CMSC 336: Type Systems for Programming Languages

Lecture 2: Operational Semantics

Acar & Ahmed

15 January 2008

Contents

1 Announcements

Please turn in your exercises. The first homework is out and is due next week on Tuesday.

2 Introduction

Operational semantics models execution of programs. Instead of real machine instructions we use *abstract machines*. An abstract machine consists of some state and an evaluation relation that enables transitions between states. The state of an abstract machine often consists of some kind of stack, perhaps some representation of memory, and an expression to be evaluated. The transition function maps the state to another state by evaluating the expressions.

The idea is to model execution without going into the details of a real machine. The reason for staying at a higher level is to enable studying the mathematical properties of such systems.

A language researcher often spends a lot of time studying the semantics of his or her language. Typically, research starts with some language and some initial semantics. Then, the language and semantics evolves as the soundness and safety properties are studied. This evolution process is usually where a lot of the time goes. The researcher often understand the problem fully during this process. After the language and its semantics reaches a stable state, the abstract machine can be implemented on a real machine. The big bonus is that, assuming that the implementation is correct with respect to the abstract machine, then the system is safe and sound.

In the rest of this lecture, we will talk about an example language and give different semantics for it.

3 Big Step Semantics

Consider the abstract syntax for our little language for arithmetic expressions.

$$\begin{array}{rcl} t & ::= & \texttt{0} \mid \texttt{true} \mid \texttt{false} \mid \\ & & \texttt{isZero}\ t \mid \texttt{succ}\ t \mid \texttt{pred}\ t \mid \\ & & \texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ t \end{array}$$

From now on, whenever we write a term, visualize in your mind the AST (Abstract Syntax Tree) of that term.

Question: Draw the abstract syntax trees for the following terms.

1. `isZero` `pred` 0

2. `if` `isZero` 0 `then` `succ` `pred` 0 `else` `pred` `succ` 0.

Note that this language is not ambiguous. We therefore do not need any parenthesis or other disambiguation techniques.

To present an evaluation semantics, the first step is to define what the results, or *values*, of evaluation consists of.

Question: What are the values?

`true`

1. ?

`false`

2. ?

3. natural numbers?

4. integers?

Solution: `true` and `false` are definitely values because there is no way that we can evaluate them to anything else. But we have a decision to make as far as natural numbers and integers is concerned. The most natural choice is to have integers but let's just start with naturals for simplicity.

3.1 A big-step semantics with natural numbers

We can define values as follows:

$$\begin{array}{lll} n & ::= & 0 \mid \texttt{succ}\ n \\ v & ::= & \texttt{true} \mid \texttt{false} \mid n \end{array}$$

Some example numbers are `0, succ 0, succ succ succ 0` . This representation of numbers can be thought as the basic unary representation for numbers (*e.g.*, $\texttt{succ succ succ}\ 0 = 3$).

Question: : Refine the definition of the language.

$$\begin{array}{lll} n & ::= & 0 \mid \texttt{succ}\ n \\ v & ::= & \texttt{true} \mid \texttt{false} \mid n \\ & & \\ t & ::= & v \mid \\ & & \texttt{isZero}\ t \mid \texttt{succ}\ t \mid \texttt{pred}\ t \mid \\ & & \texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ t \end{array}$$

We are now ready to give the semantics. We will define an evaluation relation, denoted $\Downarrow$, using inference rules.

$$\frac{}{v \Downarrow v}$$

$$\frac{t_1 \Downarrow \texttt{true} \quad t_2 \Downarrow v_2}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \Downarrow v_3} \qquad \frac{t_1 \Downarrow \texttt{false} \quad t_3 \Downarrow v_3}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \Downarrow v_3}$$

$$\frac{t \Downarrow n}{\texttt{succ}\ t \Downarrow \texttt{succ}\ n}$$

$$\frac{t \Downarrow 0}{\texttt{pred}\ t \Downarrow 0} \qquad \frac{t \Downarrow \texttt{succ}\ n}{\texttt{pred}\ t \Downarrow n}$$

$$\frac{t \Downarrow 0}{\texttt{isZero}\ t \Downarrow \texttt{true}} \qquad \frac{t \Downarrow \texttt{succ}\ n}{\texttt{isZero}\ t \Downarrow \texttt{false}}$$

The high level idea is to consider each possible term and specify the desired behavior of the evaluation. For example, to evaluate an if statement, we need to know what branch to take. So we evaluate the conditional. Based on the outcome, we evaluate the appropriate branch and return the resulting value.

