Announcements

- Homework 0 is due tonight
  - Office hours today at 4:45pm
- Homework 1 will be out tonight
  - Coding part using OCaml. Resources on the website.
- Note links to supplementary texts on operational semantics
  - Harper, Ch. 9 (Ch. 1 for judgments and inference rules)

Plan for Today and Next Time

- Review of Judgments, Inference Rules
- Big-Step Op. Sem. for IMP
- Small-Step Contextual Op. Sem. for IMP
- Reductions, Redexes, and Contexts
- "Real World"

Terminology Review: Semantics

- A formal semantics is a mathematical system to assign meaning to programs
- In operational semantics the meaning of a program is how we execute a program
- Any operational semantics gives inference rules that describe execution

Big-Step Operational Semantics

Meeting 6, CSCI 5535, Spring 2010

Questions?
Terminology Review: Semantics

- A **formal semantics** is a system for assigning meanings to programs.
- In **operational semantics** the meaning of a program is “what it evaluates to.”
- Any operational semantics gives **inference rules** that tell you how to evaluate programs.

Review: Judgments

- **Judgments** ("something that is knowable") are derived using rules of inference like
  - Example judgments we have defined:

Review: Evaluation Rules (for Aexp)

- This is called **structural operational semantics**
  - rules defined based on the structure of the expression
  - These rules do not impose an order of evaluation!
**Derivation**

- Apply inferences rules and put in a tree
- Provides proof of a judgment
  - "witnesses an element in the relation"
- Conclusion is at the bottom and the leaves at the top are axioms (rules with no hypotheses)

**Derivation (Example)**

- "Show that 3 + (4 - 5) evaluates to 2"

```
<3 + (4 - 5), \sigma> \Downarrow 2
```

**Evaluation Rules (for Bexp)**

- `<true, \sigma> \Downarrow true`
- `<false, \sigma> \Downarrow false`
- `<e_1, \sigma> \Downarrow n_1  
  <e_2, \sigma> \Downarrow n_2`
- `<e_1 \leq e_2, \sigma> \Downarrow 1`
- `<e_1 = e_2, \sigma> \Downarrow 1`
- `<e_1 \land e_2, \sigma> \Downarrow false`
- `<b_1 \land b_2, \sigma> \Downarrow false`
- `<b_1, \sigma> \Downarrow true`
- `<b_2, \sigma> \Downarrow true`
- `<b_1 \land b_2, \sigma> \Downarrow true`

What does this say about evaluating `∧`?

**Non-Short-Circuiting Rule for `∧`**

```
B - meta-vm for booleans (\textbf{w3})
\begin{align*}
\text{b- meta-vm for bool expression} & \\
\text{true} & \Downarrow B_i \\
\text{false} & \Downarrow B_j \\
\text{true} \land \text{false} & \Downarrow B_i \land B_j
\end{align*}
```

**Evaluation Rule(s) for `∨`**

```
<e_1, \sigma> \Downarrow B_1 \\
<e_2, \sigma> \Downarrow B_2 \\
<h \lor b_2, \sigma> \Downarrow \text{true}
```

**How to Read the Rules**

- Forward (top-down) = inference rules
  - if we know that the hypothesis judgments hold then we can infer that the conclusion judgment also holds
  - If we know that `<e_1, \sigma> \Downarrow 5` and `<e_2, \sigma> \Downarrow 7`, then we can infer that `<e_1 + e_2, \sigma> \Downarrow 12`
### How to Read the Rules

- **Backward (bottom-up) = evaluation rules**
  - Suppose we want to evaluate $e_1 + e_2$, i.e., find $n$ s.t. $e_1 + e_2 \Downarrow n$ is derivable using the previous rules.
  - By inspection of the rules we notice that the last step in the derivation of $e_1 + e_2 \Downarrow n$ must be the addition rule.
  - The other rules have conclusions that would not match $e_1 + e_2 \Downarrow n$.
  - This is called reasoning by **inversion** on the derivation rules.

### Syntax-Directed Evaluation

- Thus we must find $n_1$ and $n_2$ such that $e_1 \Downarrow n_1$ and $e_2 \Downarrow n_2$ are derivable.
  - This is done recursively.
- If there is exactly one rule for each kind of expression we say that the rules are **syntax-directed**.
  - At each step at most one rule applies.
  - This allows a simple evaluation procedure as above (recursive tree-walk).

### Demo: Direct translation to code

- **Evaluation of Commands**
  - The evaluation of a `Com` may have side effects but has **no direct result**.
    - What is the result of evaluating a command?
  - The "result" of a `Com` is a **new state**.
    - `<c, σ> ⊢ σ'`
    - But the evaluation of a `Com` might not terminate!?

- **Evaluation Rules (for Com)**
  - `<skip, σ> ⊢ σ`
  - `<c_1, σ> ⊢ σ'`
  - `<c_2, σ> ⊢ σ''`
  - `<if b then c_1 else c_2, σ> ⊢ σ'`
Evaluation Rules (for Com)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;\text{skip}, \sigma&gt; \Downarrow \sigma)</td>
<td>(&lt;\text{c}_1, \sigma&gt; \Downarrow \sigma' \quad \Downarrow \quad \text{c}_2, \sigma&gt; \Downarrow \sigma')</td>
</tr>
<tr>
<td>(\langle \text{if } b \text{ then } \text{c}_1 \text{ else } \text{c}_2, \sigma \rangle \Downarrow \sigma')</td>
<td>(&lt;b, \sigma&gt; \Downarrow \text{true} \quad \Downarrow \quad \text{c}_1, \sigma&gt; \Downarrow \sigma')</td>
</tr>
<tr>
<td>(&lt;b, \sigma&gt; \Downarrow \text{false} \quad \Downarrow \quad \text{c}_2, \sigma&gt; \Downarrow \sigma')</td>
<td>(&lt;\text{if } b \text{ then } \text{c}_1 \text{ else } \text{c}_2, \sigma&gt; \Downarrow \sigma')</td>
</tr>
</tbody>
</table>

Evaluation Rules (for Com, assignment)

<table>
<thead>
<tr>
<th>Def</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sigma<a href="x">x := n</a> = n)</td>
<td>(\sigma<a href="y">x := n</a> = \sigma(y))</td>
</tr>
<tr>
<td>(\sigma[x := n] \in \text{Winskel book})</td>
<td></td>
</tr>
</tbody>
</table>

Evaluation Rules (for Com, while)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;\text{while } b \text{ do } c, \sigma&gt; \Downarrow \sigma)</td>
<td>(&lt;b, \sigma&gt; \Downarrow \text{false} \quad \Downarrow \quad \text{while } b \text{ do } c, \sigma&gt; \Downarrow \sigma)</td>
</tr>
<tr>
<td>(&lt;b, \sigma&gt; \Downarrow \text{true} \quad \Downarrow \quad \text{c; while } b \text{ do } c, \sigma&gt; \Downarrow \sigma')</td>
<td>(&lt;\text{while } b \text{ do } c, \sigma&gt; \Downarrow \sigma')</td>
</tr>
</tbody>
</table>

Updated \(\sigma\) so that \([n/x]\) in Winskel book
For Next Time

• Homework 1 out
• Read Winskel, Chapter 3
  - Optional: additional background
  - Optional: more details
  - see web page