CSE 505:
Concepts of Programming Languages

Dan Grossman
Fall 2008
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”

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

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

\[ e ::= c \mid x \mid e + e \mid e \times e \]

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

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

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

E.g.: \textbf{y := 1; while } x \ (y := y \times x; x := x - 1)
Inductive definition

With care, our syntax definition is not circular!

\[
\begin{align*}
  s & ::= \text{skip} \mid x ::= e \mid s; s \mid \text{if } e \text{ } s \text{ } s \mid \text{while } e \text{ } s \\
  e & ::= c \mid x \mid e + e \mid e \ast e
\end{align*}
\]

Let \( E_0 = \emptyset \). For \( i > 0 \), let \( E_i \) be \( E_{i-1} \) union “expressions of the form \( c, x, e_1 + e_2, \) or \( e_1 \ast e_2 \) where \( e_1, e_2 \in E_{i-1} \)”. 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 \( S_1 \)? \( S_2 \)?
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 $e = 1 + (2 + 3)$. Showing $e \in E_3$ suffices because $E_3 \subseteq E$. Showing $2 + 3 \in E_2$ and $1 \in E_2$ suffices...

PL-style proof: Consider $e = 1 + (2 + 3)$ 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 \( i \), for all \( e \in E_i \), \( e \) has \( \geq 1 \) constant or variable.

- 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 \( e_1, e_2 \in E_{i-1} \ldots \)
  - \( e = e_1 \times e_2 \) 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:

- $c \ldots$
- $x \ldots$
- $e_1 + e_2 \ldots$
- $e_1 * e_2 \ldots$

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