CSE 505: Concepts of Programming Languages

Dan Grossman
Fall 2005
Lecture 3—Operational Semantics for IMP
Where we are

• Done: IMP syntax, structural induction, Caml basics
• Today: IMP operational semantics
• Tonight: You could finish homework 1
Review

IMP’s abstract syntax is defined inductively:

\[
s ::= \text{skip} | x := e | s; s | \text{if } e \text{ s s} | \text{while e s}
\]

\[
e ::= c | x | e + e | e * e
\]

\[
(c \in \{\ldots, -2, -1, 0, 1, 2, \ldots\})
\]

\[
(x \in \{x_1, x_2, \ldots, y_1, y_2, \ldots, z_1, z_2, \ldots, \ldots\})
\]

We haven’t said what programs mean yet! (Syntax is boring)

But we have a social understanding about variables and control flow
Expression semantics

\[ H ::= \cdot \mid H, x \mapsto c \]

<table>
<thead>
<tr>
<th></th>
<th>CONST</th>
<th>VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ H ; e \Downarrow c ]</td>
<td>[ H ; c \Downarrow c ]</td>
<td>[ H ; x \Downarrow H(x) ]</td>
</tr>
</tbody>
</table>

\begin{align*}
\text{ADD} & \\
H ; e_1 \Downarrow c_1 & \quad H ; e_2 \Downarrow c_2 & \quad H ; e_1 + e_2 \Downarrow c_1 + c_2 \\
\text{MULT} & \\
H ; e_1 \Downarrow c_1 & \quad H ; e_2 \Downarrow c_2 & \quad H ; e_1 * e_2 \Downarrow c_1 * c_2 \\
\end{align*}

“pronounce” as proofs (upward) or evaluations (downward)
Expression semantics cont’d

\[
H(x) = \begin{cases} 
    c & \text{if } H = H', x \mapsto c \\
    H'(x) & \text{if } H = H', y \mapsto c' \\
    0 & \text{if } H = \cdot
\end{cases}
\]

Last case avoids “errors” (makes function total)

We have rule schemas (“rules”). We instantiate a rule by replacing metavariables appropriately.
Instantiating rules

Example instantiation:

\[ \cdot, y \mapsto 4 ; 3 + y \Downarrow 7 \quad \cdot, y \mapsto 4 ; 5 \Downarrow 5 \]

\[ \cdot, y \mapsto 4 ; (3 + y) + 5 \Downarrow 12 \]

Instantiates:

\[ H ; e_1 \Downarrow c_1 \quad H ; e_2 \Downarrow c_2 \]

\[ H ; e_1 + e_2 \Downarrow c_1 + c_2 \]

with \( H = \cdot, y \mapsto 4, e_1 = (3 + y), c_1 = 7, e_2 = 5, c_2 = 5 \)
Derivations

A *(complete)* derivation is a tree of instantiations with *axioms* at the leaves.

Example:

\[
\begin{array}{c}
\cdot, \ y \mapsto 4 \ ; \ 3 \downarrow 3 \\
\cdot, \ y \mapsto 4 \ ; \ 5 \downarrow 5 \\
\cdot, \ y \mapsto 4 \ ; \ (3 + y) + 5 \downarrow 12
\end{array}
\]

So \( H \ ; \ e \downarrow c \) if there exists a derivation with \( H \ ; \ e \downarrow c \) at the root.
Some theorems

• Progress: For all $H$ and $e$, there exists a $c$ such that $H ; e \downarrow c$.

• Determinacy: For all $H$ and $e$, there is at most one $c$ such that $H ; e \downarrow c$.

We rigged it that way...

what would division, undefined-variables, or gettime() do?

Note: Our semantics is *syntax-directed*. 
Some theory comments

Inference rules are PL notation for some standard math...

- "H and e evaluating to c" is a relation on triples of the form (H, e, c) (i.e., H ; e ↓ c)

- Relation defined inductively on the derivation height

- Can define syntax the same way:

  \[
  \begin{align*}
  c \in E \quad &\quad x \in E \\
  e_1 \in E \quad &\quad e_2 \in E \quad &\quad e_1 \in E \quad &\quad e_2 \in E \\
  &\quad e_1 + e_2 \in E \quad &\quad e_1 \ast e_2 \in E
  \end{align*}
  \]

  Less metanotation for you, but not what “we” do
Statement semantics

\[ H_1 ; s_1 \to H_2 ; s_2 \]

**ASSIGN**

\[
H ; e \Downarrow c
\]

\[
H ; x := e \to H, x \mapsto c ; \text{skip}
\]

