



In this lecture, we will use the semantics of our simple language of arithmetic expressions,

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

to express useful program properties, and we will prove these properties by induction.

## 1 Program Properties

There are a number of interesting questions about a language one can ask: Is it deterministic? Are there non-terminating programs? What sorts of errors can arise during evaluation? Having a formal semantics allows us to express these properties precisely.

- **Determinism:** Evaluation is deterministic,

$$\forall e \in \mathbf{Exp}. \forall \sigma, \sigma', \sigma'' \in \mathbf{Store}. \forall e', e'' \in \mathbf{Exp}. \\ \text{if } \langle \sigma, e \rangle \rightarrow \langle \sigma', e' \rangle \text{ and } \langle \sigma, e \rangle \rightarrow \langle \sigma'', e'' \rangle \text{ then } e' = e'' \text{ and } \sigma' = \sigma''.$$

- **Termination:** Evaluation of every expression terminates,

$$\forall e \in \mathbf{Exp}. \forall \sigma \in \mathbf{Store}. \exists \sigma' \in \mathbf{Store}. \exists e' \in \mathbf{Exp}. \langle \sigma, e \rangle \rightarrow^* \langle \sigma', e' \rangle \text{ and } \langle \sigma', e' \rangle \not\rightarrow,$$

where  $\langle \sigma', e' \rangle \not\rightarrow$  is shorthand for  $\neg(\exists \sigma'' \in \mathbf{Store}. \exists e'' \in \mathbf{Exp}. \langle \sigma', e' \rangle \rightarrow \langle \sigma'', e'' \rangle)$ .

It is tempting to want the following soundness property,

- **Soundness:** Evaluation of every expression yields an integer,

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

but unfortunately it does not hold in our language! For example, consider the totally-undefined function  $\sigma$  and the expression  $i + j$ . The configuration  $\langle \sigma, i + j \rangle$  is *stuck*—it has no possible transitions—but  $i + j$  is not an integer. The problem is that  $i + j$  has *free variables* but  $\sigma$  does not contain mappings for those variables.

To fix this problem, we can restrict our attention to *well-formed* configurations  $\langle \sigma, e \rangle$ , where  $\sigma$  is defined on (at least) the free variables in  $e$ . This makes sense as evaluation typically starts with a *closed* expression. We can define the set of free variables of an expression as follows:

$$\begin{aligned} fvs(x) &\triangleq \{x\} \\ fvs(n) &\triangleq \{\} \\ fvs(e_1 + e_2) &\triangleq fvs(e_1) \cup fvs(e_2) \\ fvs(e_1 * e_2) &\triangleq fvs(e_1) \cup fvs(e_2) \\ fvs(x := e_1 ; e_2) &\triangleq fvs(e_1) \cup (fvs(e_2) \setminus \{x\}) \end{aligned}$$

Now we can formulate two properties that imply a variant of the soundness property above:

- **Progress:** For each expression  $e$  and store  $\sigma$  such that the free variables of  $e$  are contained in the domain of  $\sigma$ , either  $e$  is an integer or there exists a possible transition for  $\langle \sigma, e \rangle$ ,

$$\forall e \in \mathbf{Exp}. \forall \sigma \in \mathbf{Store}. \\ fvs(e) \subseteq dom(\sigma) \implies e \in \mathbf{Int} \text{ or } (\exists e' \in \mathbf{Exp}. \exists \sigma' \in \mathbf{Store}. \langle \sigma, e \rangle \rightarrow \langle \sigma', e' \rangle)$$

- **Preservation:** Evaluation preserves containment of free variables in the domain of the store,

$$\forall e, e' \in \mathbf{Exp}. \forall \sigma, \sigma' \in \mathbf{Store}. \\ fvs(e) \subseteq dom(\sigma) \text{ and } \langle \sigma, e \rangle \rightarrow \langle \sigma', e' \rangle \implies fvs(e') \subseteq dom(\sigma').$$

The rest of this lecture shows how can we prove such properties using induction.

## 2 Inductive sets

Induction is an important concept in programming language theory. 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

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

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 set described by a grammar is an inductive set. For instance, the set of arithmetic expressions can be described with two axioms and three inference rules:

$$\begin{array}{c} \frac{}{x \in \mathbf{Exp}} \qquad \frac{}{n \in \mathbf{Exp}} \\[10pt] \frac{e_1 \in \mathbf{Exp} \quad e_2 \in \mathbf{Exp}}{e_1 + e_2 \in \mathbf{Exp}} \qquad \frac{e_1 \in \mathbf{Exp} \quad e_2 \in \mathbf{Exp}}{e_1 * e_2 \in \mathbf{Exp}} \qquad \frac{e_1 \in \mathbf{Exp} \quad e_2 \in \mathbf{Exp}}{x := e_1 ; e_2 \in \mathbf{Exp}} \end{array}$$

