Aexp, A little language for arithmetic expressions

Grammar

\[
\begin{align*}
  a & ::= n \\
  & | (a_1 + a_2) \\
  & | (a_1 - a_2) \\
  & | (a_1 * a_2)
\end{align*}
\]

Conventions

- Metavariables: \( n, a, b, w, x \), etc.
- We write 35 for the numeral 35.

Examples

\[
\begin{align*}
  n & ::= \ldots \\
  a & ::= 2 \\
  & | (\frac{2}{5}) \\
  & | (\frac{(2 + \frac{5}{3})}{13} - 9)
\end{align*}
\]

Operational Semantics

Part I

Jim Royer

CIS 352

February 18, 2016

References

- Andrew Pitts’ Lecture Notes on Semantics of Programming Languages
  http://www.inf.ed.ac.uk/teaching/courses/lai/seml.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’s abstract syntax**

**Grammar**

- \( a ::= n \)
- \( (a_1 + a_2) \)
- \( (a_1 - a_2) \)
- \( (a_1 * a_2) \)
- \( n ::= \ldots \)

**In Haskell**

```haskell
```

**As a Parse Tree**

```
- 9
  - 2
  + 5
  *
  13
```

**What do Aexp expression mean?**

Big-step rules

- \( a ::= n | (a_1 + a_2) | (a_1 - a_2) | (a_1 * a_2) \)

**PLUS**

- \( \frac{a_1 \downarrow v_1}{(a_1 + a_2) \downarrow v} \quad (v = v_1 + v_2) \)

**MINUS**

- \( \frac{a_1 \downarrow v_1}{(a_1 - a_2) \downarrow v} \quad (v = v_1 - v_2) \)

**MULT**

- \( \frac{a_1 \downarrow v_1}{(a_1 * a_2) \downarrow v} \quad (v = v_1 * v_2) \)

**NUM**

- \( n \downarrow v \quad (N[n] = v) \)

**Notes**

- \( a \downarrow v \equiv \text{expression } a \text{ evaluates to value } v. \)
- \( \downarrow \) is an evaluation relation.
- Upstairs assertions are called premises.
- Downstairs assertions are called conclusions.
- Parenthetical equations on the side are called side conditions.
- \( N : \text{numerals} \rightarrow \mathbb{Z}. \) i.e., \( N[-43] = -43. \)
- The NUMSS rule is an example of an axiom.

**Digression: Rules, 1**

**General Format for Rules**

- **Name:** \( \text{premise}_1 \ldots \text{premise}_k \) \quad conclusion \quad (side condition)

**Example**

- **Modus Ponens:** \( p \implies q \quad p \implies q \)
- **Transitivity:** \( x \equiv y \quad y \equiv z \quad x \equiv z \)
- **PLUS:** \( \frac{a_1 \downarrow v_1}{(a_1 + a_2) \downarrow v} \quad (v = v_1 + v_2) \)

**Digression: Rules, 2**

**General Format for Rules**

- **Name:** \( \text{premise}_1 \ldots \text{premise}_k \) \quad conclusion \quad (side condition)

**Definition**

A rule with no premises is an **axiom**.

**Definition**

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

Rules can also be the basis of a computation

The big-step semantics in Haskell

The big-step semantics as an evaluator function

A Haskell version of the abstract syntax

Theorem

Suppose $e \in \text{Aexp}$. Then there is a unique integer $v$ such that $e \downarrow v$.

Proof (by rule induction).

CASE: NUM. This is immediate.

CASE: PLUS. By IH, there are unique $v_1$ and $v_2$ such that $a_1 \downarrow v_1$ and $a_2 \downarrow v_2$. By arithmetic, there is a unique $v$ such that $v = v_1 + v_2$. Hence, there is a unique $v$ such that $a_1 + a_2 \downarrow v$.

CASES: MINUS and MULT. These follow mutatis mutandis.

\[
\begin{align*}
\text{PLUS}_{\text{BSS}}: \quad & a_1 \downarrow v_1 \quad a_2 \downarrow v_2 \quad (v = v_1 + v_2) \\
\text{NUM}_{\text{BSS}}: \quad & n \downarrow v \quad (N[n] = v)
\end{align*}
\]
What do $\text{Aexp}$ expression mean? Small-step rules

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

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

**PLUS-1 SSS:**
$$a_1 \rightarrow a'_1 \quad (a_1 + a_2) \rightarrow (a'_1 + a_2)$$

**PLUS-2 SSS:**
$$a_2 \rightarrow a'_2 \quad (a_1 + a_2) \rightarrow (a_1 + a'_2)$$

**PLUS-3 SSS:**
$$v_1 + v_2 \rightarrow v \quad (v = v_1 + v_2)$$

**NUM SSS:**
$$n \rightarrow v \quad (\mathcal{N}[n] = v)$$

Class exercise

Show:
$$(((3 \cdot 2) + (8 - 3)) \cdot (5 - 2))$$

$$\rightarrow \begin{cases} ((6 + (8 - 3)) \cdot (5 - 2)) \\ (((3 \cdot 2) + 5) \cdot (5 - 2)) \\ (((3 \cdot 2) + (8 - 3)) \cdot 3) \end{cases}$$

Some full small-step derivations of transitions

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
<th>Transition</th>
<th>Transition</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINUS3</td>
<td>$(8 - 3) \rightarrow 5$</td>
<td>$((6 + (8 - 3)) \cdot (5 - 2))$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
</tr>
<tr>
<td>PLUS2</td>
<td>$(6 + 5) \rightarrow 11$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + 5) * (5 - 2))$</td>
</tr>
<tr>
<td>MULT1</td>
<td>$(6 + (8 - 3)) \cdot (6 + 5)$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
</tr>
<tr>
<td>PLUS3</td>
<td>$(6 + (8 - 3)) \cdot (6 + 5)$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + 5) * (5 - 2))$</td>
</tr>
<tr>
<td>MULT2</td>
<td>$(6 + 5) * (5 - 2)$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + 5) * (5 - 2))$</td>
</tr>
<tr>
<td>MINUS3</td>
<td>$(5 - 2) \rightarrow 3$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + 5) * (5 - 2))$</td>
</tr>
<tr>
<td>MULT2</td>
<td>$(11 * (5 - 2))$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + 5) * (5 - 2))$</td>
</tr>
<tr>
<td>MULT3</td>
<td>$(11 * 3) \rightarrow 33$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + (8 - 3)) \cdot (6 + 5))$</td>
<td>$((6 + 5) * (5 - 2))$</td>
</tr>
</tbody>
</table>

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

There is a lattice of transitions
Properties of operational semantics

Definition
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
The big-step semantics for \(Aexp\) is deterministic.

The proof is an easy rule induction.

Theorem
The given small-step semantics \((Aexp \cup Z, \Rightarrow, Z)\) fails to be deterministic, but for all \(a \in Aexp\) and \(v_1, v_2 \in Z\), 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.

- 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 justification of the step was either PLUS1 or PLUS2.
- 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).