CS 4110

Programming Languages & Logics

Lecture 3
Inductive Definitions and Proofs

Arithmetic Expressions

Last time we defined a simple language of arithmetic expressions,

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

and a small-step operational semantics, $\langle \sigma, e \rangle \rightarrow \langle \sigma', e' \rangle$.

Arithmetic Expressions

Last time we defined a simple language of arithmetic expressions,

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

and a small-step operational semantics, $\langle \sigma, e \rangle \to \langle \sigma', e' \rangle$.

Example:

Assuming σ is a store that maps *foo* to 4...

$$\dfrac{\dfrac{\dfrac{\sigma(foo) = 4}{\langle \sigma, foo \rangle \to \langle \sigma, 4 \rangle}\ \textsc{Var}}{\langle \sigma, foo + 2 \rangle \to \langle \sigma, 4 + 2 \rangle}\ \textsc{LAdd}}{\langle \sigma, (foo + 2) * (bar + 1) \rangle \to \langle \sigma, (4 + 2) * (bar + 1) \rangle}\ \textsc{LMul}$$

Properties

Today we’ll prove some useful program properties by induction.

Properties

Today we'll prove some useful program properties by induction.

- Determinism: Every configuration has at most one successor.

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

Properties

Today we'll prove some useful program properties by induction.

- Determinism: Every configuration has at most one successor.

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

- Termination: Evaluation of every expression terminates.

$$\begin{aligned}&\forall e \in \textbf{Exp}.\ \forall \sigma \in \textbf{Store}.\ \exists \sigma' \in \textbf{Store}.\ \exists e' \in \textbf{Exp}.\\ &\qquad \langle \sigma, e\rangle \rightarrow^{*} \langle \sigma', e'\rangle \text{ and } \langle \sigma', e'\rangle \not\rightarrow,\end{aligned}$$

  Where $\langle \sigma', e'\rangle \not\rightarrow$ is shorthand for:

$$\neg\,(\exists \sigma'' \in \textbf{Store}.\ \exists e'' \in \textbf{Exp}.\ \langle \sigma', e'\rangle \rightarrow \langle \sigma'', e''\rangle)$$

Soundness

- Soundness: Evaluation of every expression yields an integer.

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

Soundness

It is tempting to try to prove this property.

- Soundness: Evaluation of every expression yields an integer.

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

But it’s not true!

Soundness

It is tempting to try to prove this property.

- Soundness: Evaluation of every expression yields an integer.

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

But it's not true!

Counterexample

If $\sigma = \emptyset$, then $\langle \sigma, x \rangle \not\rightarrow$.

In general, evaluation of an expression can *get stuck*...

Well-Formedness

Idea: Restrict our attention to well-formed configurations $\langle \sigma, e \rangle$, where all the variables that e refers to have values in σ.

Well-Formedness

Idea: Restrict our attention to well-formed configurations $\langle \sigma, e \rangle$, where all the variables that e refers to have values in σ.

Free Variables: Let's define a function $fvs(e)$ *inductively*.

Well-Formedness

Idea: Restrict our attention to well-formed configurations $\langle \sigma, e \rangle$, where all the variables that e refers to have values in σ.

Free Variables: Let's define a function $fvs(e)$ *inductively*.

$$fvs(x) \triangleq \{x\}$$

Well-Formedness

Idea: Restrict our attention to well-formed configurations $\langle \sigma, e \rangle$, where all the variables that e refers to have values in σ.

Free Variables: Let's define a function $fvs(e)$ *inductively*.

$$
\begin{array}{rcl}
fvs(x) & \triangleq & \{x\} \\
fvs(n) & \triangleq & \{\}
\end{array}
$$

Well-Formedness

Idea: Restrict our attention to well-formed configurations $\langle \sigma, e \rangle$, where all the variables that e refers to have values in σ.

Free Variables: Let's define a function $fvs(e)$ *inductively*.

$$\begin{aligned} fvs(x) &\triangleq \{x\} \\ fvs(n) &\triangleq \{\} \\ fvs(e_1 + e_2) &\triangleq fvs(e_1) \cup fvs(e_2) \end{aligned}$$

Well-Formedness

Idea: Restrict our attention to well-formed configurations $\langle \sigma, e \rangle$, where all the variables that e refers to have values in σ.