These axioms and rules describe the same set of expressions as the grammar:

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

**Example 2.** The natural numbers (expressed here in unary notation) can be inductively defined:

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

**Example 3.** The small-step evaluation relation  $\rightarrow$  is an inductively defined set.

**Example 4.** The multi-step evaluation relation can be inductively defined:

$$\frac{}{\langle \sigma, e \rangle \rightarrow^* \langle \sigma, e \rangle} \text{REFL} \quad \frac{\langle \sigma, e \rangle \rightarrow \langle \sigma', e' \rangle \quad \langle \sigma', e' \rangle \rightarrow^* \langle \sigma'', e'' \rangle}{\langle \sigma, e \rangle \rightarrow^* \langle \sigma'', e'' \rangle} \text{TRANS}$$

**Example 5.** The set of free variables of an expression  $e$  can be inductively defined:

$$\begin{array}{c} \frac{}{y \in fvs(y)} \quad \frac{y \in fvs(e_1)}{y \in fvs(e_1 + e_2)} \quad \frac{y \in fvs(e_2)}{y \in fvs(e_1 + e_2)} \quad \frac{y \in fvs(e_1)}{y \in fvs(e_1 * e_2)} \quad \frac{y \in fvs(e_2)}{y \in fvs(e_1 * e_2)} \\[10pt] \frac{y \in fvs(e_1)}{y \in fvs(x := e_1 ; e_2)} \quad \frac{y \neq x \quad y \in fvs(e_2)}{y \in fvs(x := e_1 ; e_2)} \end{array}$$

### 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

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 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

Given an inductively defined set  $A$ , to prove that a 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  $P(a)$ .

Note that if the set  $A$  is the set of natural numbers from Example 2 above, then the requirements for proving that  $P$  holds for all elements of  $A$  is 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  $\rightarrow$ ) then such an induction is called *induction on derivations*. We will see examples of structural induction and induction on derivations throughout the course.

### 3.3 Example: Progress

Let's consider the progress property defined above, and repeated here:

**Progress:** For each store  $\sigma$  and expression  $e$  such that the free variables of  $e$  are contained in the domain of  $\sigma$ , either  $e$  is an integer or there exists a possible transition for  $\langle \sigma, e \rangle$ :

$$\forall e \in \mathbf{Exp}. \forall \sigma \in \mathbf{Store}. fvs(e) \subseteq dom(\sigma) \implies e \in \mathbf{Int} \text{ or } (\exists e' \in \mathbf{Exp}. \exists \sigma' \in \mathbf{Store}. \langle \sigma, e \rangle \rightarrow \langle \sigma', e' \rangle)$$

Let's rephrase this property in terms of an explicit predicate on expressions:

$$P(e) \triangleq \forall \sigma \in \mathbf{Store}. fvs(e) \subseteq dom(\sigma) \implies e \in \mathbf{Int} \text{ or } (\exists e', \sigma'. \langle \sigma, e \rangle \rightarrow \langle \sigma', e' \rangle)$$

The idea is to build a proof that follows the inductive structure given by the grammar:

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

This technique is called “structural induction on  $e$ .” We analyze each case in the grammar and show that  $P(e)$  holds for that case. Since the grammar productions  $e_1 + e_2$  and  $e_1 * e_2$  and  $x := e_1 ; e_2$  are inductive, they are inductive steps in the proof; the cases for  $x$  and  $n$  are base cases. The proof proceeds as follows.

*Proof.* Let  $e$  be an expression. We will prove that

$$\forall \sigma \in \mathbf{Store}. fvs(e) \subseteq dom(\sigma) \implies e \in \mathbf{Int} \text{ or } (\exists e', \sigma'. \langle \sigma, e \rangle \rightarrow \langle \sigma', e' \rangle)$$

by structural induction on  $e$ . We analyze several cases, one for each case in the grammar for expressions:

**Case  $e = x$ :** Let  $\sigma$  be an arbitrary store, and assume that  $fvs(e) \subseteq dom(\sigma)$ . By the definition of  $fvs$  we have  $fvs(x) = \{x\}$ . By assumption we have  $\{x\} \subseteq dom(\sigma)$  and so  $x \in dom(\sigma)$ . Let  $n = \sigma(x)$ .

By the VAR axiom we have  $\langle \sigma, x \rangle \rightarrow \langle \sigma, n \rangle$ , which finishes the case.

