Operational Semantics
Part I

Jim Royer
CIS 352
February 15, 2018

References

- Andrew Pitts’ Lecture Notes on Semantics of Programming Languages
  http://www.inf.ed.ac.uk/teaching/courses/lsi/sempl.pdf
  We’ll be following the Pitts’ notes for a while and mostly using his notation.
- Matthew Hennessy’s Semantics of programming languages:
  is very readable and very good.
- There are many of other good references in Hennessy’s reading list:

Aexp, A little language for arithmetic expressions

Syntax

Concrete syntax
≈ phonemes, characters, words, tokens — the raw stuff of language

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, ...

Grammar
≈ collection of formation rules to organize parts into a whole. E.g.,
  - words into noun phrases, verb phrases, ..., sentences
  - key words, tokens, ... into expressions, statements, ..., programs

Abstract syntax
≈ a structure (e.g., labeled tree or data structure) showing how a “phrase” breaks down into pieces according to a specific rule.
Aexp’s abstract syntax

Grammar

\[ a ::= n \mid (a_1 + a_2) \mid (a_1 - a_2) \mid (a_1 \ast a_2) \]

In Haskell

```haskell
data AExp = Num Integer
          | Plus AExp AExp
          | Minus AExp AExp
          | Times AExp AExp
```

```
(((⌜2⌝ + ⌜5⌝) * ⌜13⌝) – ⌜9⌝)
```

```
Minus (Times (Plus (Num 2) (Num 5)) (Num 13)) (Num 9)
```

As a Parse Tree

Digression: Rules, 1

General Format for Rules

Name: \( \text{premise}_1 \cdots \text{premise}_k \) (side condition)

\( \text{conclusion} \)

Example 1.

- Modus Ponens: \( p \implies q \quad p \implies q \)
- Transitivity: \( x = y \quad y = z \implies x = z \)
- PLUS: \( a_1 \Downarrow v_1 \quad a_2 \Downarrow v_2 \quad (v_1 + v_2) \Downarrow v \quad (v = v_1 + v_2) \)

Digression: Rules, 2

Definition 2.

A rule with no premises is an axiom.

Definition 3.

A rule is sound if and only if the conclusion is true whenever the premises (and side-condition—if any) are true.

Question

So an axiom is sound when . . . ?
Digression: Rules, 3

General Format for Rules

Name: \( \text{premise}_1 \quad \cdots \quad \text{premise}_k \) (side condition)

\( \text{conclusion} \)

Proofs from gluing together rule applications

\[
\begin{align*}
\text{Num: } \ & \frac{2 \Downarrow 2 \quad 5 \Downarrow 5}{(2 + 5) \Downarrow 7} \quad \text{(side condition)} \quad \frac{13 \Downarrow 13}{(2 + 5) \Downarrow 13} \\
\text{Plus: } \ & \frac{2 \Downarrow 2 \quad 5 \Downarrow 5}{(2 + 5) \Downarrow 13} \quad \text{(side condition)} \quad \frac{13 \Downarrow 13}{(2 + 5) \Downarrow 13} \\
\text{Times: } \ & \frac{2 \Downarrow 2 \quad 5 \Downarrow 5}{(2 + 5) \Downarrow 91} \quad \text{(side condition)} \quad \frac{13 \Downarrow 13}{(2 + 5) \Downarrow 91}
\end{align*}
\]

The big-step semantics in Haskell

A Haskell version of the abstract syntax

```haskell
data Aexp = Num Integer
  | Add Aexp Aexp
  | Sub Aexp Aexp
  | Mult Aexp Aexp
```

The big-step semantics as an evaluator function

\[
aBig \ (\text{Add } a1 \ a2) = (aBig \ a1) + (aBig \ a2) \\
aBig \ (\text{Sub } a1 \ a2) = (aBig \ a1) - (aBig \ a2) \\
aBig \ (\text{Mult } a1 \ a2) = (aBig \ a1) * (aBig \ a2) \\
aBig \ (\text{Num } n) = n
\]

Rules can also be the basis of a computation

Do these rules make sense?

**Theorem 4.**

Suppose \( e \in Aexp \).

Then there is a unique integer \( \nu \) such that \( e \Downarrow \nu \).

Proof (by rule induction).

**CASE: NUM.** This is immediate.

**CASE: PLUS.**

By IH, there are unique \( \nu_1 \) and \( \nu_2 \) such that \( a1 \Downarrow \nu_1 \) and \( a2 \Downarrow \nu_2 \).

By arithmetic, there is a unique \( \nu \) such that \( \nu = \nu_1 + \nu_2 \).

Hence, there is a unique \( \nu \) such that \( a1 + a2 \Downarrow \nu \).

