Perhaps the single most important proof technique in the theory of programming languages is *mathematical induction*. Today we'll look at how to define the semantics of ARITH, how we need induction to do that, and how we can use induction to prove theorems about it.

1 Operational Semantics

Last time we saw a simple arithmetic language ARITH. Recall the syntax for ARITH.

$$e \quad ::= \quad x \mid n \mid e_1 + e_2 \mid e_1 * e_2 \mid x := e_1 \,;\, e_2$$

We have an intuitive notion of what expressions mean. For example, $3 + (4 * 2)$ evaluates to 11, and $i := 6 + 1 \,;\, 2 * 3 * i$ evaluates to 42. We will now make this intuition precise.

An *operational semantics* describes how a program executes as an abstract machine. A *small-step* operational semantics describes how such an execution proceeds in terms of successive reductions of terms in the language—here expressions—until we eventually reach a value that is the result of the computation. The state of an abstract machine is often called a *configuration*. To handle variables, our configurations will include two parts.

- The *expression* to evaluate.
- a *store* (also known as an environment, state, or valuation), which maps variables to integers. Note that not all variables need to be mapped at a given them. The store is often denoted by σ.

More formally,

$$\begin{aligned} \textbf{Store} &\triangleq \textbf{Var} \rightharpoonup \textbf{Int} \\ \textbf{Config} &\triangleq \textbf{Exp} \times \textbf{Store} \end{aligned}$$

We will denote configurations using angle brackets. For instance, $\langle (x+2) * (y+3), \sigma \rangle$ is the configuration where $(x + 2) * (y + 3)$ is the expression and σ is the store. The small step operational semantics for our language is then a binary relation $\longrightarrow \subseteq \textbf{Config} \times \textbf{Config}$. We will use infix notation for this relation, so if $(\langle e_1, \sigma_1 \rangle, \langle e_2, \sigma_2 \rangle) \in \longrightarrow$, we will write $\langle e_1, \sigma_1 \rangle \longrightarrow \langle e_2, \sigma_2 \rangle$.

We define the small step operational semantics compactly using a set *inference rules*.

$$\textsc{Var}\ \frac{\sigma(x) = n}{\langle x, \sigma \rangle \longrightarrow \langle n, \sigma \rangle}$$

$$\textsc{Add}\ \frac{m = n_1 + n_2}{\langle n_1 + n_2, \sigma \rangle \longrightarrow \langle m, \sigma \rangle} \qquad \textsc{LAdd}\ \frac{\langle e_1, \sigma \rangle \longrightarrow \langle e_1', \sigma' \rangle}{\langle e_1 + e_2, \sigma \rangle \longrightarrow \langle e_1' + e_2, \sigma' \rangle} \qquad \textsc{RAdd}\ \frac{\langle e_2, \sigma \rangle \longrightarrow \langle e_2', \sigma' \rangle}{\langle n + e_2, \sigma \rangle \longrightarrow \langle n + e_2', \sigma' \rangle}$$

$$\textsc{Mul}\ \frac{m = n_1 \times n_2}{\langle n_1 * n_2, \sigma \rangle \longrightarrow \langle m, \sigma \rangle} \qquad \textsc{LMul}\ \frac{\langle e_1, \sigma \rangle \longrightarrow \langle e_1', \sigma' \rangle}{\langle e_1 * e_2, \sigma \rangle \longrightarrow \langle e_1' * e_2, \sigma' \rangle} \qquad \textsc{RMul}\ \frac{\langle e_2, \sigma \rangle \longrightarrow \langle e_2', \sigma' \rangle}{\langle n * e_2, \sigma \rangle \longrightarrow \langle n * e_2', \sigma' \rangle}$$

$$\textsc{Asgn1}\ \frac{\langle e_1, \sigma \rangle \longrightarrow \langle e_1', \sigma' \rangle}{\langle x := e_1 \,;\, e_2, \sigma \rangle \longrightarrow \langle x := e_1' \,;\, e_2, \sigma' \rangle} \qquad \textsc{Asgn2}\ \frac{}{\langle x := n \,;\, e_2, \sigma \rangle \longrightarrow \langle e_2, \sigma[x \mapsto n] \rangle}$$

The meaning of an inference rule is that if the facts above the line holds, then the fact below the line holds. The fact 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.

The notation $\sigma[x \mapsto n]$ denotes the (partial) function that behaves exactly like σ except on x, where it returns n. It doesn't matter if σ is defined on x or not. If $\sigma(x)$ was previously defined, it is replaced, and if it was previously undefined, it is added. That is, $f = \sigma[x \mapsto n]$ if

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

2 Inductive Definitions

The semantics described above appears to reference itself. Is that allowed? The short answer is "yes," but we need to unpack exactly what this inductive definition means.

