Introduction to Semantics

February 13, 2001
CS 131: Programming Languages

Semantics

• To understand a programming language, not enough to know its syntax.
• The semantics of a language specifies the meaning of a program
  - What program phrases mean when put together.
  - What answer does each program produce?
  - How should execution proceed?
• The large majority of the work in defining a language is specifying the semantics.

Purposes of a Language Definition

• For the programmer
  - Understanding the language
  - Reasoning about programs
• For the language implementor
  - Understanding what correct implementations must/may do
  - Deciding whether program transformations are correct
  - Facilitate multiple (compatible) implementations
• For the language designer
  - Recording design decisions
  - Understanding interaction between language features
  - Reasoning about the language

Formal Definitions?

• Why a formal semantics?
  - Informal definitions invariably contain ambiguities or errors.
  - Facilitates reasoning about the language
  - Facilitates reasoning about programs in the language
  - Facilitates reasoning about program transformations
  - May permit automatic generation of implementations
• Truth in advertising: very hard to give a formal description of a full, real language
  - But can handle quite large subsets
  - Active research topic
Two Approaches to Formal Semantics

• Denotational semantics
  - The meaning of every program phrase is a mathematical object (a number, a function, a pair, a sequence, etc.)
  - *Compositionality*: meaning of an expression is a function of the meanings of its sub-expressions.
    • E.g., the meaning of the loop *while* *b* *do* *c* is calculated from the meanings of the guard expression *b* and of the loop body *c*
    • Two expressions with the same meaning are interchangeable.

• Operational semantics
  - Explains execution of complete programs
    • Formal specification of an interpreter
  - We can choose the level of abstraction
    • Which (if any) low-level machine details we want to describe
      - Data representations
      - Memory management
    • Which concepts considered primitive

"Big-Step" Operational Semantics

• Also known as "natural semantics".
• Definition of an "evaluates to" relation.
  
  \[
  \text{exp} \Downarrow \text{value}
  \]

• Example: \((3+4)+2 \Downarrow 9\)

Recall: Abstract Syntax

\[
\text{exp ::= num \hspace{1cm} integers}
\]

\[
\begin{array}{l}
\text{bool \hspace{1cm} booleans} \\
\text{var \hspace{1cm} variables} \\
(\text{var}) \Rightarrow \text{exp \hspace{1cm} functions} \\
\text{exp \hspace{1cm} applications} \\
\text{exp + exp \hspace{1cm} additions} \\
\text{exp == exp \hspace{1cm} equality-test} \\
\text{exp ? exp : exp \hspace{1cm} conditionals} \\
\text{<exp,exp> \hspace{1cm} pairs} \\
\text{fst exp \hspace{1cm} 1st projection} \\
\text{snd exp \hspace{1cm} 2nd projection} \\
\text{let var be exp in exp \hspace{1cm} local definitions}
\end{array}
\]
Recall: Values

\[
\text{value ::= num} \quad \text{integers}
\]
\[
\text{bool} \quad \text{booleans}
\]
\[
(var) \Rightarrow \text{exp} \quad \text{functions}
\]
\[
<\text{value,value}> \quad \text{pairs}
\]

\[
\text{exp} \Downarrow \text{value}
\]

Defining the Semantics

• An operational semantics is usually specified as a deductive system of inference rules.
• Recall the form of these rules:

\[
\text{...premises...}
\]
\[
\text{...conclusion...}
\]

Arithmetic

• An axiom scheme:

\[
\text{value} \Downarrow \text{value}
\]

• A rule of inference with two premise:

\[
\text{exp}_1 \Downarrow m \quad \text{exp}_2 \Downarrow n
\]
\[
\frac{}{\text{exp}_1 + \text{exp}_2 \Downarrow m+n}
\]

Exercise

• Give the complete proof tree showing that

\[
(3+4)+2 \Downarrow 9
\]
Rules for Equality and Conditional

\[
\begin{aligned}
\text{if } & \text{ exp }_1 \equiv \text{ exp }_2 \text{ then } \text{ exp }_3 \text{ else } \text{ exp }_4 \\
\text{then } & \text{ if } \text{ exp }_1 \equiv \text{ exp }_2 \text{ then } \text{ exp }_3 \text{ else } \text{ exp }_4 \\
\text{else } & \text{ if } \text{ exp }_1 \equiv \text{ exp }_2 \text{ then } \text{ exp }_3 \text{ else } \text{ exp }_4
\end{aligned}
\]

Exercise

- Show the proof tree for evaluation of

\[
\text{if } (3=4) \text{ then } (5+6) \text{ else } (7+8)
\]