**CASES: MINUS and MULT.** These follow mutatis mutandis. □

\[
\begin{align*}
\text{PLUS}_{\text{BSS}}: & \quad \frac{a1 \Downarrow \nu_1 \quad a2 \Downarrow \nu_2}{(a1 + a2) \Downarrow \nu} \quad (\nu = \nu_1 + \nu_2) \\
\text{NUM}_{\text{BSS}}: & \quad \frac{n}{\Downarrow \nu} \quad (N[n] = \nu)
\end{align*}
\]
What do $\mathbf{Aexp}$ expression mean? \textbf{Small-step rules}

$a ::= n \mid (a_1 + a_2) \mid (a_1 - a_2) \mid (a_1 \ast a_2) \mid v$

$PLUS_{1SSS}$: \quad $a_1 \rightarrow a'_1$
$\quad \frac{a_1 + a_2}{(a_1 + a_2) \rightarrow (a'_1 + a_2)}$

$PLUS_{2SSS}$: \quad $a_2 \rightarrow a'_2$
$\quad \frac{a_1 + a_2}{(a_1 + a_2) \rightarrow (a'_1 + a'_2)}$

$PLUS_{3SSS}$: \quad $v = v_1 + v_2$
$\quad \frac{v_1 + v_2}{v = v_1 + v_2}$

$NUM_{SSS}$: \quad $n \rightarrow v$
$\quad \frac{(N[n] = v)}{n \rightarrow v}$

Notes
- These are \textit{rewrite} rules.
- We now allow values in expressions.
- $a \rightarrow a'$ is a transition.
- $a \rightarrow a'$ is $\mathbf{Aexp}$ expression $a$ evaluates (or rewrites) to $a'$ in one-step.
- $v$ is a terminal expression.
- The rules for $\rightarrow$ and $\ast$ follow the same pattern as the $+$-rules.

Some full small-step derivations of transitions

The derivations show that the steps in the transition sequence below are legal (i.e., follow from the rules).

<table>
<thead>
<tr>
<th>Transition</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MINUS_{3}$</td>
<td>$(8 - 3) \rightarrow 5$</td>
</tr>
<tr>
<td>$PLUS_{2}$</td>
<td>$((6 + (8 - 3)) \rightarrow (6 + 5))$</td>
</tr>
<tr>
<td>$MULT_{1}$</td>
<td>$((6 + 5) \rightarrow 11)$</td>
</tr>
<tr>
<td>$MULT_{2}$</td>
<td>$((6 + 5) * (5 - 2)) \rightarrow 11 * (5 - 2)$</td>
</tr>
<tr>
<td>$MINUS_{3}$</td>
<td>$(5 - 2) \rightarrow 3$</td>
</tr>
<tr>
<td>$MULT_{3}$</td>
<td>$((11 * 3) \rightarrow 33)$</td>
</tr>
</tbody>
</table>

Class exercise

Show:

$(((3\ast2)+(8\ast3))\ast((2\ast3)+(5\ast3)))$
$\rightarrow$
$(((3\ast2)+(8\ast3))\ast((2\ast3)+(5\ast3)))$
$\rightarrow$
$(((3\ast2)+(8\ast3))\ast((2\ast3)+(5\ast3)))$

There is a lattice of transitions
Properties of operational semantics

**Definition 5.**
A transition system \((\Gamma, \leadsto, T)\) is deterministic when for all \(a, a_1,\) and \(a_2:\)

\[
\text{If } a \leadsto a_1 \text{ and } a \leadsto a_2, \text{ then } a_1 = a_2.
\]

**Theorem 6.**
The big-step semantics for \(\text{Aexp} \cup \mathbb{Z} \cup \Rightarrow, \mathbb{Z}\) is deterministic.

The proof is an easy rule induction.

**Theorem 7.**
The given small-step semantics \((\text{Aexp} \cup \mathbb{Z} \cup \Rightarrow, \mathbb{Z})\) fails to be deterministic, but for all \(a \in \text{Aexp}\) and \(v_1, v_2 \in \mathbb{Z},\) if \(a \Rightarrow^* v_1\) and \(a \Rightarrow^* v_2,\) then \(v_1 = v_2.

This proof is tricky because of the nondeterminism.

Very sketchy proof-sketch, continued

**Theorem 9.**
The given small-step semantics \((\text{Aexp} \cup \mathbb{Z} \cup \Rightarrow, \mathbb{Z})\) fails to be deterministic, but for all \(a \in \text{Aexp}\) and \(v_1, v_2 \in \mathbb{Z},\) if \(a \Rightarrow^* v_1\) and \(a \Rightarrow^* v_2,\) then \(v_1 = v_2.

- The \(a_1\) and \(a_2\) are expressions with \(n\) or fewer operators.
- The last step in any transition sequence \(a \Rightarrow^* v\) is of the form \(v_1 + v_2 \Rightarrow v\) and justified by PLUS3.
- In each step before the last, the final rule in the step-justification was either a PLUS1 or a PLUS2. [Clarify!]
- If we look at the premises of the PLUS1’s, they give a small-step derivation \(a_1 \Rightarrow^* v_1.\) By the IH, we know that any \(\Rightarrow\)-reduction sequence for \(a_1\) that ends with a value must produce \(v_1.
- Similarly, \(a_2 \Rightarrow^* v_2\) is also determined.
- So, it follows that if \(a \Rightarrow^* v,\) we must have \(v = v_1 + v_2.\)
The leftmost path through the lattice of transitions

Why multiple flavors of semantics?

They provide different views of computations.
- Big-step is good for reasoning about how the (big) pieces of things fit together.
- Small step is good at reasoning about the (small) steps of a computation fit together.
- Small step semantics is much better at modeling inherent nondeterminism (e.g., in concurrent programs).
- ... and there are other flavors (e.g., denotational) for other purposes (e.g., obtaining stronger forms of soundness).