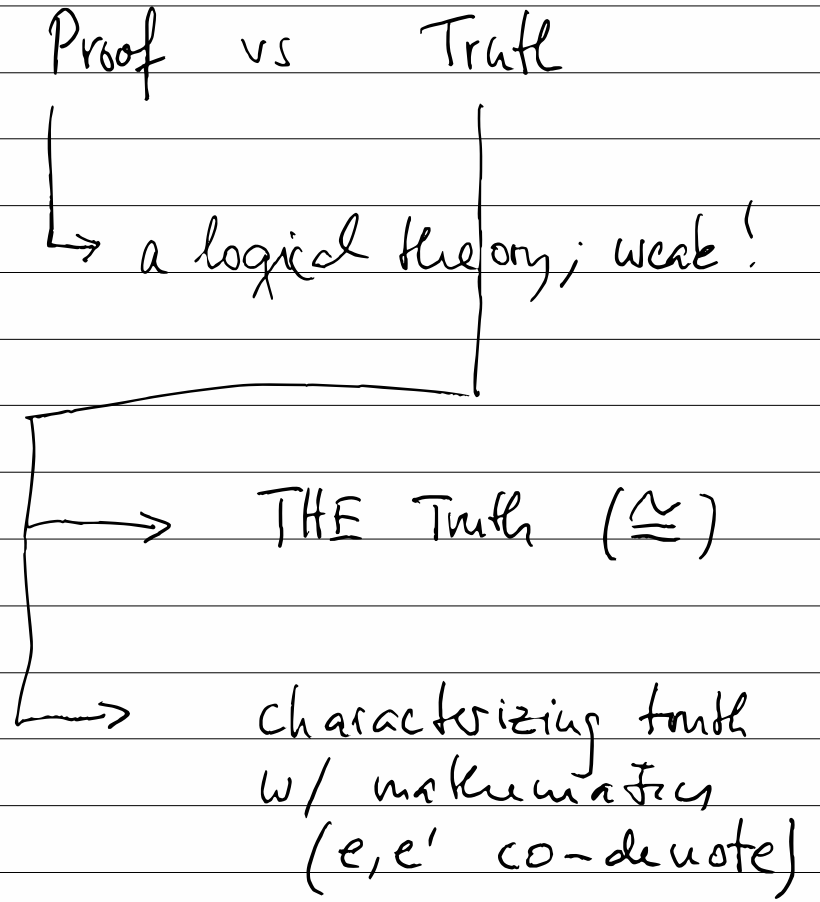
⇝ Use simple discrete math.

How to make a well-defined operational semantics for languages based on λ?

→ Does it have to be an ad hoc as the [EC] -machine?

And (why) is there more than one λ-calculus?

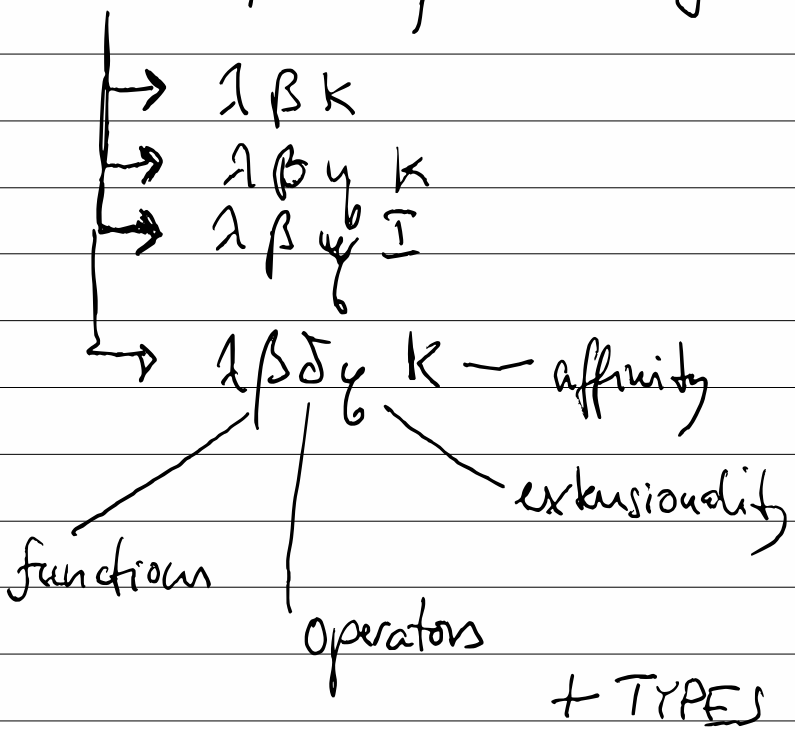
→ How do we reason about observational equivalence using operational semantics?


