CSE 505:
Concepts of Programming Languages

Dan Grossman

Fall 2009

Lecture 3— Operational Semantics for IMP

Where we are

- Done: Caml basics, IMP syntax, structural induction
- Today: IMP operational semantics
- Tonight: You could (almost?) finish homework 1

Review

IMP's abstract syntax is defined inductively:

$$
\begin{array}{rcl}
s & ::= & \textbf{skip} \mid x := e \mid s; s \mid \textbf{if } e\ s\ s \mid \textbf{while } e\ s \\
e & ::= & c \mid x \mid e + e \mid e * e \\
(c & \in & \{\ldots, -2, -1, 0, 1, 2, \ldots\}) \\
(x & \in & \{\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{z}_1, \mathrm{z}_2, \ldots, \ldots\})
\end{array}
$$

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

Encode our "social understanding" about variables and control flow

Outline

- Semantics for expressions
  1. Informal idea; the need for *heaps*
  2. Definition of heaps
  3. The evaluation *judgment* (a relation form)
  4. The evaluation *inference rules* (the relation definition)
  5. Using inference rules
     - *Derivation trees* as interpreters
     - Or as proofs about expressions
  6. *Metatheory*: Proofs about the semantics
- Then semantics for statements
  - ...

Informal idea

Given e, what c does it evaluate to?

It depends on the values of variables (of course).

Use a heap H to encode a total function from variables to constants.

- Could use partial functions, but then $\exists$ H and e for which there is no c.

We'll define a *relation* over triples of H, e, and c.

- Will turn out to be *function* if we view H and e as inputs and c as output.
- With our metalanguage, easier to define a relation and then prove it is a function (if it is).

Heaps

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

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

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

"What heap to use" will arise in the statement semantics

- For expression evaluation, "we are given an H"

The judgment

We will write:

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

to mean, "e evaluates to c under heap H".

It is just a relation on triples of the form (H, e, c).

We just made up metasyntax $H \;;\; e \Downarrow c$ to follow PL convention and to distinguish it from other relations.

We can write: $., x \mapsto 3 \;;\; x + y \Downarrow 3$, which will turn out to be *true* (this triple will be in the relation we define).

Or: $., x \mapsto 3 \;;\; x + y \Downarrow 6$, which will turn out to be *false* (this triple will not be in the relation we define).

Inference rules

$$\textsc{Const}\ \frac{}{H\ ;\ c \Downarrow c} \qquad \textsc{Var}\ \frac{}{H\ ;\ x \Downarrow H(x)} \qquad \textsc{Add}\ \frac{H\ ;\ e_1 \Downarrow c_1 \qquad H\ ;\ e_2 \Downarrow c_2}{H\ ;\ e_1 + e_2 \Downarrow c_1 + c_2}$$

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

Bottom: *conclusion*

Top: *hypotheses*

By definition, if all hypotheses hold, then the conclusion holds.

Each rule is a *schema* you "instantiate consistently".

- So rules "work" "for all" H, c, e_1, etc.
- But "each" e_1 has to be the "same" expression.

Instantiating rules

Example instantiation:

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

Instantiates:

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

with $\boldsymbol{H} = \cdot, \mathrm{y} \mapsto 4$, $\boldsymbol{e_1} = (\mathbf{3} + \mathrm{y})$, $\boldsymbol{c_1} = \mathbf{7}$, $\boldsymbol{e_2} = \mathbf{5}$, $\boldsymbol{c_2} = \mathbf{5}$

Derivations

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

Example:

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

So $\boldsymbol{H}\ ;\ \boldsymbol{e} \Downarrow \boldsymbol{c}$ if there exists a derivation with $\boldsymbol{H}\ ;\ \boldsymbol{e} \Downarrow \boldsymbol{c}$ at the root.

Back to relations

So what relation do our inference rules define?

- Start with empty relation (no triples) R_0
- Let R_i be R_{i-1} union all $H \; ; e \Downarrow c$ such that we can instantiate some inference rule to have conlusion $H \; ; e \Downarrow c$ and all hypotheses in R_{i-1}.
  - So R_i is all triples at the bottom of height-j complete derivations for $j \leq i$.
- R_∞ is the relation we defined
  - All triples at the bottom of complete derivations.

For the math folks: R_∞ is the smallest relation closed under the inference rules.

What are these things?

We can view the inference rules as defining an *interpreter*.

- Complete derivation shows recursive calls to the "evaluate expression" function.
  - Recursive calls from conclusion to hypotheses.
  - Syntax-directed means the interpreter need not "search".
- See OCaml code in homework 1

Or we can view the inference rules as defining a *proof system*.

- Complete derivation proves facts from other facts starting with axioms.
  - Facts established from hypotheses to conclusions.

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*.

Proofs are by induction on the the structure (i.e., height) of the expression e.

On to statements

A statement doesn't produce a constant.

It produces a new, possibly-different heap.

- If it terminates

We could define $H_1 \; ; \; s \Downarrow H_2$

- Would be a partial function from H_1 and s to H_2
- Works fine; could be a homework problem

Instead we'll define a "small-step" semantics and then "iterate" to "run the program"

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

Statement semantics

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

ASSIGN

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

SEQ1

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

SEQ2

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

IF1

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

IF2

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

Statement semantics cont'd

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

$$\dfrac{\textsc{while}}{H\ ;\ \textbf{while}\ e\ s \rightarrow H\ ;\ \textbf{if}\ e\ (s; \textbf{while}\ e\ s)\ \textbf{skip}}$$

Many other equivalent definitions possible

Program semantics

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

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

with each step justified by a complete derivation using our single-step statement semantics

Let $H_1 ; s_1 \rightarrow^* H_2 ; s_2$ mean "becomes after 0 or more 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

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

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

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

Continued...

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

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”.

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

So now you have seen both.

Large-step does not distinguish errors and divergence.

- But we defined IMP to have no errors
- And expressions never diverge

Establishing Properties

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

Example: Our last program terminates with $\mathtt{x}$ holding $\mathbf{0}$.

We can prove a program diverges, i.e., for all H and n,
$\cdot\,;\, s \;\rightarrow^{n}\; H\,;\,$ **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.