**Case  $e = n$ :** We immediately have  $e \in \mathbf{Int}$ , which finishes the case.

**Case  $e = e_1 + e_2$ :** Let  $\sigma$  be an arbitrary store, and assume that  $fvs(e) \subseteq dom(\sigma)$ . We will assume that  $P(e_1)$  and  $P(e_2)$  hold and show that  $P(e)$  holds. Let's expand these properties. We have

$$P(e_1) = \forall \sigma \in \mathbf{Store}. fvs(e_1) \subseteq dom(\sigma) \implies e_1 \in \mathbf{Int} \text{ or } (\exists e', \sigma'. \langle \sigma, e_1 \rangle \rightarrow \langle \sigma', e' \rangle)$$

$$P(e_2) = \forall \sigma \in \mathbf{Store}. fvs(e_2) \subseteq dom(\sigma) \implies e_2 \in \mathbf{Int} \text{ or } (\exists e', \sigma'. \langle \sigma, e_2 \rangle \rightarrow \langle \sigma', e' \rangle)$$

and want to prove:

$$P(e_1 + e_2) = \forall \sigma \in \mathbf{Store}. fvs(e_1 + e_2) \subseteq dom(\sigma) \implies e_1 + e_2 \in \mathbf{Int} \text{ or } (\exists e', \sigma'. \langle \sigma, e_1 + e_2 \rangle \rightarrow \langle \sigma', e' \rangle)$$

We analyze several subcases.

**Subcase  $e_1 = n_1$  and  $e_2 = n_2$ :** By rule ADD, we immediately have  $\langle \sigma, n_1 + n_2 \rangle \rightarrow \langle \sigma, p \rangle$ , where  $p = n_1 + n_2$ .

**Subcase  $e_1 \notin \mathbf{Int}$ :** By assumption and the definition of  $fvs$  we have

$$fvs(e_1) \subseteq fvs(e_1 + e_2) \subseteq dom(\sigma)$$

Hence, by the induction hypothesis  $P(e_1)$  we also have  $\langle \sigma, e_1 \rangle \rightarrow \langle \sigma', e' \rangle$  for some  $e'$  and  $\sigma'$ . By rule LADD we have  $\langle \sigma, e_1 + e_2 \rangle \rightarrow \langle \sigma', e' + e_2 \rangle$ .

**Subcase  $e_1 = n_1$  and  $e_2 \notin \mathbf{Int}$ :** By assumption and the definition of  $fvs$  we have

$$fvs(e_2) \subseteq fvs(e_1 + e_2) \subseteq dom(\sigma)$$

Hence, by the induction hypothesis  $P(e_2)$  we also have  $\langle \sigma, e_2 \rangle \rightarrow \langle \sigma', e' \rangle$  for some  $e'$  and  $\sigma'$ . By rule RADD we have  $\langle \sigma, e_1 + e_2 \rangle \rightarrow \langle \sigma', e_1 + e' \rangle$ , which finishes the case.

**Case  $e = e_1 * e_2$ :** . Analogous to the previous case.

**Case  $e = x := e_1 ; e_2$ :** . Let  $\sigma$  be an arbitrary store, and assume that  $fvs(e) \subseteq dom(\sigma)$ . As above, we assume that  $P(e_1)$  and  $P(e_2)$  hold and show that  $P(e)$  holds. Let's expand these properties. We have

$$P(e_1) = \forall \sigma. fvs(e_1) \subseteq dom(\sigma) \implies e_1 \in \mathbf{Int} \text{ or } (\exists e', \sigma'. \langle \sigma, e_1 \rangle \rightarrow \langle \sigma', e' \rangle)$$

$$P(e_2) = \forall \sigma. fvs(e_2) \subseteq dom(\sigma) \implies e_2 \in \mathbf{Int} \text{ or } (\exists e', \sigma'. \langle \sigma, e_2 \rangle \rightarrow \langle \sigma', e' \rangle)$$

and want to prove:

$$P(x := e_1 ; e_2) = x := e_1 ; e_2 \in \mathbf{Int} \text{ or } (\exists e', \sigma'. \langle \sigma, x := e_1 ; e_2 \rangle \rightarrow \langle \sigma', e' \rangle)$$

We analyze several subcases.

**Subcase  $e_1 = n_1$ :** By rule ASSGN we have  $\langle \sigma, x := n_1 ; e_2 \rangle \rightarrow \langle \sigma', e_2 \rangle$  where  $\sigma' = \sigma[x \mapsto n_1]$ .

**Subcase  $e_1 \notin \mathbf{Int}$ :** By assumption and the definition of  $fvs$  we have

$$fvs(e_1) \subseteq fvs(x := e_1 ; e_2) \subseteq dom(\sigma)$$

Hence, by induction hypothesis we also have  $\langle \sigma, e_1 \rangle \rightarrow \langle \sigma', e' \rangle$  for some  $e'$  and  $\sigma'$ . By the rule ASSGN1 we have  $\langle \sigma, x := e_1 ; e_2 \rangle \rightarrow \langle \sigma', x := e'_1 ; e_2 \rangle$ , which finishes the case and the inductive proof.  $\square$