Proof vs Truth
a logical theory; weak!
THE Truth (≅)
characterizing truth
w/ mathematics
(e, e' co-denote)

Let's start w/ the λ-CALCULUS

People use λ-syntax and say λ-calculus. ARGH.

Church really made a calculus meaning an equational logic.



λ-calculus for PCF / ISWIM

$e = x \mid \lambda x.e \mid ee \mid n \mid add1\ e$

Syntax : solve this math. set equation:

$e = X \uplus \lambda x.e \uplus ee \uplus \dots$

$e = (0 \times Vars) \cup (1 \times Vars \times e) \cup (2 \times e \times e) \dots$

$Vars =$ inf. set of distinct abs. "things"

$\therefore$ equations in sets
unions
products
least fixpoints (induction)

STANDARD!

$$e = x \mid \lambda x.e \mid ee \mid n \mid add1\ e$$

Equations An equation is a pair of e.

A system of equality is a binary relation on e that is
- reflexive
- transitive
- symmetric
- congruent *

It can be generated from a binary notion of reduction via algebraic closure operations.

$$e = x \mid \lambda x.e \mid ee \mid n \mid \text{add1}\ e$$

Notions of reduction

$$a = \{ (\text{add1}\ n, k) \mid n \in \mathbb{Z}, k = n+1 \}$$

$$b = \{ ((\lambda x.e)n, e[x \leftarrow n]) \mid n \in \mathbb{Z} \}$$

$$c = \{ (\lambda x.ex, e) \}$$

$$\vdots$$

usually we write

$$\text{add1}\ n \ \underline{a}\ n+1$$

$$(\lambda x.e)\, n \ \underline{b}\ e[x \leftarrow n]$$

$$\lambda x.ex \ \underline{c}\ e$$

$e = x \mid \lambda x.e \mid ee \mid n \mid \text{add1}\ e$

add1 (add1 0) $\underline{a}$ 2

NO ↯

Why not?

$e = x \mid \lambda x.e \mid ee \mid n \mid \text{add1}\ e$

Compatible closure:

$e \rightarrow_a e'$ iff $e \underline{a} e'$

or $e = \lambda x.\, e_0$, $e' = \lambda x.\, e_1$,
and $e_0 \rightarrow_a e_1$

or $e = e_0 e_1$, $e' = e_2 e_3$,
and $e_0 \twoheadrightarrow_a e_2$
or $e_1 \rightarrow_a e_3$

⋮ one clause per syntactic construction in e

inductive definition of $\rightarrow_a$

$e = x \mid \lambda x.e \mid ee \mid n \mid \text{add1}\ e$

$e \twoheadrightarrow_a e'$: transitive, reflexive closure of $\to_a$

$e =_a e'$: trans., refl., sym. closure of $=_a$

NOW we can "calculate" with $=_a$ at any sub-expression position of any expression e.

This much logicians knew in 1974.

!

$e = x \mid \lambda x.e \mid ee \mid n \mid \text{add1}\ e$

Plotkin: 1. $\lambda x.e$ and n are values

2. There are two notions of reductions

$(\lambda x.e)\, e' \ \beta_n \ e[x \leftarrow e']$

$(\lambda x.e)\, v \ \beta_v \ e[x \leftarrow v]$

↑ value

3. Evaluators do not reduce to normal form but values

eval : $e \longrightarrow_p v$

Plotkin: The rest is a bunch of theorems that use standard proof techniques.