**SEQ1**

\[
H ; \text{skip}; s \to H ; s
\]

**SEQ2**

\[
H ; s_1 \to H' ; s'_1
\]

\[
H ; s_1 ; s_2 \to H' ; s'_1 ; s_2
\]

**IF1**

\[
H ; e \Downarrow c \quad c > 0
\]

\[
H ; \text{if } e \ s_1 \ s_2 \to H ; s_1
\]

**IF2**

\[
H ; e \Downarrow c \quad c \leq 0
\]

\[
H ; \text{if } e \ s_1 \ s_2 \to H ; s_2
\]
Statement semantics cont’d

What about \textbf{while }e \textbf{ s} (do s and loop if e > 0)?

\begin{align*}
\text{WHILE} \\
H \text{ ; while e s } \rightarrow H \text{ ; if e (s; while e s) skip}
\end{align*}

Many other equivalent definitions possible
Program semantics

We defined $H ; s \rightarrow H' ; s'$, but what does “$s$” mean/do?

Our machine iterates: $H_1 ; s_1 \rightarrow H_2 ; s_2 \rightarrow H_3 ; s_3 \ldots$

Let $H_1 ; s_1 \rightarrow^* H_2 ; s_2$ mean “becomes after some number of steps” and pick a special “answer” variable $ans$

The program $s$ produces $c$ if $\cdot ; s \rightarrow^* H ; \text{skip}$ and $H(ans) = c$

Does every $s$ produce a $c$?
Example program execution

\[ x := 3; (y := 1; \textbf{while } x (y := y \times x; x := x - 1)) \]

(Let’s write some of the state sequence. You can justify each step with a full derivation. Let \( s = (y := y \times x; x := x - 1) \).)

\[
\begin{align*}
\vdots & ; x := 3; y := 1; \textbf{while } x \ s \\
\rightarrow & , x \mapsto 3; \textbf{skip}; y := 1; \textbf{while } x \ s \\
\rightarrow & , x \mapsto 3; y := 1; \textbf{while } x \ s \\
\rightarrow^2 & , x \mapsto 3, y \mapsto 1; \textbf{while } x \ s \\
\rightarrow & , x \mapsto 3, y \mapsto 1; \textbf{if } x \ (s; \textbf{while } x \ s) \ \textbf{skip} \\
\rightarrow & , x \mapsto 3, y \mapsto 1; y := y \times x; x := x - 1; \textbf{while } x \ s
\end{align*}
\]
Continued...

\[ \rightarrow^2 \quad \cdot, \ x \leftarrow 3, \ y \leftarrow 1, \ y \leftarrow 3; \ x := x - 1; \ \textbf{while} \ x \ s \]
\[ \rightarrow^2 \quad \cdot, \ x \leftarrow 3, \ y \leftarrow 1, \ y \leftarrow 3, \ x \leftarrow 2; \ \textbf{while} \ x \ s \]
\[ \rightarrow \quad \ldots, \ y \leftarrow 3, \ x \leftarrow 2; \ \textbf{if} \ x \ (s; \ \textbf{while} \ x \ s) \ \textbf{skip} \]
\[ \quad \ldots \]
\[ \rightarrow \quad \ldots, \ldots, \ y \leftarrow 6, \ x \leftarrow 0; \ \textbf{skip} \]
Where we are

We have defined $H; e \downarrow c$ and $H; s \rightarrow H'; s'$ and extended the latter to give $s$ a meaning.

The way we did expressions is “large-step” or “natural”.

The way we did statements is “small-step”.

So now you have seen both.

Large-step does not distinguish errors and divergence.
Establishing Properties

We can prove a property of a terminating program by “running” it.

Example: Our last program terminates with x holding 0.

We can prove a program diverges, i.e., for all \( H \) and \( n \),

\[ \cdot; s \rightarrow^n H; \text{skip} \]

cannot be derived.

Example: \textbf{while 1 skip}

By induction on \( n \) with stronger induction hypothesis: If we can derive

\[ \cdot; s \rightarrow^n H; s' \]

then \( s' \) is \textbf{while 1 skip} or

\textbf{if 1 (skip; while 1 skip) skip} or \textbf{skip; while 1 skip}. 
More General Proofs

We can prove properties of executing all programs (satisfying another property)

Example: If $H$ and $s$ have no negative constants and $H; s \rightarrow^* H'; s'$, then $H'$ and $s'$ have no negative constants.

Example: If for all $H$, we know $s_1$ and $s_2$ terminate, then for all $H$, we know $H; (s_1; s_2)$ terminates.