Programming Language Semantics

$\{$Ret$|$Int$|$P$\}$rospective Discussion

Ohad Kammar
<ohad.kammar@ed.ac.uk>

LFCS Lab Lunch
September 20, 2011

Strategic Planning

Semantics

Semantics = Meaning

Semantics

Semantics = Meaning
of
Programming Languages

Semantic Goals

How to give semantics?

What to do with semantics?

Relate different semantics.

Outline

- Operational Semantics
  - How?
  - What?
  - Relating semantics.
  - Selected History.
- Denotational Semantics
  - How?
  - What?
  - Relating semantics.
  - Very Selected History.
- Introspective and Prospective Discussion.

Build machines!

Meaning = machine behaviour

Example

`gcc v.4.5.2` running on Ubuntu 11.04 on a 32 bit x86 Intel...

In general:

$$\langle \mathrm{Prog}, \mathrm{Conf} \rangle \rightarrow \left\langle \mathrm{Prog}', \mathrm{Conf}' \right\rangle \rightarrow \ldots$$

Portability

Aim for abstract meaning: cleaner, reason-able

Observable Behaviour

$$\langle P, C\rangle \xrightarrow{\texttt{print("hell")}} \langle P', C'\rangle \to \ldots$$

Non-Determinism/Randomness

$$\begin{array}{ccl} & & \langle P', C'\rangle \to \ldots \\ & \frac{1}{3}\nearrow & \\ \langle P, C\rangle & & \\ & \searrow & \\ & \frac{2}{3} & \langle P'', C''\rangle \to \ldots \end{array}$$

Crashes

$$\langle P, C\rangle \lightning \texttt{NullPointerException}$$

Structural Operational Semantics

- Abstract syntax manipulation
- Syntax-directed rules

$$P ::= \mathtt{x} \leftarrow \mathtt{i} \mid P_1; P_2 \mid \mathtt{nop}$$

$$\frac{}{\langle \mathtt{x} \leftarrow \mathtt{i}, C\rangle \to \langle \mathtt{nop}, C[\mathtt{x} \mapsto \mathtt{i}]\rangle} \qquad \frac{}{\langle \mathtt{nop}; P, C\rangle \to \langle P, C\rangle}$$

$$\frac{\langle P_1, C\rangle \to \langle P_1', C'\rangle}{\langle P_1; P_2, C\rangle \to \langle P_1'; P_2, C'\rangle}$$

Simple

$P_1 \sim P_2 \iff$ they end with the same configurations:

$$\begin{array}{l} \texttt{temp = x;} \\ \texttt{x = y;} \\ \texttt{y = temp} \end{array} \quad \sim \quad \begin{array}{l} \texttt{x = x XOR y;} \\ \texttt{y = x XOR y;} \\ \texttt{x = x XOR y} \end{array}$$

Simple

$P_1 \sim P_2 \iff$ they end with the same configurations:

$$\begin{array}{lcl} \texttt{temp = x;} & & \texttt{x = x XOR y;} \\ \texttt{x = y;} & \sim & \texttt{y = x XOR y;} \\ \texttt{y = temp;} & & \texttt{x = x XOR y;} \\ \texttt{temp = 0} & & \texttt{temp = 0} \end{array}$$

Simple

$P_1 \sim P_2 \iff$ they end with the same configurations:

$$\begin{array}{lcl} \texttt{temp = x;} & & \texttt{x = x XOR y;} \\ \texttt{x = y;} & \sim & \texttt{y = x XOR y;} \\ \texttt{y = temp;} & & \texttt{x = x XOR y;} \\ \texttt{temp = 0} & & \texttt{temp = 0} \end{array}$$

Useful (Contextual Equivalence)

$P_1 \cong P_2 \iff$ for all wrapping contexts $\mathcal{C}[-]$: $\mathcal{C}[P_1] \sim \mathcal{C}[P_2]$

Example

$\mathcal{C}[-]$: run `temp = 1` in parallel to $-$

Safety

Safe to run in specialised environments (GPU, DB, browser).

(Bi)simulation

```
while(true)                 while(true)
  x = 2 + x;      vs.         x = 1 + x;
  print(x)                    print(2 * x)
```

$$S_0 \to S_1 \to S_2 \xrightarrow{\texttt{print(2)}} S_0 \to S_1 \to S_2 \xrightarrow{\texttt{print(4)}} S_0 \to \dots$$

$$S'_0 \to S'_1 \to S'_2 \xrightarrow{\texttt{print(2)}} S'_0 \to S'_1 \to S'_2 \xrightarrow{\texttt{print(4)}} S'_0 \to \dots$$

Why?

- Implement abstract machines with concrete ones.
- Transfer properties between semantics.