Although the language is simple, the evaluation rules may have already become quite confusing. So let's play with the semantics a bit to get a sense of what is going on.

Question: Evaluate `pred succ pred 0` .
Solution:

$$\dfrac{\dfrac{\texttt{pred}\ 0 \Downarrow 0}{\texttt{succ pred}\ 0 \Downarrow \texttt{succ}\ 0}}{\texttt{pred succ pred}\ 0 \Downarrow 0}$$

Question: Evaluate `pred succ pred pred 0` .
Solution:

$$\dfrac{\dfrac{\dfrac{\texttt{pred}\ 0 \Downarrow 0}{\texttt{pred pred}\ 0 \Downarrow 0}}{\texttt{succ pred pred}\ 0 \Downarrow \texttt{succ}\ 0}}{\texttt{pred succ pred pred}\ 0 \Downarrow 0}$$

Question: Evaluate `if isZero pred succ pred 0 then true else false`
Solution:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{pred}\ 0 \Downarrow 0}{\texttt{succ pred}\ 0 \Downarrow \texttt{succ}\ 0}}{\texttt{pred succ pred}\ 0 \Downarrow 0}}{\texttt{isZero pred succ pred}\ 0 \Downarrow \texttt{true}} \quad \texttt{true} \Downarrow \texttt{true}}{\texttt{if isZero pred succ pred}\ 0\ \texttt{then true else false} \Downarrow \texttt{true}}}{}$$

These are called *derivation trees.*

Exercise: Evaluate `if isZero pred succ pred 0 then true else false`
Answer: ...

Question: Prove that any term indeed evaluates to a proper value.
Solution: The proof is by *induction on the evaluation.* We will consider each evaluation rule and assume that the property holds for the precedents, and prove that it holds for the antecedents. Consider each rule. For the rules that return `true`, `false`, $0, n$, and `succ` n this holds trivially. Consider the rules that return a value v (or its variants). By induction hypothesis, we know that v is a proper value and thus the property holds.

Question: We used induction on evaluation. What exactly do we mean by this? Can we state this idea more precisely?
Solution: The induction is on the depth of the derivation tree (or the length of the evaluation) . For all evaluations up to length n, let's assume that this property holds. We can then consider an evaluation of length and n and show that it holds. Since the evaluation of precedents are shorter by the definition, the induction hypothesis applies to them.

3.2 A semantics with integers

In the previous section, we gave a semantics for this language, where we only returned natural numbers. A more intuitive semantics would actually involve integers (*i.e.*, include negative numbers). Let's try formulate an operational semantics for our language where the result can also be integers.

Question: How can we specify negative numbers?

$$
\begin{array}{lcl}
pn & ::= & 0 \mid \texttt{succ}\ pn \\
nn & ::= & 0 \mid \texttt{pred}\ nn \\
v & ::= & \texttt{true} \mid \texttt{false} \mid nn \mid pn \\
 & & \\
t & ::= & v \mid \\
 & & \texttt{isZero}\ t \mid \texttt{succ}\ t \mid \texttt{pred}\ t \mid \\
 & & \texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ t
\end{array}
$$

Question: Let's extend the semantics according to these values.

$$\frac{}{v \Downarrow v}$$

$$\frac{t_1 \Downarrow \texttt{true} \quad t_2 \Downarrow v_2}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \Downarrow v_3} \qquad \frac{t_1 \Downarrow \texttt{false} \quad t_3 \Downarrow v_3}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \Downarrow v_3}$$

$$\frac{t \Downarrow pn}{\texttt{succ}\ t \Downarrow \texttt{succ}\ pn} \qquad \frac{t \Downarrow \texttt{pred}\ nn}{\texttt{succ}\ t \Downarrow nn}$$

$$\frac{t \Downarrow nn}{\texttt{pred}\ t \Downarrow \texttt{pred}\ nn} \qquad \frac{t \Downarrow \texttt{succ}\ pn}{\texttt{pred}\ t \Downarrow pn}$$

$$\frac{t \Downarrow 0}{\texttt{isZero}\ t \Downarrow \texttt{true}} \qquad \frac{t \Downarrow \texttt{succ}\ pn}{\texttt{isZero}\ t \Downarrow \texttt{false}} \qquad \frac{t \Downarrow \texttt{pred}\ nn}{\texttt{isZero}\ t \Downarrow \texttt{false}}$$

