

Small-step semantics; Inductive definitions and proofs

Lecture 2

Thursday, January 28, 2016

1 Small-step operational semantics

At this point we have defined the syntax of our simple arithmetic language. We have some informal, intuitive notion of what programs in this language mean. For example, the program $7 + (4 \times 2)$ should equal 15, and the program $\text{foo} := 6 + 1; 2 \times 3 \times \text{foo}$ should equal 42.

We would like now to define formal semantics for this language.

Operational semantics describe how a program would execute on an abstract machine. A *small-step operational semantics* describe how such an execution in terms of successive reductions of an expression, until we reach a number, which represents the result of the computation.

The state of the abstract machine is usually referred to as a configuration, and for our language it must include two pieces of information:

- a store (aka environment or state), which assigns integer values to variables. During program execution, we will refer to the store to determine the values associated with variables, and also update the store to reflect assignment of new values to variables.
- the expression left to evaluate.

Thus, the domain of stores is functions from **Var** to **Int** (written $\mathbf{Var} \rightarrow \mathbf{Int}$), and the domain of configurations is pairs of expressions and stores.

$$\mathbf{Config} = \mathbf{Exp} \times \mathbf{Store}$$

$$\mathbf{Store} = \mathbf{Var} \rightarrow \mathbf{Int}$$

We will denote configurations using angle brackets. For instance, $\langle (foo + 2) \times (bar + 1), \sigma \rangle$, where σ is a store and $(foo + 2) \times (bar + 1)$ is an expression that uses two variables, *foo* and *bar*.

The small-step operational semantics for our language is a relation $\rightarrow \subseteq \mathbf{Config} \times \mathbf{Config}$ that describes how one configuration transitions to a new configuration. That is, the relation \rightarrow shows us how to evaluate programs, one step at a time. We use infix notation for the relation \rightarrow . That is, given any two configurations $\langle e_1, \sigma_1 \rangle$ and $\langle e_2, \sigma_2 \rangle$, if $(\langle e_1, \sigma_1 \rangle, \langle e_2, \sigma_2 \rangle)$ is in the relation \rightarrow , then we write $\langle e_1, \sigma_1 \rangle \rightarrow \langle e_2, \sigma_2 \rangle$.

For example, we have $\langle (4 + 2) \times y, \sigma \rangle \rightarrow \langle 6 \times y, \sigma \rangle$. That is, we can evaluate the configuration $\langle (4 + 2) \times y, \sigma \rangle$ by one step, to get the configuration $\langle 6 \times y, \sigma \rangle$.

Now defining the semantics of the language boils down to defining the relation \rightarrow that describes the transitions between machine configurations.

One issue here is that the domain of integers is infinite, and so is the domain of expressions. Therefore, there is an infinite number of possible machine configurations, and an infinite number of possible one-step transitions. We need to use a finite description for the infinite set of transitions.

We can compactly describe the transition function \rightarrow using inference rules:

$$\mathbf{VAR} \frac{}{\langle x, \sigma \rangle \rightarrow \langle n, \sigma \rangle} \text{ where } n = \sigma(x)$$

$$\mathbf{LADD} \frac{\langle e_1, \sigma \rangle \rightarrow \langle e'_1, \sigma' \rangle}{\langle e_1 + e_2, \sigma \rangle \rightarrow \langle e'_1 + e_2, \sigma' \rangle}$$

$$\mathbf{RADD} \frac{\langle e_2, \sigma \rangle \rightarrow \langle e'_2, \sigma' \rangle}{\langle n + e_2, \sigma \rangle \rightarrow \langle n + e'_2, \sigma' \rangle}$$

$$\text{ADD} \frac{}{\langle n + m, \sigma \rangle \longrightarrow \langle p, \sigma \rangle} \text{ where } p \text{ is the sum of } n \text{ and } m$$

$$\text{LMUL} \frac{\langle e_1, \sigma \rangle \longrightarrow \langle e'_1, \sigma' \rangle}{\langle e_1 \times e_2, \sigma \rangle \longrightarrow \langle e'_1 \times e_2, \sigma' \rangle} \quad \text{RMUL} \frac{\langle e_2, \sigma \rangle \longrightarrow \langle e'_2, \sigma' \rangle}{\langle n \times e_2, \sigma \rangle \longrightarrow \langle n \times e'_2, \sigma' \rangle}$$

$$\text{MUL} \frac{}{\langle n \times m, \sigma \rangle \longrightarrow \langle p, \sigma \rangle} \text{ where } p \text{ is the product of } n \text{ and } m$$

$$\text{ASG1} \frac{\langle e_1, \sigma \rangle \longrightarrow \langle e'_1, \sigma' \rangle}{\langle x := e_1; e_2, \sigma \rangle \longrightarrow \langle x := e'_1; e_2, \sigma' \rangle} \quad \text{ASG} \frac{}{\langle x := n; e_2, \sigma \rangle \longrightarrow \langle e_2, \sigma[x \mapsto n] \rangle}$$

The meaning of an inference rule is that if the fact above the line holds and the side conditions also hold, then the fact below the line holds. The fact(s) above the line are called premises; the fact below the line is called the conclusion. The rules without premises are axioms; and the rules with premises are inductive rules.