An *inductively-defined* set A is one that is described using a finite collection of axioms and inductive (inference) rules. Axioms of the form

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

indicate that a is in the set A. Inductive rules

$$\frac{a_1 \in A \qquad \cdots \qquad a_n \in A}{a \in A}$$

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

Rather than providing an immediate definition, these inference rules lay out a set of requirements that A must satisfy. Loosely speaking, we can prove that some value a is in the set A if we can do so using any *finite* number of applications of these rules. The finite requirement means we must "bottom out" at an axiom (rather than an inductive rule) eventually. Then A is the set of elements that we can prove are in A in this manner.

Viewing the $\longrightarrow$ binary relation on two configurations as a subset of **Config** $\times$ **Config**, that is how the definition above works.

Formally defining that set is beyond the scope of this lecture.

Here are some other examples of inductive sets

Example 1. The natural numbers $\mathbb{N}$ can be defined as an inductive set from the rules

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

Example 2. The syntax of ARITH using the following set of inference rules.

$$\frac{}{x \in \textbf{Exp}} \qquad\qquad \frac{}{n \in \textbf{Exp}}$$

$$\frac{e_1 \in \textbf{Exp} \qquad e_2 \in \textbf{Exp}}{e_1 + e_2 \in \textbf{Exp}} \qquad \frac{e_1 \in \textbf{Exp} \qquad e_2 \in \textbf{Exp}}{e_1 * e_2 \in \textbf{Exp}} \qquad \frac{e_1 \in \textbf{Exp} \qquad e_2 \in \textbf{Exp}}{x := e_1 \,;\, e_2 \in \textbf{Exp}}$$

These axioms describe the same set of expressions as the BNF grammar in Section 1.

Example 3. The multi-step relation $\longrightarrow^*$—taking zero or more steps—can be inductively defined.

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

3 Inductive Proofs

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

3.1 Mathematical Induction

The principle of mathematical induction is usually described over natural numbers. In such proofs, to prove that some proposition P holds for all natural numbers n, we prove that $P(0)$ holds and whenever $P(n)$ holds then $P(n+1)$ also holds. The principle of induction says that those two are enough to prove that $\forall n \in \mathbb{N}.\, P(n)$. More formally,

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

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

3.2 Structural Induction

A generalized version of mathematical induction is *structural induction*, which allows us to prove properties of inductively-defined sets. If we want to prove that some predicate P holds for all ements of an inductively-defined set A—that is, $\forall a \in A.\, P(a)$—we can do it by following the *structure* of the inductive definition. Specifically, for each rule, we must show that P holds for the conclusion whenever it holds for each of the inductive premises. The axioms serve as the bases cases for this induction, as they have no inductive premises. That is:

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

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

   show that $P(a)$ holds.

2. **Inductive Case:** For each inductive rule

   $$\frac{a_1 \in A \qquad \cdots \qquad a_n \in A}{a \in A}$$

   show that, if $P(a_1), \ldots, P(a_n)$, then $P(a)$.

If we view the natural numbers as an inductive set, as in Example 1, we see that mathematical induction is actually a special case of a more general property called structural induction. The only axiom is $\frac{}{0 \in \mathbb{N}}$, so we must show $P(0)$. The only inductive rule is $\frac{n \in \mathbb{N}}{succ(n) \in \mathbb{N}}$, so we must show that $P(n) \implies P(succ(n)) = P(n+1)$ for any $n \in \mathbb{N}$. Those two are then sufficient to show that $\forall n \in \mathbb{N}.\, P(n)$. We have recovered the principle of mathematical induction stated above.

4 Using Induction to Prove Program Properties

We can use the same structure to prove properties about programs in ARITH. We may wish, for instance, to prove that every ARITH expression produces an integer. We could phrase that as the following soundness theorem.

Theorem 1 (Soundness)**.** *Evaluation of expressions yields an integer.*

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

Unfortunately, this theorem is false. Consider $e = x + y$ and $\sigma = \emptyset$. This program cannot step at all with this store, but it is not an integer. The problem here is *free variables*. There are variables in e that have no explicit assignment, nor do they have values in σ.

To get a bound on what variables we have to worry about, we can compute the free variables of e as follows.

$$\begin{aligned}
\mathrm{FV}(n) &= \emptyset \\
\mathrm{FV}(x) &= \{x\} \\
\mathrm{FV}(e_1 + e_2) &= \mathrm{FV}(e_1) \cup \mathrm{FV}(e_2) \\
\mathrm{FV}(e_1 * e_2) &= \mathrm{FV}(e_1) \cup \mathrm{FV}(e_2) \\
\mathrm{FV}(x := e_1\ ;\ e_2) &= \mathrm{FV}(e_1) \cup (\mathrm{FV}(e_2) - \{x\})
\end{aligned}$$

Note that this definition of $\mathrm{FV}(e)$ is also self-referential! We can this of $\mathrm{FV}(e)$ as an inductive set. We could also think of $\mathrm{FV}(e)$ as a *recursive procedure* that computes a set by recursion on **Exp**. These two views are deeply linked to each other. Induction is the means we use to construct inductive data, and recursion is what we use to examine and compute on it!