Free Variables: Let's define a function $fvs(e)$ *inductively*.

$$
\begin{array}{rcl}
fvs(x) & \triangleq & \{x\} \\
fvs(n) & \triangleq & \{\} \\
fvs(e_1 + e_2) & \triangleq & fvs(e_1) \cup fvs(e_2) \\
fvs(e_1 * e_2) & \triangleq & fvs(e_1) \cup fvs(e_2)
\end{array}
$$

Well-Formedness

Idea: Restrict our attention to well-formed configurations $\langle \sigma, e \rangle$, where all the variables that e refers to have values in σ.

Free Variables: Let's define a function $fvs(e)$ *inductively*.

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

Well-Formedness

Idea: Restrict our attention to well-formed configurations $\langle \sigma, e \rangle$, where all the variables that e refers to have values in σ.

Free Variables: Let's define a function $fvs(e)$ *inductively*.

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

Well-Formedness:

A configuration $\langle \sigma, e \rangle$ is *well-formed* if and only if $fvs(e) \subseteq dom(\sigma)$.

Progress and Preservation

Now we can formulate two properties that imply soundness:

- Progress:

$$
\begin{aligned}
&\forall e \in \textbf{Exp}.\ \forall \sigma \in \textbf{Store}. \\
&\quad \langle \sigma, e \rangle \text{ well-formed } \implies \\
&\quad e \in \textbf{Int} \text{ or } (\exists e' \in \textbf{Exp}.\ \exists \sigma' \in \textbf{Store}.\ \langle \sigma, e \rangle \to \langle \sigma', e' \rangle)
\end{aligned}
$$

Progress and Preservation

Now we can formulate two properties that imply soundness:

- Progress:

$$\begin{aligned}&\forall e \in \textbf{Exp}.\ \forall \sigma \in \textbf{Store}.\\ &\quad \langle \sigma, e\rangle \text{ well-formed} \implies\\ &\quad e \in \textbf{Int} \text{ or } (\exists e' \in \textbf{Exp}.\ \exists \sigma' \in \textbf{Store}.\ \langle \sigma, e\rangle \to \langle \sigma', e'\rangle)\end{aligned}$$

- Preservation:

$$\begin{aligned}&\forall e, e' \in \textbf{Exp}.\ \forall \sigma, \sigma' \in \textbf{Store}.\\ &\quad \langle \sigma, e\rangle \text{ well-formed and } \langle \sigma, e\rangle \to \langle \sigma', e'\rangle \implies\\ &\quad \langle \sigma', e'\rangle \text{ well-formed.}\end{aligned}$$

Progress and Preservation

Now we can formulate two properties that imply soundness:

- Progress:

$$\begin{aligned}&\forall e \in \textbf{Exp}.\ \forall \sigma \in \textbf{Store}.\\ &\quad \langle \sigma, e\rangle \text{ well-formed} \implies\\ &\quad e \in \textbf{Int} \text{ or } (\exists e' \in \textbf{Exp}.\ \exists \sigma' \in \textbf{Store}.\ \langle \sigma, e\rangle \to \langle \sigma', e'\rangle)\end{aligned}$$

- Preservation:

$$\begin{aligned}&\forall e, e' \in \textbf{Exp}.\ \forall \sigma, \sigma' \in \textbf{Store}.\\ &\quad \langle \sigma, e\rangle \text{ well-formed and } \langle \sigma, e\rangle \to \langle \sigma', e'\rangle \implies\\ &\quad \langle \sigma', e'\rangle \text{ well-formed.}\end{aligned}$$

How are we going to prove these properties? Induction!

Inductive Sets

Inductive Sets

An *inductively-defined set* A is one that can be described using a finite collection of inference rules:

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

This rules states that if a_1 through a_n are elements of A, then a is also an element of A.

Inductive Set Examples

The small-step evaluation relation we just defined, $\rightarrow$, is an inductive set.

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

$$\frac{\langle \sigma, e_1 \rangle \rightarrow \langle \sigma', e_1' \rangle}{\langle \sigma, e_1 + e_2 \rangle \rightarrow \langle \sigma', e_1' + e_2 \rangle}\ \textsc{LAdd} \qquad \frac{\langle \sigma, e_2 \rangle \rightarrow \langle \sigma', e_2' \rangle}{\langle \sigma, n + e_2 \rangle \rightarrow \langle \sigma', n + e_2' \rangle}\ \textsc{RAdd}$$