Also, we use the notation $\sigma[x \mapsto n]$ for a store that maps the variable x to integer n , and maps every other variable to whatever σ maps it to. More explicitly, if f is the function $\sigma[x \mapsto n]$, then we have

$$f(y) = \begin{cases} n & \text{if } y = x \\ \sigma(y) & \text{otherwise} \end{cases}$$

2 Using the Semantic Rules

Let's see how we can use these rules. Suppose we want to evaluate expression $(\text{foo} + 2) \times (\text{bar} + 1)$ in a store σ where $\sigma(\text{foo}) = 4$ and $\sigma(\text{bar}) = 3$. That is, we want to find the transition for configuration $\langle (\text{foo} + 2) \times (\text{bar} + 1), \sigma \rangle$. For this, we look for a rule with this form of a configuration in the conclusion. By inspecting the rules, we find that the only matching rule is LMUL, where $e_1 = \text{foo} + 2$, $e_2 = \text{bar} + 1$, but e'_1 is not yet known. We can *instantiate* the rule LMUL, replacing the metavariables e_1 and e_2 with appropriate expressions.

$$\text{LMUL} \frac{\langle \text{foo} + 2, \sigma \rangle \longrightarrow \langle e'_1, \sigma \rangle}{\langle (\text{foo} + 2) \times (\text{bar} + 1), \sigma \rangle \longrightarrow \langle e'_1 \times (\text{bar} + 1), \sigma \rangle}$$

Now we need to show that the premise actually holds and find out what e'_1 is. We look for a rule whose conclusion matches $\langle \text{foo} + 2, \sigma \rangle \longrightarrow \langle e'_1, \sigma \rangle$. We find that LADD is the only matching rule:

$$\text{LADD} \frac{\langle \text{foo}, \sigma \rangle \longrightarrow \langle e''_1, \sigma \rangle}{\langle \text{foo} + 2, \sigma \rangle \longrightarrow \langle e''_1 + 2, \sigma \rangle}$$

where $e'_1 = e''_1 + 2$. We repeat this reasoning for $\langle \text{foo}, \sigma \rangle \longrightarrow \langle e''_1, \sigma \rangle$, and we find that the only applicable rule is the axiom VAR:

$$\text{VAR} \frac{}{\langle \text{foo}, \sigma \rangle \longrightarrow \langle 4, \sigma \rangle}$$

because we have $\sigma(\text{foo}) = 4$. Since this is an axiom and has no premises, there is nothing left to prove. Hence, $e''_1 = 4$ and $e'_1 = 4 + 2$. We can put together the above pieces and build the following proof:

$$\text{LMUL} \frac{\text{LADD} \frac{\text{VAR} \frac{}{\langle \text{foo}, \sigma \rangle \longrightarrow \langle 4, \sigma \rangle}}{\langle \text{foo} + 2, \sigma \rangle \longrightarrow \langle 4 + 2, \sigma \rangle}}{\langle (\text{foo} + 2) \times (\text{bar} + 1), \sigma \rangle \longrightarrow \langle (4 + 2) \times (\text{bar} + 1), \sigma \rangle}}$$

This proves that, given our inference rules, the one-step transition $\langle (\text{foo} + 2) \times (\text{bar} + 1), \sigma \rangle \longrightarrow \langle (4 + 2) \times (\text{bar} + 1), \sigma \rangle$ is possible. The above proof structure is called a “proof tree” or “derivation”. It is important to keep in mind that proof trees must be finite for the conclusion to be valid.

