Programming Language Semantics

Prospective Discussion

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

LFCS Lab Lunch
September 20, 2011
Semantics = Meaning
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.
Operational Semantics

How?

Build machines!
 Meaning = machine behaviour

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

In general:

\[ \langle \text{Prog, Conf}\rangle \rightarrow \langle \text{Prog'}, \text{Conf'}\rangle \rightarrow \ldots \]

Portability
Aim for abstract meaning: cleaner, reason-able
Operational Semantics

Variations

Observable Behaviour

\[ \langle P, C \rangle \xrightarrow{\text{print("hell")}} \langle P', C' \rangle \rightarrow \ldots \]

Non-Determinism/Randomness

\[ \langle P', C' \rangle \rightarrow \ldots \]

\[ \frac{1}{3} \]

\[ \langle P, C \rangle \]

\[ \frac{2}{3} \]

Crashes

\[ \langle P, C \rangle \nleftarrow \text{NullPointerException} \]

\[ \langle P'', C'' \rangle \rightarrow \ldots \]
Structural Operational Semantics

- Abstract syntax manipulation
- Syntax-directed rules

\[ P ::= x \leftarrow i | P_1; P_2 | \text{nop} \]

\[ \langle x \leftarrow i, C \rangle \rightarrow \langle \text{nop}, C[x \leftarrow i] \rangle \quad \langle \text{nop}; P, C \rangle \rightarrow \langle P, C \rangle \]

\[ \langle P_1, C \rangle \rightarrow \langle P'_1, C' \rangle \rightarrow \langle P_1; P_2, C \rangle \rightarrow \langle P'_1; P_2, C' \rangle \]
Operational Semantics

What? Equivalence!

Simple

\[ P_1 \sim P_2 \iff \text{they end with the same configurations:} \]

\[
\begin{align*}
\text{temp} &= x; & x &= x \text{ XOR } y; \\
 x &= y; & y &= x \text{ XOR } y; \\
 y &= \text{temp} & x &= x \text{ XOR } y
\end{align*}
\]
Simple

\[ P_1 \sim P_2 \iff \text{they end with the same configurations:} \]

\[
\begin{align*}
\text{temp} &= x; & \quad x &= x \text{ XOR } y; \\
\text{x} &= y; & \quad y &= x \text{ XOR } y; \\
\text{y} &= \text{temp}; & \quad x &= x \text{ XOR } y; \\
\text{temp} &= 0 & \quad \text{temp} &= 0
\end{align*}
\]
Operational Semantics

What? Equivalence!

Simple

\[ P_1 \sim P_2 \iff \text{they end with the same configurations:} \]

\[
\begin{align*}
\text{temp} &= x; & x &= x \text{ XOR } y; \\
x &= y; & y &= x \text{ XOR } y; \\
y &= \text{temp}; & x &= x \text{ XOR } y; \\
\text{temp} &= 0 & \text{temp} &= 0
\end{align*}
\]

Useful (Contextual Equivalence)

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

Example

\[ C[-]: \text{run } \text{temp} = 1 \text{ in parallel to } - \]
Safety
Safe to run in specialised environments (GPU, DB, browser).

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

\[
\begin{align*}
S_0 &\rightarrow S_1 \rightarrow S_2 \xrightarrow{\text{print}(2)} S_0 \rightarrow S_1 \rightarrow S_2 \xrightarrow{\text{print}(4)} S_0 \rightarrow \ldots \\
S'_0 &\rightarrow S'_1 \rightarrow S'_2 \xrightarrow{\text{print}(2)} S'_0 \rightarrow S'_1 \rightarrow S'_2 \xrightarrow{\text{print}(4)} S'_0 \rightarrow \ldots
\end{align*}
\]
Why?

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

Some Tools

- (Bi)simulation.
- Logical relations.
Pros
Intuitive, elementary \(\rightarrow\) ubiquitous in PL community.
Mechanised support.

Cons
“Chaotic”, ad-hoc.

Sample Applications
E.g., Sewell et al.:
- *Mathematizing C++ Concurrency*, POPL’11.
Selected History

Operational Semantics

'60: McCarthy (Meta-Circular LISP interpreter)

'65: Landin (SECD Machine)

'66-'70: Scott-Strachey (Denotational Semantics)

'69: Hoare (-Logic)

'81: Plotkin (SOS)

'89: Moggi (Monads)
Denotational Semantics

How?

Code to Math

\[ [0x\text{BEEF}] = 48,879 \]

Compositionality

\[ [\text{x + 1} \gg 2 \times i] = \left[ \frac{[\text{x + 1}]}{2^{\lfloor 2 \times i \rfloor}} \right] \]

Serious Math

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

What?

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.
Denotational Semantics

Overall

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

Cons
Non-elementary, slower development.
Selected History

Operational Semantics
Denotational Semantics

'60: McCarthy (Meta-Circular LISP interpreter)
'65: Landin (SECD Machine)
'66-'70: Scott-Strachey (Denotational Semantics)
'69: Hoare (-Logic)
'81: Plotkin (SOS)
'89: Moggi (Monads)
Selected History

Operational Semantics
Denotational Semantics
Axiomatic Semantics

'60: McCarthy (Meta-Circular LISP interpreter)
'65: Landin (SECD Machine)
'66-'70: Scott-Strachey (Denotational Semantics)
'69: Hoare (-Logic)
'81: Plotkin (SOS)
'89: Moggi (Monads)
**SOS vs. Denotational Semantics**

Denotational semantics isn’t as commonly found as SOS. Are they less successful? Why?
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?
So what should we do?