Natural and Contextual Semantics

Lecture 5
CS 565
1/30/06
Natural Semantics

- The semantics given previously is known as “small-step”
  - Evaluation relation shows how each individual step in the computation takes place
  - Closely mirrors how an interpreter might evaluate a program
  - Apply a multi-step evaluation relation on top to talk about terms evaluating (in many steps) to values.

- An alternative style called “natural semantics” directly formulates the notion of “this term evaluates to this value”
  - Define $\Rightarrow$ as the smallest binary relation that satisfies the rules given on the following slides
Natural Semantics for AExp

\[
\begin{align*}
<n, \sigma> &\Rightarrow n & \langle x, \sigma \rangle &\Rightarrow \sigma(x) \\
\langle e_1, \sigma \rangle &\Rightarrow n_1, \langle e_2, \sigma \rangle &\Rightarrow n_2 \\
\hline
\langle e_1 + e_2, \sigma \rangle &\Rightarrow n_1 + n_2
\end{align*}
\]

Note that evaluation of does not restrict order of evaluation, nor does it specify intermediate states generated in the course of evaluation.
Natural Semantics for BExp

\[
\begin{align*}
\langle \text{true}, \sigma \rangle & \Rightarrow \text{true} \quad \langle \text{false}, \sigma \rangle & \Rightarrow \text{false} \\
\langle n_1 = n_1, \sigma \rangle & \Rightarrow \text{true} \\
\langle n_1 = n_2, \sigma \rangle & \Rightarrow \text{false} \quad n_1 \neq n_2 \\
\langle e_1, \sigma \rangle & \Rightarrow n_1, \langle e_2, \sigma \rangle & \Rightarrow n_2 \\
\langle e_1 = e_2, \sigma \rangle & \Rightarrow \langle n_1 = n_2, \sigma \rangle \\
\langle b_1, \sigma \rangle & \Rightarrow \text{false} \quad \langle b_1, \sigma \rangle & \Rightarrow \text{true} \\
\langle b_1 \lor b_2, \sigma \rangle & \Rightarrow \langle b_2, \sigma \rangle \quad \langle b_1 \land b_2, \sigma \rangle & \Rightarrow \langle b_2, \sigma \rangle
\end{align*}
\]
Natural Semantics for Com

\[ \langle e, \sigma \rangle \Rightarrow n \]
\[ \langle x := e, \sigma \rangle \Rightarrow \sigma[x \mapsto n] \]

\[ \langle \text{skip}, \sigma \rangle \Rightarrow \sigma \]
\[ \langle c_1, \sigma \rangle \Rightarrow \sigma', \langle c_2, \sigma' \rangle \Rightarrow \sigma'' \]
\[ \langle c_1 ; c_2, \sigma \rangle \Rightarrow \sigma'' \]

\[ \langle b, \sigma \rangle \Rightarrow \text{true}, \langle c_1, \sigma \rangle \Rightarrow \sigma' \]
\[ \langle b, \sigma \rangle \Rightarrow \text{false}, \langle c_2, \sigma \rangle \Rightarrow \sigma' \]
\[ \langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \Rightarrow \sigma' \]

\[ \langle b, \sigma \rangle \Rightarrow \text{false} \]
\[ \langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \Rightarrow \sigma' \]
\[ \langle \text{while } b \text{ do } c, \sigma \rangle \Rightarrow \sigma \]
\[ \langle \text{while } b \text{ do } c, \sigma \rangle \Rightarrow \sigma' \]
Evaluation of Commands

- Order of evaluation is captured by implicit "threading" of the store
  - In a sequence operation, $\sigma'$ (produced by evaluating $c_1$) is needed to evaluate $c_2$

- Rules are not "syntax-directed"
  - In rule for while, antecedent clause is "bigger" than consequent
    - What is the implication? Consider reasoning about termination.
Issues

- Two disadvantages to natural-style semantics:
  - Reasoning about termination
  - Exposing intermediate states
    - order of evaluation is implicit in use of store, not explicit in rules
Evaluation Contexts

- Both styles of semantics address two concerns:
  - order of evaluation
    - explicit in small-step semantics
    - implicit in natural semantics
  - meaning of terms
- Can we separate out these two notions?
  - Decompose a term into two parts:
    - the part of the term that is to be evaluated
    - the remaining portion of the term that should be examined after the subterm evaluates
      - call this part of the term a “context”
Redex