This set is sufficient to get us something useful. In particular, we can use it to formulate two properties that are useful for proving the soundness theorem above.

Theorem 2 (Progress)**.** *For any expression e and store σ, if the free variables of e are all contained in σ, then either e is already an integer or $\langle e, \sigma\rangle$ can take a step.*

$$\forall e \in \textbf{Exp}.\, \forall \sigma \in \textbf{Store}.\, \mathrm{FV}(e) \subseteq \mathrm{dom}(\sigma) \implies e \in \textbf{Int} \textit{ or } \exists e' \in \textbf{Exp}.\, \exists \sigma' \in \textbf{Store}.\, \langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle$$

Theorem 3 (Preservation)**.** *Evaluation preserves containment of free variables in the domain of the store,*

$$\forall e, e' \in \textbf{Exp}.\, \forall \sigma, \sigma' \in \textbf{Store}.\, \mathrm{FV}(e) \subseteq \mathrm{dom}(\sigma) \textit{ and } \langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle \implies \mathrm{FV}(e') \subseteq \mathrm{dom}(\sigma')$$

As an example, we will now prove Progress (Theorem 2) by induction on e.

Proof of Theorem 2. Let e be an expression. We will prove by structural induction on e that

$$\forall \sigma \in \textbf{Store}.\, \mathrm{FV}(e) \subseteq \mathrm{dom}(\sigma) \implies e \in \textbf{Int} \text{ or } \exists e' \in \textbf{Exp}.\, \exists \sigma' \in \textbf{Store}.\, \langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle$$

Case $e = n$**:** Immediately $e \in \textbf{Int}$.

Case $e = x$**:** Here $\mathrm{FV}(e) = \mathrm{FV}(x) = \{x\}$. Therefore, by the fact that $\mathrm{FV}(e) \subseteq \mathrm{dom}(\sigma)$, $x \in \mathrm{dom}(\sigma)$ and thus $\sigma(x) = m$ for some m. VAR Therefore applies to show that $\langle e, \sigma\rangle \longrightarrow \langle m, \sigma\rangle$.

Case $e = e_1 + e_2$**:** The principle of induction allows us to assume that the result holds for both e_1 and e_2, giving us two inductive hypotheses. By definition, $\mathrm{FV}(e) = \mathrm{FV}(e_1) \cup \mathrm{FV}(e_2)$, so therefore if σ, then $\mathrm{FV}(e_1), \mathrm{FV}(e_2) \subseteq \mathrm{dom}(\sigma)$ as well. We now know by application of the first inductive hypothesis that either $e_1 \in \textbf{Int}$ or e_1 steps. We do a sub-case analysis on these two options.

If e_1 steps, that is $\exists e_1', \sigma'$ such that $\langle e_1, \sigma\rangle \longrightarrow \langle e_1', \sigma'\rangle$, then rule LADD applies and $\langle e, \sigma\rangle \longrightarrow \langle e_1' + e_2, \sigma'\rangle$.

If $e_1 \in \textbf{Int}$, then we apply the second inductive hypothesis to show that either $e_2 \in \textbf{Int}$ or e_2 steps. If $e_2 \in \textbf{Int}$, then rule ADD applies and there is some m such that $m = e_1 + e_2$ and $\langle e, \sigma\rangle \longrightarrow \langle m, \sigma\rangle$. If e_2 steps, that is $\exists e_2', \sigma'$ such that $\langle e_2, \sigma\rangle \longrightarrow \langle e_2', \sigma'\rangle$, then rule RADD applies.

Case $e = e_1 * e_2$**:** This case is identical to the previous case, but using MUL, LMUL, and RMUL instead of the addition rules.

Case $e = (x := e_1 \; ; e_2)$**:** As in the previous two cases, there are two inductive hypotheses, one for e_1 and one for e_2. We again know that $\mathrm{FV}(e_1) \subseteq \mathrm{dom}(\sigma)$ by the same argument as before and again apply the first inductive hypothesis.

If $e_1 \in \mathbf{Int}$, then ASGN2 applies and $\langle e, \sigma \rangle \longrightarrow \langle e_2, \sigma[x \mapsto e_1] \rangle$. If there exist e_1', σ' such that $\langle e_1, \sigma \rangle \longrightarrow \langle e_1', \sigma' \rangle$, then ASGN1 applies, and $\langle e, \sigma \rangle \longrightarrow \langle x := e_1' \; ; e_2, \sigma' \rangle$.

This completes all cases for e, so the principle of induction tells us that the theorem holds. □