We can use a similar reasoning to find out the next evaluation step:

$$\text{LMUL} \frac{\text{ADD} \frac{}{\langle 4 + 2, \sigma \rangle \longrightarrow \langle 6, \sigma \rangle}}{\langle (4 + 2) \times (\text{bar} + 1), \sigma \rangle \longrightarrow \langle 6 \times (\text{bar} + 1), \sigma \rangle}}$$

And we can continue this process. At the end, we can put together all of these transitions, to get a view of the entire computation:

$$\begin{aligned} \langle (\text{foo} + 2) \times (\text{bar} + 1), \sigma \rangle &\longrightarrow \langle (4 + 2) \times (\text{bar} + 1), \sigma \rangle \\ &\longrightarrow \langle 6 \times (\text{bar} + 1), \sigma \rangle \\ &\longrightarrow \langle 6 \times (3 + 1), \sigma \rangle \\ &\longrightarrow \langle 6 \times 4, \sigma \rangle \\ &\longrightarrow \langle 24, \sigma \rangle \end{aligned}$$

The result of the computation is a number, 24. The machine configuration that contains the final result is the point where the evaluation stops; they are called *final configurations*. For our language of expressions, the final configurations are of the form $\langle n, \sigma \rangle$ where n is a number and σ is a store.

We write \longrightarrow^* for the reflexive transitive closure of the relation \longrightarrow . That is, if $\langle e, \sigma \rangle \longrightarrow^* \langle e', \sigma' \rangle$, then using zero or more steps, we can evaluate the configuration $\langle e, \sigma \rangle$ to the configuration $\langle e', \sigma' \rangle$. Thus, we can write

$$\langle (\text{foo} + 2) \times (\text{bar} + 1), \sigma \rangle \longrightarrow^* \langle 24, \sigma \rangle.$$

3 Expressing Program Properties

Now that we have defined our small-step operational semantics, we can formally express different properties of programs. For instance:

- **Progress:** For each store σ and expression e that is not an integer, there exists a possible transition for $\langle e, \sigma \rangle$:

$$\forall e \in \mathbf{Exp}. \forall \sigma \in \mathbf{Store}. \text{either } e \in \mathbf{Int} \text{ or } \exists e', \sigma'. \langle e, \sigma \rangle \longrightarrow \langle e', \sigma' \rangle$$

- **Termination:** The evaluation of each expression terminates:

$$\forall e \in \mathbf{Exp}. \forall \sigma_0 \in \mathbf{Store}. \exists \sigma \in \mathbf{Store}. \exists n \in \mathbf{Int}. \langle e, \sigma_0 \rangle \longrightarrow^* \langle n, \sigma \rangle$$

- **Deterministic Result:** The evaluation result for any expression is deterministic:

$$\forall e \in \mathbf{Exp}. \forall \sigma_0, \sigma, \sigma' \in \mathbf{Store}. \forall n, n' \in \mathbf{Int}.$$

$$\text{if } \langle e, \sigma_0 \rangle \longrightarrow^* \langle n, \sigma \rangle \text{ and } \langle e, \sigma_0 \rangle \longrightarrow^* \langle n', \sigma' \rangle \text{ then } n = n' \text{ and } \sigma = \sigma'.$$

How can we prove such kinds of properties? *Inductive proofs* allow us to prove statements such as the properties above. We first introduce inductive sets, introduce inductive proofs, and then show how we can prove progress (the first property above) using inductive techniques.

4 Inductive sets

Induction is an important concept in the theory of programming language. We have already seen it used to define language syntax, and to define the small-step operational semantics for the arithmetic language.

An inductively defined set A is a set that is built using a set of axioms and inductive (inference) rules. Axioms of the form

$$\frac{}{a \in A}$$

indicate that a is in the set A . Inductive rules

$$\frac{a_1 \in A \quad \dots \quad a_n \in A}{a \in A}$$

indicate that if a_1, \dots, a_n are all elements of A , then a is also an element of A .

The set A is the set of all elements that can be inferred to belong to A using a (finite) number of applications of these rules, starting only from axioms. In other words, for each element a of A , we must be able to construct a finite proof tree whose final conclusion is $a \in A$.

Example 1. The language of a grammar is an inductive set. For instance, the set of arithmetic expressions can be described with 2 axioms, and 3 inductive rules:

$$\begin{array}{c} \text{VAR} \frac{}{x \in \mathbf{Exp}} \quad x \in \mathbf{Var} \quad \text{INT} \frac{}{n \in \mathbf{Exp}} \quad n \in \mathbf{Int} \\ \\ \text{ADD} \frac{e_1 \in \mathbf{Exp} \quad e_2 \in \mathbf{Exp}}{e_1 + e_2 \in \mathbf{Exp}} \quad \text{MUL} \frac{e_1 \in \mathbf{Exp} \quad e_2 \in \mathbf{Exp}}{e_1 \times e_2 \in \mathbf{Exp}} \quad \text{ASS} \frac{e_1 \in \mathbf{Exp} \quad e_2 \in \mathbf{Exp}}{x := e_1; e_2 \in \mathbf{Exp}} \quad x \in \mathbf{Var} \end{array}$$

This is equivalent to the grammar $e ::= x \mid n \mid e_1 + e_2 \mid e_1 \times e_2 \mid x := e_1; e_2$.