Question: Prove that an evaluation indeed returns a proper value. **Solution:** Similar to the proof with the natural numbers.

Question: Evaluate `pred succ pred 0 .` **Solution:**

$$\dfrac{\dfrac{\texttt{pred}\ 0 \Downarrow \texttt{pred}\ 0}{\texttt{succ}\ \texttt{pred}\ 0 \Downarrow 0}}{\texttt{pred}\ \texttt{succ}\ \texttt{pred}\ 0 \Downarrow \texttt{pred}\ 0}$$

Question: Evaluate `pred succ pred pred 0 .` **Solution:**

$$\dfrac{\dfrac{\dfrac{\texttt{pred}\ 0 \Downarrow \texttt{pred}\ 0}{\texttt{pred}\ \texttt{pred}\ 0 \Downarrow \texttt{pred}\ \texttt{pred}\ 0}}{\texttt{succ}\ \texttt{pred}\ \texttt{pred}\ 0 \Downarrow \texttt{pred}\ 0}}{\texttt{pred}\ \texttt{succ}\ \texttt{pred}\ \texttt{pred}\ 0 \Downarrow \texttt{pred}\ \texttt{pred}\ 0}$$

4 Small Step Semantics

Big-step semantics leads to a natural and intuitive specifications. Proving type safety properties using big-step semantics, however, is possible only indirectly. When type safety is an important concern, it is often preferable to give a small stem semantics. In this class, we will therefore prefer the small-step semantics.

At a high level, the big step semantics can be viewed as a top-down approach, whereas the small-step semantics is bottom-up.

4.1 A Language of Booleans

As an example, consider the following simple language of booleans obtained by throwing away the arithmetic operations from our running example.

$$\begin{array}{lcl} v & ::= & \texttt{true} \mid \texttt{false} \\ t & ::= & v \mid \texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ t \end{array}$$

Here is a small-step semantics for this language.

$$\texttt{if true then}\ t_2\ \texttt{else}\ t_3 \rightarrow t_2$$

$$\texttt{if false then}\ t_2\ \texttt{else}\ t_3 \rightarrow t_3$$

$$\dfrac{t_1 \rightarrow t_1'}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_2 \rightarrow \texttt{if}\ t_1'\ \texttt{then}\ t_2\ \texttt{else}\ t_2}$$

The *evaluation relation* → represent a step of evaluation. We will read $t \rightarrow t'$, as t steps (or evaluates) to t'. The evaluation relation describes how the machine makes transitions from one state to another, or how it computes.

Unlike the big step semantics there is no rule for values. The idea is that an evaluation halts at a value.

The evaluation rules for the `if` statements takes one of three forms. If the value of the conditional is known, then the machine continues evaluating the appropriate branch. If the conditional is not yet a value, then the machine makes progress on evaluating the conditional.

Question: Give the derivation tree for the following derivations.

1. `if false then if false then false else true else true` → `false`.

2. `if if false then false else true then if true then false else false else true` → `if true then if true then false else false else true`

When giving the small-step semantics, there is often a choice as to what terms the evaluation must reduce first. Although this may seem like a minor point, it has very important implications. To enforce a consistent application of choices, we often rely on an *evaluation strategy* that specifies which choices must be made.

For example, the expression `if` $true$ `then if` $false$ `then` $false$ `else` $true$ `else` $true$ can be evaluated in two ways.

1. Take the `then` branch.

   ```
   if true then if false then false else true else true  →
                             if false then false else true
   ```

2. Evaluate the `then` branch.

   ```
   if true then if false then false else true else true  →
                               if true then true else true
   ```

The semantics that we presented will perform the first evaluation step. It is possible to give a semantics that will do the second.