Def For $x = n$ or $x = v$,

$$eval_x(e) = \begin{cases} n & \text{if } e =_x n \\ \text{"closure"} & \text{if } e =_x \lambda \cdots \end{cases}$$

Thm $eval_x$ is a function.

Proof Use Church Rosser.

Thm If $e =_x e'$ then $e \cong_x e'$

Def. $\longmapsto_x$ use x at leftmost-outermost position in expression e.

Thm $eval_x$ is trans.-ref. closure of $\longmapsto_x$

Proof Use Curry-Feys.

N.B. Plotkin then investigates the relationship between $=_n$ and $=_v$ and $\cong_n$ and $\cong_v$ respectively.

And they are non-trivial.

Felleisen LM/OM ⇝ evaluation contexts

⇝ x can be continuations
⇝ x can be assignments
⇝ x can be threads
⇝ x can be lazy (Haskell

$$E[(\lambda x.e)\ v] \longmapsto_v E[e[x \leftarrow v]]$$

$$E[(\lambda x.e)\ e'] \longmapsto_n E[e[x \leftarrow e']]$$

or even

$$E[\text{call/cc}\ e] \longmapsto_c E[e\ E]$$

Summary

1. Build a calculus from relation $\times$ on syntax in a systematic + uniform manner.

2. Every step uses plain set th. and algebraic techniques.

3. The "machine" ($\longmapsto_{\times}$) needs nothing but syntax.

4. Theorems can leverage classical techniques from the 1930s – 1950s, plus some tricks.

Is this operational semantics equivalent/related to denotational semantics?

or

$[\![e]\!]_n$ vs $eval_n(e)$

and similarly for call-by-value

Thm For $e : int$ and $k \in \mathbb{Z}$

$[\![e]\!]_n = k$ iff $eval(e) = k$

Proof By structural induction on e, but with the stronger IH:

$$\mathrm{Comp}^{\mathrm{int}} = \{ e \mid e:\mathrm{int} \text{ and } \mathrm{val}(e) = k \text{ implies } [\![e]\!] = k \}$$

$$\mathrm{Comp}^{t \to t'} = \{ e \mid e \text{ closed at } t \to t' \text{ and for all closed } e' \text{ in } \mathrm{Comp}^{t},\ ee' \in \mathrm{Comp}^{t'} \}$$

$$\mathrm{Comp}^{t} = \{ e \mid e \text{ has free variables } \vec{x} \text{ and for all closed } \vec{e'} \text{ in } \overrightarrow{\mathrm{Comp}},\ e[\vec{x} \leftarrow \vec{e'}] \in \mathrm{Comp}^{t} \}$$

But how does *fix* work?

$$fix^0 = \Omega \qquad \text{(inf. loop)}$$

$$fix^{n+1} = \lambda f : t \to t.\ f\,(fix^n\, f)$$

step-indexed term

then

$$[\![\, fix \,]\!]\,(\bot) = \bigsqcup_{n \in \mathbb{N}} [\![\, fix^n \,]\!]\,(\bot)$$

NOTE For a language w/o types, the construction of the entire domain is indexed. [Sitaram 94]

Why are there so many forms of operational sem.?

1980s thinking

1. Programmers wish to prove theorems about individual programs.

2. They need semantics for them.

3. Axiomatic semantics works for toy PL's i.e. =, if, while.

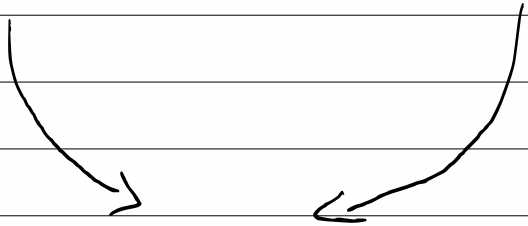
4. Denotational supports more & allows structural ind. due to compositionality.