To show that $(\text{foo} + 3) \times \text{bar}$ is an element of the set \mathbf{Exp} , it suffices to show that $\text{foo} + 3$ and bar are in the set \mathbf{Exp} , since the inference rule MUL can be used, with $e_1 \equiv \text{foo} + 3$ and $e_2 \equiv \text{bar}$, and, since if the premises $\text{foo} + 3 \in \mathbf{Exp}$ and $\text{bar} \in \mathbf{Exp}$ are true, then the conclusion $(\text{foo} + 3) \times \text{bar} \in \mathbf{Exp}$ is true.

Similarly, we can use rule ADD to show that if $\text{foo} \in \mathbf{Exp}$ and $3 \in \mathbf{Exp}$, then $(\text{foo} + 3) \in \mathbf{Exp}$. We can use axiom VAR (twice) to show that $\text{foo} \in \mathbf{Exp}$ and $\text{bar} \in \mathbf{Exp}$ and rule INT to show that $3 \in \mathbf{Exp}$. We can put these all together into a derivation whose conclusion is $(\text{foo} + 3) \times \text{bar} \in \mathbf{Exp}$:

$$\text{MUL} \frac{\text{ADD} \frac{\text{VAR} \frac{}{\text{foo} \in \mathbf{Exp}} \quad \text{INT} \frac{}{3 \in \mathbf{Exp}}}{(\text{foo} + 3) \in \mathbf{Exp}} \quad \text{VAR} \frac{}{\text{bar} \in \mathbf{Exp}}}{(\text{foo} + 3) \times \text{bar} \in \mathbf{Exp}}$$

Example 2. The natural numbers can be inductively defined:

$$\frac{}{0 \in \mathbb{N}} \quad \frac{n \in \mathbb{N}}{\text{succ}(n) \in \mathbb{N}}$$

where $\text{succ}(n)$ is the successor of n .

Example 3. The small-step evaluation relation \longrightarrow is an inductively defined set. The definition of this set is given by the semantic rules.

Example 4. The transitive, reflexive closure \longrightarrow^* (i.e., the multi-step evaluation relation) can be inductively defined:

$$\frac{}{\langle e, \sigma \rangle \longrightarrow^* \langle e, \sigma \rangle} \quad \frac{\langle e, \sigma \rangle \longrightarrow \langle e', \sigma' \rangle \quad \langle e', \sigma' \rangle \longrightarrow^* \langle e'', \sigma'' \rangle}{\langle e, \sigma \rangle \longrightarrow^* \langle e'', \sigma'' \rangle}$$

5 Inductive proofs

We can prove facts about elements of an inductive set using an inductive reasoning that follows the structure of the set definition.

5.1 Mathematical induction

You have probably seen proofs by induction over the natural numbers, called *mathematical induction*. In such proofs, we typically want to prove that some property P holds for all natural numbers, that is, $\forall n \in \mathbb{N}. P(n)$. A proof by induction works by first proving that $P(0)$ holds, and then proving for all $m \in \mathbb{N}$, if $P(m)$ then $P(m + 1)$. The principle of mathematical induction can be stated succinctly as

$$P(0) \text{ and } (\forall m \in \mathbb{N}. P(m) \implies P(m + 1)) \implies \forall n \in \mathbb{N}. P(n).$$

The assertion that $P(0)$ is the *basis* of the induction (also called the *base case*). Establishing that $P(m) \implies P(m + 1)$ is called *inductive step*, or the *inductive case*. While proving the inductive step, the assumption that $P(m)$ holds is called the *inductive hypothesis*.

5.2 Structural induction

Given an inductively defined set A , to prove that property P holds for all elements of A , we need to show:

1. **Base cases:** For each axiom

$$\frac{}{a \in A},$$

$P(a)$ holds.

2. **Inductive cases:** For each inference rule

$$\frac{a_1 \in A \quad \dots \quad a_n \in A}{a \in A},$$

if $P(a_1)$ and \dots and $P(a_n)$ then $P(a)$.

If the set A is the set of natural numbers (see Example 2 above), then the requirements given above for proving that P holds for all elements of A are equivalent to mathematical induction.

If A describes a syntactic set, then we refer to induction following the requirements above as *structural induction*. If A is an operational semantics relation (such as the small-step operational semantics relation \longrightarrow) then such induction is called *induction on derivations*. We will see examples of structural induction and induction on derivations throughout the course.

In the next lecture, we will consider some examples of proving properties of inductive sets using inductive proofs.