Exercise

- Complete the following three inference rules:

\[
\begin{aligned}
\text{let } & \text{ var } \text{ be } \text{ exp }_1 \text{ in } \text{ exp }_2 \\
\text{let } & \text{ var } \text{ be } \text{ exp }_1 \text{ in } \text{ exp }_2
\end{aligned}
\]

Local Definitions

- Local definitions can be described using substitution as seen in last class:

\[
\begin{aligned}
\text{let } & \text{ var } \text{ be } \text{ exp }_1 \text{ in } \text{ exp }_2 \\
\text{let } & \text{ var } \text{ be } \text{ exp }_1 \text{ in } \text{ exp }_2
\end{aligned}
\]

- Example instance of this rule:

\[
\begin{aligned}
1+1 & \downarrow 2 \\
2+2 & \downarrow 4 \\
\text{let } x \text{ be } 1+1 \text{ in } x+x & \downarrow 4
\end{aligned}
\]
### Function Application

\[
\begin{align*}
\text{exp}_1 \Downarrow ((\text{var}) \Rightarrow \text{exp}_3) \\
\text{exp}_2 \Downarrow \text{value}_2 \\
\text{exp}_3[\text{var} \leftarrow \text{value}_2] \Downarrow \text{value}_3 \\
\frac{2 \Downarrow 3}{5 \Downarrow 1} \\
((x) \Rightarrow x+1) \Downarrow ((x) \Rightarrow x+1) \\
2 + 3 \Downarrow 5 \\
(2+3) \Downarrow 6
\end{align*}
\]

3 premises!

### Reading Off an Interpreter

exception Error

fun eval expr =
  (case expr of
   Num n => Num n
  | Bool b => Bool b
  | FnVal(x,e) => FnVal(x,e)
  | Var _ => raise Error

### Addition

<table>
<thead>
<tr>
<th>Plus(e1,e2) =&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>let</td>
</tr>
<tr>
<td>val v1 = eval e1</td>
</tr>
<tr>
<td>val v2 = eval e2</td>
</tr>
<tr>
<td>in</td>
</tr>
<tr>
<td>(case (v1,v2) of</td>
</tr>
<tr>
<td>(Num m, Num n) =&gt; Num (m+n)</td>
</tr>
<tr>
<td>_ =&gt; raise Error)</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

### Equality Test

<table>
<thead>
<tr>
<th>Equal(e1,e2) =&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>let</td>
</tr>
<tr>
<td>val v1 = eval e1</td>
</tr>
<tr>
<td>val v2 = eval e2</td>
</tr>
<tr>
<td>in</td>
</tr>
<tr>
<td>(case (v1,v2) of</td>
</tr>
<tr>
<td>(Num m1, Num m2) =&gt; Bool(m1=m2)</td>
</tr>
<tr>
<td>_ =&gt; raise Error)</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>
Conditional
\[ \text{Cond}(e_1, e_2, e_3) \Rightarrow \begin{cases} \text{true} & \text{eval } e_1 = \text{true} \Rightarrow \text{eval } e_2 \\ \text{false} & \text{eval } e_1 = \text{false} \Rightarrow \text{eval } e_3 \\ \_ & \Rightarrow \text{raise Error} \end{cases} \]

Local Definitions
\[ \text{Let}(x, \text{exp}_1, \text{exp}_2) \Rightarrow \begin{cases} \text{value}_1 & \text{eval } \text{exp}_1 = \text{value}_1 \\ \text{value}_2 & \text{eval } (\text{subst}(\text{exp}_2, x, \text{value}_1)) = \text{value}_2 \end{cases} \]

Application
\[ \text{Apply}(e_1, e_2) \Rightarrow \begin{cases} (\text{var} \Rightarrow \text{exp}_3) & \text{exp}_1 \Downarrow \text{value} \\ \text{value} & \text{exp}_2 \Downarrow \text{value} \\ \text{value}_3 & \text{exp}_1 \text{exp}_2 \Downarrow \text{value}_3 \end{cases} \]

Pairs and Projections
\[ \text{Pair}(e_1, e_2) \Rightarrow \text{Pair}(\text{eval } e_1, \text{eval } e_2) \]
\[ \text{Fst } M \Rightarrow \begin{cases} (\text{eval } M) \Rightarrow (\text{eval } M) \\ \_ \Rightarrow \text{raise Error} \end{cases} \]
\[ \text{Snd } M \Rightarrow \begin{cases} (\text{eval } M) \Rightarrow (\text{eval } M) \\ \_ \Rightarrow \text{raise Error} \end{cases} \]