CSE 505: Concepts of Programming Languages

Dan Grossman

Fall 2007

Lecture 2— Abstract Syntax

Finally, some content

For our first *formal language*, let's leave out functions, objects, records, threads, exceptions, ...

What's left: integers, assignment (mutation), control-flow

(Abstract) syntax using a common meta-notation:
"A program is a statement s defined as follows"

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

Syntax definition

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

- Blue is metanotation ($::=$ "can be a", $\mid$ "or")
- *Metavariables* represent "anything in the *syntax class*"
- Use parentheses to *disambiguate*, e.g., **if** x **skip** $\mathrm{y} := 0; \mathrm{z} := 0$

E.g.: $\mathrm{y} := 1;$ **while** $\mathrm{x}\ (\mathrm{y} := \mathrm{y} * \mathrm{x}; \mathrm{x} := \mathrm{x} - 1)$

Inductive definition

With care, our syntax definition is *not* circular!

$$s ::= \textbf{skip} \mid x := e \mid s; s \mid \textbf{if}\ e\ s\ s \mid \textbf{while}\ e\ s$$

$$e ::= c \mid x \mid e + e \mid e * e$$

Let $E_0 = \emptyset$. For $i > 0$, let E_i be E_{i-1} union "expressions of the form c, x, $e + e$, or $e * e$ *where* $e \in E_{i-1}$". Let $E = \bigcup_{i \geq 0} E_i$. The set E is what we mean by our compact metanotation.

To get it: What set is E_1? E_2?

Explain statements the same way. What is S_1? S_2? Stop only when you're bored.

Proving Obvious Stuff

All we have is syntax (sets of abstract-syntax trees), but let's get the idea of proving things carefully...

Theorem 1: There exist expressions with three constants.

Our First Theorem

There exist expressions with three constants.

Pedantic Proof: Consider $e = \mathbf{1} + (\mathbf{2} + \mathbf{3})$. Showing $e \in E_3$ suffices because $E_3 \subseteq E$. Showing $\mathbf{2} + \mathbf{3} \in E_2$ and $\mathbf{1} \in E_2$ suffices...

PL-style proof: Consider $e = \mathbf{1} + (\mathbf{2} + \mathbf{3})$ and definition of E.

Theorem 2: All expressions have at least one constant or variable.

Our Second Theorem

All expressions have at least one constant or variable.

Pedantic proof: By induction on i, show for all $e \in E_i$.

- Base: $i = 0$ implies $E_i = \emptyset$
- Inductive: $i > 0$. Consider *arbitrary* $e \in E_i$ by cases:
  - $e \in E_{i-1} \ldots$
  - $e = c \ldots$
  - $e = x \ldots$
  - $e = e_1 + e_2$ where $e_1, e_2 \in E_{i-1} \ldots$
  - $e = e_1 * e_2$ where $e_1, e_2 \in E_{i-1} \ldots$

A "Better" Proof

All expressions have at least one constant or variable.

PL-style proof: By *structural induction* on (rules for forming an expression) e. Cases:

- c ...
- x ...
- $e_1 + e_2$ ...
- $e_1 * e_2$ ...

Structural induction invokes the induction hypothesis on *smaller* terms. It is equivalent to the pedantic proof, and the convenient way.