$$\frac{p = m + n}{\langle \sigma, n + m \rangle \rightarrow \langle \sigma, p \rangle}\ \textsc{Add} \qquad \frac{\langle \sigma, e_1 \rangle \rightarrow \langle \sigma', e_1' \rangle}{\langle \sigma, e_1 * e_2 \rangle \rightarrow \langle \sigma', e_1' * e_2 \rangle}\ \textsc{LMul}$$

$$\frac{\langle \sigma, e_2 \rangle \rightarrow \langle \sigma', e_2' \rangle}{\langle \sigma, n * e_2 \rangle \rightarrow \langle \sigma', n * e_2' \rangle}\ \textsc{RMul} \qquad \frac{p = m \times n}{\langle \sigma, m * n \rangle \rightarrow \langle \sigma, p \rangle}\ \textsc{Mul}$$

$$\frac{\langle \sigma, e_1 \rangle \rightarrow \langle \sigma', e_1' \rangle}{\langle \sigma, x := e_1 ;\ e_2 \rangle \rightarrow \langle \sigma', x := e_1' ;\ e_2 \rangle}\ \textsc{Assgn1} \qquad \frac{\sigma' = \sigma[x \mapsto n]}{\langle \sigma, x := n ;\ e_2 \rangle \rightarrow \langle \sigma', e_2 \rangle}\ \textsc{Assgn}$$

Inductive Set Examples

Every BNF grammar defines an inductive set.

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

Here are the equivalent inference rules:

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

$$\frac{e_1 \in \textbf{Exp} \qquad e_2 \in \textbf{Exp}}{e_1 + e_2 \in \textbf{Exp}} \qquad \frac{e_1 \in \textbf{Exp} \qquad e_2 \in \textbf{Exp}}{e_1 * e_2 \in \textbf{Exp}}$$

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

Inductive Set Examples

The multi-step evaluation relation is an inductive set.

$$\frac{}{\langle \sigma, e \rangle \to^* \langle \sigma, e \rangle} \textsc{Refl}$$

$$\frac{\langle \sigma, e \rangle \to \langle \sigma', e' \rangle \qquad \langle \sigma', e' \rangle \to^* \langle \sigma'', e'' \rangle}{\langle \sigma, e \rangle \to^* \langle \sigma'', e'' \rangle} \textsc{Trans}$$

Inductive Set Examples

The set of free variables of an expression is an inductive set.

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

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

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

Inductive Set Examples

The natural numbers are an inductive set.

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

Induction Principle

Recall the principle of mathematical induction.

To prove $\forall n.\ P(n)$, we must establish several cases.

- Base case: $P(0)$
- Inductive case: $P(m) \Rightarrow P(m+1)$

Induction Principle

Every inductive set has an analogous principle.

To prove $\forall a.\ P(a)$ we must establish several cases.

- Base cases: $P(a)$ holds for each axiom

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

- Inductive cases: For each non-axiom 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 $P(a)$.

Inductive Proof: a Recipe

1. Choose the inductively-defined set, A, that you want to prove something about.
2. Make up a property P such that, if $\forall a \in A.\ P(a)$, then you'll be happy.
3. Write, using your own property P: *We prove that* $\forall a \in A.\ P(a)$ *by inducting on the structure of A.*
4. Write down a case for each inference rule in the definition of A.
5. Prove each case by writing down the induction hypotheses (P applied to each of the premises) and using them to prove the goal (P applied to the conclusion).
6. QED!

Example: Induction on Natural Numbers

Recall the inductive definition of the natural numbers:

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

To prove $\forall n.\ P(n)$, it suffices to show:

- Base case: $P(0)$
- Inductive case: $P(m) \Rightarrow P(m+1)$

...which is the usual principle of mathematical induction!

Example: Progress

Recall the progress property.

$$\begin{array}{l} \forall e \in \textbf{Exp}.\ \forall \sigma \in \textbf{Store}. \\ \quad \langle \sigma, e \rangle \text{ well-formed } \implies \\ \quad e \in \textbf{Int} \text{ or } (\exists e' \in \textbf{Exp}.\ \exists \sigma' \in \textbf{Store}.\ \langle \sigma, e \rangle \to \langle \sigma', e' \rangle) \end{array}$$

We'll prove this by structural induction on e.

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

$$\frac{e_1 \in \textbf{Exp} \qquad e_2 \in \textbf{Exp}}{e_1 + e_2 \in \textbf{Exp}} \qquad \frac{e_1 \in \textbf{Exp} \qquad e_2 \in \textbf{Exp}}{e_1 * e_2 \in \textbf{Exp}}$$

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