We say $t \rightarrow t'$ is *derivable* if and only if there is a derivation tree for $t \rightarrow t'$.

Let's now prove some properties of the semantics to get a sense of how inductive proofs on single-step semantics work.

Question: Prove that evaluation is deterministic, i.e., $t \rightarrow t'$ and $t \rightarrow t''$ then $t' = t''$.
Solution: We will consider each possible term t separately.

1. Terms `true` and `false`: since no steps can be taken the theorem holds vacuously.

2. `if` t_1 `then` t_2 `else` t_3. In this case, we will do induction on derivation. Since we have two derivations that may be of different length, we need to show some care.

   In the base case, the length of the first derivation is one. In this case t_1 is `true` or `false`. Since in both cases, there is only one rule that can be applied to t, the theorem holds, *i.e.*, $t' = t'' = t_2$ or $t' = t'' = t_3$ for when $t_1 = \texttt{true}$ or $t_1 = \texttt{false}$.

   Suppose that the property holds when both derivations have depth less than d. Consider two derivations where at least one has depth d. Consider the last step of each derivation and let $t_1 \rightarrow t_1'$ and $t_1 \rightarrow t_1''$ be premises. We then know that $t' = \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3$, and $t'' = \texttt{if } t_1'' \texttt{ then } t_2 \texttt{ else } t_3$. Since the derivations $t_1 \rightarrow t_1'$ and $t_1 \rightarrow t_1''$, have depth less than d, the induction hypothesis applies and we have $t_1' = t_1''$ and therefore $t' = t''$.

4.2 Normal Forms

We say that a term t is in *normal form* if no evaluation rule applies to that term. For example, all values are in normal form because, by definition, no reduction rules apply to a value.

In a sound language, all values that are in normal form are also values.

Question: Show, for our language of booleans, that if t is in normal form, then it is a value. **Solution:** By structural induction on terms. If t is a value, then the statement holds trivially. Consider now a non-value term. If $t = \texttt{if true then } t_2 \texttt{ else } t_3$, then it is clearly not a value; the case when the conditional is `false` is symmetric. Suppose now that $t = \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3$, where t_1 is not a value. But then by induction hypothesis t_1 is not in normal form, and there is some t_1' such that $t_1 \rightarrow t_1'$. Thus, t is not in normal form.

4.3 Stuck Terms

A closed term is *stuck* if it is in normal form but it is not a value.

In other words, no evaluation rules apply to a stuck term, yet, the term is not reduced to a value. These are terms that "confuses" the operational semantics. Stuck terms correspond to *run-time errors* such as segmentation faults. Civilized languages do not have stuck terms.

4.4 Multi-Step Evaluation

The *multi-step evaluation* relation $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$ i.e.,

1. if $t \rightarrow t'$ then $t \rightarrow^* t'$,
2. for any t, $t \rightarrow^* t$,
3. if $t \rightarrow^* t'$ and $t' \rightarrow^* t''$, then $t \rightarrow^* t''$.

In inference notation, we can write

$$\frac{t \rightarrow t'}{t \rightarrow^* t'} \qquad \frac{}{t \rightarrow^* t} \qquad \frac{t \rightarrow^* t' \quad t' \rightarrow^* t''}{t \rightarrow^* t''}$$

Question: If $t \rightarrow^* u$ and $t \rightarrow^* u'$ and u and u' are both normal forms, then $u = u'$. **Solution:** Follows from the fact that evaluation is deterministic.

Question: Prove that for any term t there is a normal form t' such that $t \rightarrow^* t'$.
Solution:

By induction on the size of the terms. If size is 1, then the terms are values, and thus the statement holds trivially. Suppose now that the statement holds for all terms of size up to n and consider a term of size $n+1$. The term must be of the form `if` t_1 `then` t_2 `else` t_3 where t_1, t_2, t_3 have size n or less. By definition, we have $t_{1,2,3} \rightarrow^*$ `true` or $t_{1,2,3} \rightarrow^*$ `false`, thus the term evaluates to a `true` of `false`.