Contextual Operational Semantics

Meeting 8, CSCI 5535, Spring 2010

Announcements

• Homework 1 is due tonight
  - How’s it going?

• Office hours
  - Move earlier to after class today?

Questions?

Key Ideas

• Small-step operational semantics
  - Value, Normal form, Terminal program
• Contextual small-step operational sem.
  - Redex
  - Local reduction rules
  - Context
  - Global reduction rule
• What are these things about?
### Local Reduction Rules for IMP

One for each redex: \( \langle r, \sigma \rangle \rightarrow \langle e, \sigma' \rangle \)

- \( \langle x, \sigma \rangle \rightarrow \langle \sigma(x), \sigma \rangle \)
- \( \langle n_1 + n_2, \sigma \rangle \rightarrow \langle n, \sigma \rangle \) where \( n = n_1 + n_2 \)
- \( \langle n_1 = n_2, \sigma \rangle \rightarrow \langle \text{true}, \sigma \rangle \) if \( n_1 = n_2 \)
- \( \langle n_1 = n_2, \sigma \rangle \rightarrow \langle \text{false}, \sigma \rangle \) if \( n_1 \neq n_2 \)

### Global reduction rule

- \( \langle x := n, \sigma \rangle \rightarrow \langle \text{skip}, \sigma[x := n] \rangle \)
- \( \langle \text{skip}; c, \sigma \rangle \rightarrow \langle c, \sigma \rangle \)
- \( \langle \text{if true then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \langle c_1, \sigma \rangle \)
- \( \langle \text{if false then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \langle c_2, \sigma \rangle \)
- \( \langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \langle \text{if } b \text{ then } c; \text{while } b \text{ do } c \text{ else } \text{skip}, \sigma \rangle \)
Contextual Semantics Example

- \( x := 1 \); \( x := x + 1 \) with initial state \([x:=0]\)

<table>
<thead>
<tr>
<th>&lt;Comm, State&gt;</th>
<th>Redux •</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x := 1; x := x + 1, [x := 0] )</td>
<td>( x := 1 )</td>
<td>( x := x + 1 )</td>
</tr>
<tr>
<td>( \text{skip}; x := x + 1, [x := 1] )</td>
<td>( \text{skip}; x := x + 1 )</td>
<td>•</td>
</tr>
<tr>
<td>( x := x + 1, [x := 1] )</td>
<td>( x )</td>
<td>( x := x + 1 )</td>
</tr>
<tr>
<td>( x := 1 + 1, [x := 1] )</td>
<td>( 1 + 1 )</td>
<td>( x := x + 1 )</td>
</tr>
<tr>
<td>( x := 2, [x := 1] )</td>
<td>( x := 2 )</td>
<td>•</td>
</tr>
<tr>
<td>( \text{skip}, [x := 2] )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Defining Contexts

- **Contexts** are defined by a grammar:

  \[
  H ::= • | n + H \\
  | H + e | ... \\
  | x := H \\
  | \text{if } H \text{ then } c_1 \text{ else } c_2 \\
  | H; c
  \]

- A context has **exactly one** • marker
- Does this say something about order of evaluation?

What’s in a context?

- **Contexts** specify precisely how to find the next redex
  - Consider \((e_1, e_2)\) and its decomposition as \(H[r]\)
  - What are all the ways that \((e_1, e_2)\) could be split into \(H\) and \(r\)?
What's in a context?

• Contexts specify precisely how to find the next redex
  - Consider $e_1 + e_2$ and its decomposition as $H[r]
  - If $e_1$ is $n_1$ and $e_2$ is $n_2$ then $H = \cdot$ and $r = n_1 + n_2$.
  - If $e_1$ is $n_1$ and $e_2$ is not $n_2$ then $H = n_1 + H_2$ and $e_2 = H_2[r]$.
  - If $e_1$ is not $n_1$ then $H = H_1 + e_2$ and $e_1 = H_1[r]$.
  - In the last two cases the decomposition is done recursively.
  - Check that in each case the solution is unique.

Unique Next Redex: “Proof” By Handwaving Examples

• e.g., $c = \text{“} c_1; c_2 \text{”}$ – either
  - $c_1 = \text{skip}$ and then $c = H[\text{skip}; c_2]$ with $H = \cdot$
  - or $c_1 \neq \text{skip}$ and then $c_1 = H[r]$; so $c = H[r]$ with $H = H_1; c_2$

• e.g., $c = \text{“} \text{if } b \text{ then } c_1 \text{ else } c_2 \text{”}$
  - either $b = \text{true}$ or $b = \text{false}$ and then $c = H[r]$ with $H = \cdot$
  - or $b$ is not a value and $b = H[r]$; so $c = H[r]$ with $H = \text{if } H \text{ then } c_1 \text{ else } c_2$

Context Decomposition Theorem

• If $c$ is not “skip” then there exist unique $H$ and $r$ such that $c = H[r]$
  - “Exist” means progress.
  - “Unique” means determinism.

What if we want short-circuit evaluation of $\land$?

• Define the following contexts, redexes and local reduction rules:

  $H ::= \ldots |$
  $r ::= \ldots |$
  $<\ldots, \sigma> \rightarrow <\ldots, \sigma> |$
  $<\ldots, \sigma> \rightarrow <\ldots, \sigma> |$

  - the local reduction kicks in before $b_2$ is evaluated.

Summary: Contextual Operational Semantics

• Can view $\cdot$ as representing the program counter.
• The advancement rules for $\cdot$ are non-trivial.
  - At each step the entire command is decomposed.
  - This makes contextual semantics inefficient to implement directly.
• The major advantage of contextual semantics:
  allows a mix of local and global reduction rules.
  - For IMP we have only local reduction rules: only the redex is reduced.
  - Sometimes it is useful to work on the context too.

Reading Real-World Examples

• Cobbe and Felleisen, POPL 2005
• Small-step contextual op. sem. for Java
• Their rule for object field access:
  $P \vdash E[\text{obj.fd}], S \rightarrow E[F(fd)].S$
  where $F = \text{fields}(S(\text{obj}))$ and $fd \in \text{dom}(F)$
• They use “$E$“ for context, we use “$H$“.
• They use “$S$“ for state, we use “$\sigma$“.

Lost In Translation

• \( P \vdash <H[\text{obj.fd}], \sigma> \rightarrow <H[F(fd)], \sigma> \)
  where \( F = \text{fields}(\sigma(\text{obj})) \) and \( fd \in \text{dom}(F) \)

• They have "\( P \vdash \)" but that just means "it can be proved in our system given \( P \)"

• \( <H[\text{obj.fd}], \sigma> \rightarrow <H[F(fd)], \sigma> \)
  where \( F = \text{fields}(\sigma(\text{obj})) \) and \( fd \in \text{dom}(F) \)

• They model objects (like \( \text{obj} \)), but we do not (yet); let's just make \( fd \) a variable:

• \( <H[fd], \sigma> \rightarrow <H[F(fd)], \sigma> \)
  where \( F = \sigma \) and \( fd \in L \)

• That's just our variable-lookup rule:

• \( <H[fd], \sigma> \rightarrow <H[\sigma(fd)], \sigma> \) (when \( fd \in L \))