A redex is a term that can be transformed in a single step

- A redex has no antecedents
- \( r ::= x | x := n | x := n + n | \text{skip}; c | \text{if true then } c_1 \text{ else } c_2 | \text{if false then } c_1 \text{ else } c_2 | \text{true} \land b | \text{false} \lor b | \ldots \)
Evaluation Contexts

- An evaluation context is a term with a “hole” in the place of a subterm
  - Location of the hole points to the next subexpression that should be evaluated
  - If E is a context than E[r] is the expression obtained by replacing redex r for the hole defined by context E
  - Now, if <r,σ> → <t,σ'> then <E[t],σ> → <E[t],σ'>
Contexts

- Can define evaluation context via a grammar:

\[ E ::= [ ] | n + E | n * E | x := E | \]

\[ \text{if } E \text{ then } c_1 \text{ else } c_2 | \]

\[ E; c | \text{while } E \text{ do } c \]

The grammar fixes the order of evaluation, allowing us to simplify the number and structure of the rules used in the semantics.
Evaluation Contexts

- A context has exactly one hole
- Redexes that are substituted for a context are never values
- A context uniquely identifies the next redex to be evaluated
  - Consider \( e_1 + e_2 \) and its decomposition as \( E[r] \)
  - If \( e_1 = n_1 \) and \( e_2 \) is \( n_2 \) then \( E = [] \)
  - If \( e_1 = n_1 \) and \( e_2 \neq n_2 \) then \( E = n_1 + E[e_2] \)
  - If \( e_1 \neq n_1 \) then \( E = E[e_1] + e_2 \)
  - Last two cases are evaluated recursively
Evaluation Contexts

- Consider \( c = c_1; c_2 \)
  - Suppose \( c_1 = \text{skip} \). Then, \( c = E[\text{skip}; c_2] \) with \( E = [\] \)
  - Suppose \( c_1 \neq \text{skip} \). Then, \( c_1 = E[r] \) and \( c = E'[r] \) with \( E' = E; c_2 \)

- Consider \( c = \text{if } b \text{ then } c_1 \text{ else } c_2 \)
  - If \( b = \text{true} \) then \( c = E[r] \) where \( r \) is a redex in \( c_1 \) and \( E \) defines its context
  - If \( b = \text{false} \) then \( c = E[r] \) where \( r \) is a redex in \( c_2 \) and \( E \) defines its context
  - Otherwise, \( b = E[r] \), so \( c = E'[r] \) where \( E' = \text{if } E \text{ then } c_1 \text{ else } c_2 \)

- Decomposition:
  - If \( c \) is not “\text{skip}” then there exists a unique \( E \) and \( r \) such that \( c = E[r] \)
    - Exists means progress
    - Unique means determinism
Example

Consider the evaluation of:
\[ x := 1; x := x + 1 \] with \( \sigma_0 = [x \mapsto 0] \)

<table>
<thead>
<tr>
<th>State</th>
<th>Context</th>
<th>Redex</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;x := 1; x := x + 1, \sigma_0&gt;)</td>
<td>([]; x := x + 1)</td>
<td>(x := 1)</td>
</tr>
<tr>
<td>(&lt;\text{skip}; x := x + 1, [x \mapsto 1]&gt;)</td>
<td>([] )</td>
<td>(\text{skip}; x := x + 1)</td>
</tr>
<tr>
<td>(&lt;x := x + 1, [x \mapsto 1]&gt;)</td>
<td>(x := [] + 1)</td>
<td>(x)</td>
</tr>
<tr>
<td>(&lt;x := 1 + 1, [x \mapsto 1]&gt;)</td>
<td>(x := [] )</td>
<td>(x := 1 + 1)</td>
</tr>
<tr>
<td>(&lt;\text{skip}, [x \mapsto 1 + 1]&gt;)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Contextual Semantics

Summary
- Think of a hole as representing a program counter
  - The rules for advancing holes is non-trivial
    - Must decompose the entire expression
    - How would you implement this?
- Major advantage of contextual semantics is that allows a mix of global and local reduction rules
  - Global rules indicate next redex to be evaluated
  - Local rules indicate how to perform the reduction