CSE 505: Concepts of Programming Languages

Dan Grossman

Fall 2003

Lecture 3— Operational Semantics for IMP

Where we are

- Done: IMP syntax, structural induction, OCaml basics
- Today: IMP operational semantics
- Tonight: You could finish homework 1

Review

IMP's abstract syntax is defined inductively:

$$
\begin{array}{lcl}
s & ::= & \textbf{skip} \mid x := e \mid s; s \mid \textbf{if}\ e\ s\ s \mid \textbf{while}\ e\ s \\
e & ::= & c \mid x \mid e + e \mid e * e \\
(c & \in & \{\ldots, -2, -1, 0, 1, 2, \ldots\}) \\
(x & \in & \{x_1, x_2, \ldots, y_1, y_2, \ldots, z_1, z_2, \ldots, \ldots\})
\end{array}
$$

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

But we have a social understanding about variables and control flow

Expression semantics

$$H ::= \cdot \mid H, x \mapsto c$$

$$\boxed{H \; ; \; e \Downarrow c}$$

CONST

$$\frac{}{H \; ; \; c \Downarrow c}$$

VAR

$$\frac{}{H \; ; \; x \Downarrow H(x)}$$

ADD

$$\frac{H \; ; \; e_1 \Downarrow c_1 \qquad H \; ; \; e_2 \Downarrow c_2}{H \; ; \; e_1 + e_2 \Downarrow c_1 + c_2}$$

MULT

$$\frac{H \; ; \; e_1 \Downarrow c_1 \qquad H \; ; \; e_2 \Downarrow c_2}{H \; ; \; e_1 * e_2 \Downarrow c_1 * c_2}$$

"pronounce" as proofs (upward) or evaluations (downward)

Expression semantics cont'd

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

Last case avoids "errors" (makes function *total*)

We have *rule schemas* ("rules"). We *instantiate* a rule by replacing metavariables appropriately.

Instantiating rules

Example instantiation:

$$\frac{\cdot, y \mapsto 4 \; ; \; 3 + y \Downarrow 7 \qquad \cdot, y \mapsto 4 \; ; \; 5 \Downarrow 5}{\cdot, y \mapsto 4 \; ; \; (3 + y) + 5 \Downarrow 12}$$

Instantiates:

$$\frac{H \; ; \; e_1 \Downarrow c_1 \qquad H \; ; \; e_2 \Downarrow c_2}{H \; ; \; e_1 + e_2 \Downarrow c_1 + c_2}$$

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:

$$\dfrac{\dfrac{\dfrac{}{\cdot, y{\mapsto}4 \,;\, 3 \Downarrow 3} \qquad \dfrac{}{\cdot, y{\mapsto}4 \,;\, y \Downarrow 4}}{\cdot, y{\mapsto}4 \,;\, 3 + y \Downarrow 7} \qquad \dfrac{}{\cdot, y{\mapsto}4 \,;\, 5 \Downarrow 5}}{\cdot, y{\mapsto}4 \,;\, (3 + y) + 5 \Downarrow 12}$$

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., $H\ ;\ e \Downarrow c$)
- Relation defined inductively on the derivation *height*
- Can define syntax the same way:

$$\frac{}{c \in E} \qquad \frac{}{x \in E}$$

$$\frac{e_1 \in E \qquad e_2 \in E}{e_1 + e_2 \in E} \qquad \frac{e_1 \in E \qquad e_2 \in E}{e_1 * e_2 \in E}$$

Less metanotation for you, but not what "we" do

Statement semantics

$$\boxed{H_1 \; ; \; s_1 \longrightarrow H_2 \; ; \; s_2}$$

ASSIGN

$$\frac{H \; ; \; e \Downarrow c}{H \; ; \; x := e \longrightarrow H, x \mapsto c \; ; \; \textsf{skip}}$$

SEQ1

$$\frac{}{H \; ; \; \textsf{skip}; s \longrightarrow H \; ; \; s}$$

SEQ2

$$\frac{H \; ; \; s_1 \longrightarrow H' \; ; \; s_1'}{H \; ; \; s_1; s_2 \longrightarrow H' \; ; \; s_1'; s_2}$$

IF1

$$\frac{H \; ; \; e \Downarrow c \qquad c > 0}{H \; ; \; \textsf{if} \; e \; s_1 \; s_2 \longrightarrow H \; ; \; s_1}$$

IF2

$$\frac{H \; ; \; e \Downarrow c \qquad c \leq 0}{H \; ; \; \textsf{if} \; e \; s_1 \; s_2 \longrightarrow H \; ; \; s_2}$$

Statement semantics cont'd

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

$$\frac{\text{WHILE}}{H \; ; \; \textbf{while}\; e \; s \rightarrow H \; ; \; \textbf{if}\; e \; (s; \textbf{while}\; e \; s) \; \textbf{skip}}$$

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 $\cdot; s \rightarrow^* H; \textbf{skip}$ and $H(ans) = c$

Does every s produce a c?

Example program execution

$x := 3; (y := 1; \textsf{while}\ x\ (y := y * x; x := x - 1))$

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

$$
\begin{array}{ll}
 & \cdot; x := 3; y := 1; \textsf{while}\ x\ s \\
\rightarrow & \cdot, x \mapsto 3; \textsf{skip}; y := 1; \textsf{while}\ x\ s \\
\rightarrow & \cdot, x \mapsto 3; y := 1; \textsf{while}\ x\ s \\
\rightarrow^2 & \cdot, x \mapsto 3, y \mapsto 1; \textsf{while}\ x\ s \\
\rightarrow & \cdot, x \mapsto 3, y \mapsto 1; \textsf{if}\ x\ (s; \textsf{while}\ x\ s)\ \textsf{skip} \\
\rightarrow & \cdot, x \mapsto 3, y \mapsto 1; y := y * x; x := x - 1; \textsf{while}\ x\ s
\end{array}
$$

Continued...

$$\begin{array}{ll}
\rightarrow^2 & \cdot, x \mapsto 3, y \mapsto 1, y \mapsto 3; x := x{-}1; \textsf{while}\ x\ s \\
\rightarrow^2 & \cdot, x \mapsto 3, y \mapsto 1, y \mapsto 3, x \mapsto 2; \textsf{while}\ x\ s \\
\rightarrow & \ldots, y \mapsto 3, x \mapsto 2; \textsf{if}\ x\ (s; \textsf{while}\ x\ s)\ \textsf{skip} \\
\ldots & \\
\rightarrow & \ldots, \ldots, y \mapsto 6, x \mapsto 0; \textsf{skip}
\end{array}$$

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 $\mathbf{0}$.

We can prove a program diverges, i.e., for all H and n, $\cdot; s \rightarrow^n H; \textbf{skip}$ cannot be derived.

Example: **while** 1 **skip**

By induction on n with stronger induction hypothesis: If we can derive $\cdot; s \rightarrow^n H; s'$ then s' is **while** 1 **skip** or **if** 1 (**skip; while** 1 **skip**) **skip** or **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.

Even more general proofs to come

We *defined* the semantics.

Given our semantics, we established properties of programs and sets of programs.

More interesting is having multiple semantics—for what program states are they equivalent? (For what notion of equivalence?)

Or having a more abstract semantics (e.g., a type system) and asking if it is preserved under evaluation. (If e has type τ and e becomes e', does e' have type τ?)

But first a one-lecture detour to "denotational" semantics.