Some Tools

- (Bi)simulation.
- Logical relations.

Pros

Intuitive, elementary $\Longrightarrow$ ubiquitous in PL community.
Mechanised support.

Cons

"Chaotic", ad-hoc.

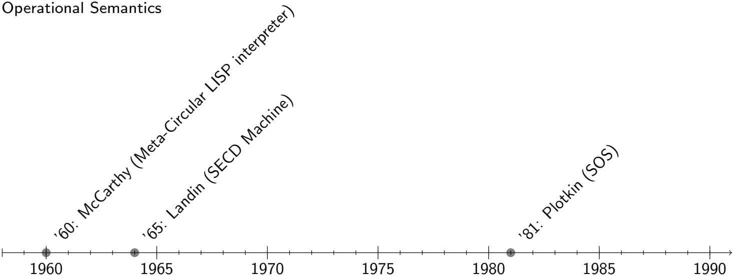
Sample Applications

E.g., Sewell et al.:

- *Mathematizing C++ Concurrency*, POPL'11.
- *The Semantics of x86-CC Multiprocessor Machine Code*, POPL'09.

Selected History

Operational Semantics



Code to Math

$$[\![\mathtt{0xBEEF}]\!] = 48{,}879$$

Compositionality

$$[\![\mathtt{x} + \mathtt{1} >> \mathtt{2} * \mathtt{i}]\!] = \left\lfloor \frac{[\![\mathtt{x} + \mathtt{1}]\!]}{2^{[\![\mathtt{2} * \mathtt{i}]\!]}} \right\rfloor$$

Serious Math

Domain Theory, Topology, Analysis, Logic, Abstract Algebra, Category Theory, ...

Equivalence

Easy! $P_1 \equiv P_2 \iff [\![P_1]\!] = [\![P_2]\!]$
Compositional by construction.

Soundness

Denotations guarantee *some* sense.

Design

New paradigms expose semantic structure.

- Monads!
- GUI Languages: *A Semantic Model for Graphical User Interfaces*, Benton et al., ICFP'11.
- Effect handlers and Eff (Plotkin and Pretnar ESOP'09, Pretnar and Bauer '11).

Tools

- Denotational to denotational: logical relations.
- Denotational to operational: adequacy and full abstraction.

Pros

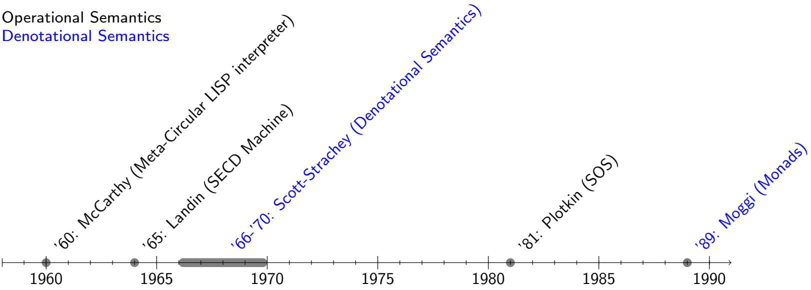
Stable, global (=big picture) $\implies$ powerful tool, when applicable.
Domain Specific Languages?

Cons

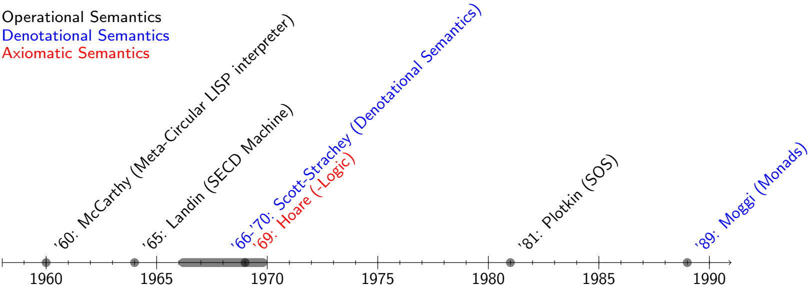
Non-elementary, slower development.

Selected History

Operational Semantics
Denotational Semantics



Selected History



SOS vs. Denotational Semantics

Denotational semantics isn't as commonly found as SOS.

Are they less successful? Why?

Introspection

Adaptation

Semantics in the Dragon Book (2007):

> *With the notations currently available, the semantics of a language is much more difficult to describe than the syntax. For specifying semantics, we shall therefore use informal descriptions and suggestive examples.*

The organisations and people who should benefit the most from semantics are not using it. Why?

Introspection

Difference

How could PL designers and implementers ignore semantics for so long? What is the difference between *semantics* and complexity or automata theory?

Prospects

So what should we do?