CS 4110 – Programming Languages and Logics
Lecture #3: Inductive definitions and proofs

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,

$$\begin{aligned}&\forall e \in \mathbf{Exp}.\ \forall \sigma, \sigma', \sigma'' \in \mathbf{Store}.\ \forall e', e'' \in \mathbf{Exp}.\\ &\quad \text{if } \langle \sigma, e \rangle \longrightarrow \langle \sigma', e' \rangle \text{ and } \langle \sigma, e \rangle \longrightarrow \langle \sigma'', e'' \rangle \text{ then } e' = e'' \text{ and } \sigma' = \sigma''.\end{aligned}$$

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

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

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

It is tempting to want the following soundness property,

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

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

but unfortunately it does not hold in our language! For example, consider the totally-undefined function σ 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 σ 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 σ 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:

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

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

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

$$\begin{aligned}&\forall e \in \mathbf{Exp}.\ \forall \sigma \in \mathbf{Store}.\\ &\qquad fvs(e) \subseteq dom(\sigma) \implies e \in \mathbf{Int} \text{ or } (\exists e' \in \mathbf{Exp}.\ \exists \sigma' \in \mathbf{Store}.\ \langle \sigma, e \rangle \to \langle \sigma', e' \rangle)\end{aligned}$$

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

$$\begin{aligned}&\forall e, e' \in \mathbf{Exp}.\ \forall \sigma, \sigma' \in \mathbf{Store}.\\ &\qquad fvs(e) \subseteq dom(\sigma) \text{ and } \langle \sigma, e \rangle \to \langle \sigma', e' \rangle \implies fvs(e') \subseteq dom(\sigma').\end{aligned}$$

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

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

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

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:

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

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

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:

$$\frac{}{\langle\sigma, e\rangle \rightarrow^{*} \langle\sigma, e\rangle}\ \text{Refl} \qquad \frac{\langle\sigma, e\rangle \rightarrow \langle\sigma', e'\rangle \qquad \langle\sigma', e'\rangle \rightarrow^{*} \langle\sigma'', e''\rangle}{\langle\sigma, e\rangle \rightarrow^{*} \langle\sigma'', e''\rangle}\ \text{Trans}$$

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

$$\frac{}{y \in \mathit{fvs}(y)} \qquad \frac{y \in \mathit{fvs}(e_1)}{y \in \mathit{fvs}(e_1 + e_2)} \qquad \frac{y \in \mathit{fvs}(e_2)}{y \in \mathit{fvs}(e_1 + e_2)} \qquad \frac{y \in \mathit{fvs}(e_1)}{y \in \mathit{fvs}(e_1 * e_2)} \qquad \frac{y \in \mathit{fvs}(e_2)}{y \in \mathit{fvs}(e_1 * e_2)}$$

$$\frac{y \in \mathit{fvs}(e_1)}{y \in \mathit{fvs}(x := e_1 \,;\, e_2)} \qquad \frac{y \neq x \qquad y \in \mathit{fvs}(e_2)}{y \in \mathit{fvs}(x := e_1 \,;\, e_2)}$$

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

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

The proposition $P(0)$ is the *basis* of the induction (also called the *base case*) while $P(m) \Longrightarrow P(m+1)$ 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
$$\frac{a_1 \in A \quad \ldots \quad a_n \in A}{a \in A},$$
if $P(a_1)$ and … 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 $\longrightarrow$) 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 σ and expression e such that the free variables of e are contained in the domain of σ, either e is an integer or there exists a possible transition for $\langle \sigma, e \rangle$:

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

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

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

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

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

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

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

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

and want to prove:

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

We analyze several subcases.

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

Subcase $e_1 \notin$ **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 $\langle \sigma, e_1 \rangle \longrightarrow \langle \sigma', e' \rangle$ for some e' and σ'. By rule LAdd we have $\langle \sigma, e_1 + e_2 \rangle \longrightarrow \langle \sigma', e' + e_2 \rangle$.

Subcase $e_1 = n_1$ **and** $e_2 \notin$ **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 $\langle \sigma, e_2 \rangle \longrightarrow \langle \sigma', e' \rangle$ for some e' and σ'. By rule RAdd we have $\langle \sigma, e_1 + e_2 \rangle \longrightarrow \langle \sigma', e_1 + e' \rangle$, which finishes the case.

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

Case $e = x := e_1 \,;\, e_2$**:** . Let σ 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

$$P(e_1) = \forall \sigma.\ fvs(e_1) \subseteq dom(\sigma) \Longrightarrow e_1 \in \textbf{Int} \text{ or } (\exists e', \sigma'.\ \langle \sigma, e_1 \rangle \to \langle \sigma', e' \rangle)$$
$$P(e_2) = \forall \sigma.\ fvs(e_2) \subseteq dom(\sigma) \Longrightarrow e_2 \in \textbf{Int} \text{ or } (\exists e', \sigma'.\ \langle \sigma, e_2 \rangle \to \langle \sigma', e' \rangle)$$

and want to prove:

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

We analyze several subcases.

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

Subcase $e_1 \notin$ **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 $\langle \sigma, e_1 \rangle \longrightarrow \langle \sigma', e' \rangle$ for some e' and σ'. By the rule Assgn1 we have $\langle \sigma, x := e_1 \,;\, e_2 \rangle \longrightarrow \langle \sigma', x := e_1' \,;\, e_2 \rangle$, which finishes the case and the inductive proof. □