5. <u>But</u> The math knowledge is high

Now that we "get" operational semantics as sets, relations, and inductive definitions, let's do it.

Milner (Edinburgh) Plotkin

Kahn (Sophia-Antip.)



mimic denotational "tricks" in operational semantics, but remain structural

Big Step

$e = x \mid \lambda x.e \mid e\,e \mid n \mid \mathrm{add1}\ e$

Closures & Environments

$\rho \overset{!}{=} x \xrightarrow[f]{} cl = \{(x_0, cl_0), \ldots, (x_n, cl_n)\}$

$cl \overset{!}{=} n \mid \langle \lambda x.e, \rho \rangle$

Why is this okay?

judgment: $\rho \vdash e \Downarrow cl$

pronounce "ρ proves that e evaluates to cl"

$$\rho \vdash n \Downarrow n \qquad \rho \vdash \lambda x.e \Downarrow (\lambda x.e, \rho)$$

$$\frac{\rho \vdash e \Downarrow n}{\rho \vdash \mathrm{add1}\, e \Downarrow n+1}$$

if ρ proves e yields n, then ρ proves add1 e evaluates to $n+1$

$$\frac{\rho^* \vdash e_0 \Downarrow cl_0 \quad \rho^* \vdash e_1 \Downarrow cl_1 \quad cl_0 \equiv \langle \lambda x.e, \rho \rangle \quad \rho[x \leftarrow cl_1] \vdash e \Downarrow cl}{\rho^* \vdash e_0\, e_1 \Downarrow cl}$$

$$\frac{\rho \ni (x, cl)}{\rho \vdash x \Downarrow cl}$$

Plotkin's Small Step

$$\frac{\rho \vdash e_0 \longrightarrow e_0'}{\rho \vdash e_0\, e_1 \longrightarrow e_0'\, e_1}$$

$$\frac{\rho \vdash e_1 \longrightarrow e_1'}{\rho \vdash (\lambda x.e)\, e_1 \longrightarrow (\lambda x.e)\, e_1'}$$

$$\rho \vdash (\lambda x.e)\, v \longrightarrow \underline{\text{let}}\ x = v\ \underline{\text{in}}\ e$$

"BEFORE DISCUSSING OUR SPECIFIC PROPOSAL WE SHOULD ADMIT THAT THIS SEEMS ... TO BE A POSSIBLE WEAK POINT IN OUR TREATMENT."

p. 125

What about errors?

$$\frac{\rho^* \vdash e_0 \Downarrow n, \quad \rho^* \vdash e_1 \Downarrow cl_1}{\rho^* \vdash e_0\, e_1 \Downarrow \text{error}}$$

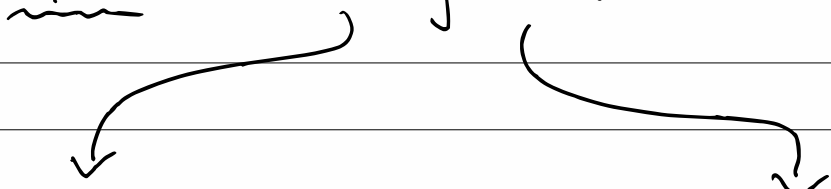
∴ explosion of rules

$$\frac{\rho^* \vdash e_0 \Downarrow \text{error}}{\rho^* \vdash e_0\, e_1 \Downarrow \text{error}}$$

$$\frac{\rho^* \vdash e_0 \Downarrow cl_0 \quad \rho^* \vdash e_1 \Downarrow \text{error}}{\rho^* \vdash e_0\, e_1 \Downarrow \text{error}}$$

and so on, for every syntactic construct.

1980s PL general



"PL" deals with all programs in a language

"FM" deals w/ individual programs

type soundness becomes the interesting question

1) den. sem. fails
2) small step is stuck
3) big step needs co-ind.
4) red. sem. just works (post 1988)

2000s

P2 verifies compilers (w/ PAs).

A compiler is a program.

(Even though it is close to eval.)

For an individual program,
structural induction
is a good technique.

Interesting (big step) operational
semantics reasoning.