Induction is an important concept in the theory of programming languages – it occurs when defining language syntax, as well as when specifying the execution of repetitive constructs (e.g., loops, recursion).

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

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

specify elements that are by default in the set A. And inductive rules:

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

indicate that a is an element of A, provided that $a_1, ..., a_n$ are elements 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 2 inductive rules:

$$\frac{}{x \in \mathsf{Expr}} \qquad \frac{}{n \in \mathsf{Expr}} \qquad \frac{e_1 \in \mathsf{Expr} \quad e_2 \in \mathsf{Expr}}{e_1 + e_2 \in \mathsf{Expr}} \qquad \frac{e_1 \in \mathsf{Expr} \quad e_2 \in \mathsf{Expr}}{e_1 * e_2 \in \mathsf{Expr}}$$

- Example 2. Non-negative integers can be inductively defined:

$$\frac{}{0 \in \mathsf{NonNegInt}} \qquad \frac{n \in \mathsf{NonNegInt}}{succ(n) \in \mathsf{NonNegInt}}$$

- Example 3: The one-step evaluation relation $\rightarrow$ (regarded as a subset of $\mathsf{Config} \times \mathsf{Config}$) is an inductively defined set. The definition of this set is given by the semantic rules.
- Example 4: The transitive, reflexive closure $\rightarrow^*$ (i.e., the multi-step evaluation relation) can be inductively defined:

$$\frac{}{mc \rightarrow^* mc} \qquad \frac{mc \rightarrow mc'' \quad mc'' \rightarrow^* mc'}{mc \rightarrow^* mc'}$$

1 Inductive proofs

We can prove facts about elements of an inductive set using an inductive reasoning that follows the structure of the set definition. This is called "rule induction". To prove that property P holds for all elements of A:

$$\forall a \in A : P(a)$$

we need to prove:

1. (base cases) $P(a)$ holds for all axioms $\frac{}{a \in A}$
2. (inductive cases) $P(a_1)$ and ... and $P(a_n)$ implies $P(a)$ for all rules: $\frac{a_1 \in A \quad ... \quad a_n \in A}{a \in A}$

A very similar technique is called "induction on derivation", where we reason about height of the proof tree of $a \in A$:

1. (base cases) $P(a)$ holds for all a with $height(a) = 0$
2. (inductive cases) $P(a)$ holds for all a such that $height(a) < n$ implies that $P(a)$ holds for all a such that $height(a) = n$.

If A describes a syntactic set, we refer to rule induction as "structural induction". And if A is the set of non-negative integers, rule induction becomes "mathematical induction".

2 Intro to large-step semantics

So far we have defined the small step evaluation relation $\rightarrow$, and then took its transitive and reflexive closure $\rightarrow^*$ to describe the execution of multiple steps of evaluation. In particular, if mc is some start configuration, and fc is a final configuration, the evaluation $mc \rightarrow^* fc$ shows the result of the computation of mc.

It turns out that there is an alternative way to specify the operational semantics of a language in such a way that it would directly give the final result. This alternative method is called "large-step semantics" (as opposed to the one-step evaluation relation, which provides a "small-step semantics").

We'll use the same configurations as before, but define a large step evaluation relation:

$$\mathsf{Eval} \subseteq \mathsf{Expr} \times \mathsf{Store} \times \mathsf{Int}$$

and will write $\langle e,\, s\rangle \Downarrow n$ to mean that $(e, s, n) \in \mathsf{Eval}$. In other words, e in store s evaluates in one big step directly to n.

The large step semantics boils down to defining the relation $\Downarrow$. We can do that inductively:

$$\frac{}{\langle n,\, s\rangle \Downarrow n}\ \text{(int)}$$

$$\frac{}{\langle x,\, s\rangle \Downarrow s(x)}\ \text{(var)}$$

$$\frac{\langle e_1,\, s\rangle \Downarrow n_1 \quad \langle e_2,\, s\rangle \Downarrow n_2}{\langle e_1 + e_2,\, s\rangle \Downarrow n}\ \text{where } n = n_1 + n_2\ \text{(plus)}$$

$$\frac{\langle e_1,\, s\rangle \Downarrow n_1 \quad \langle e_2,\, s\rangle \Downarrow n_2}{\langle e_1 * e_2,\, s\rangle \Downarrow n}\ \text{where } n = n_1 * n_2\ \text{(mul)}$$

To see how we use these rules, take an example: evaluate $(y + 2) * (x + 1)$ in state $s = \{x = 4, y = 3\}$. We get the following proof tree for the fact that $\langle (y + 2) * (x + 1),\, s\rangle \Downarrow 24$:

$$\dfrac{\dfrac{\dfrac{}{\langle y,\, s\rangle \Downarrow 4} \quad \dfrac{}{\langle 2,\, s\rangle \Downarrow 2}}{\langle y + 2,\, s\rangle \Downarrow 6} \quad \dfrac{\dfrac{}{\langle x,\, s\rangle \Downarrow 3} \quad \dfrac{}{\langle 1,\, s\rangle \Downarrow 1}}{\langle x + 1,\, s\rangle \Downarrow 4}}{\langle (y + 2) * (x + 1),\, s\rangle \Downarrow 24}$$

A closer look to this structure reveals the relation between small step and large-step evaluation: a depth-first traversal of the large-step proof tree yields the sequence of one-step transitions in small-step evaluation.

Equivalence of semantics

So far, we have specified the semantics of our language of arithmetic expressions using two different sets of rules: small-step and large-step. Are they expressing the same meaning of arithmetic expressions? Can we show that they express the same thing?

Equivalence of semantics: For all expressions e, stores s, and integers n, we have:

$$\langle e,\, s\rangle \to^* \langle n,\, s\rangle \text{ iff } \langle e,\, s\rangle \Downarrow n$$

Proof sketch.

$\Leftarrow$ Want to prove that property:

$$P(e) = \forall s, n : \langle e,\, s\rangle \Downarrow n \text{ implies } \langle e,\, s\rangle \to^* \langle n,\, s\rangle$$

holds for all $e \in \mathsf{Expr}$. We can use structural induction on expressions e and examine the following cases.

- Case $e = x$.
  If $\langle x,\, s\rangle \Downarrow n$, then, by looking at the large-step rules, we see that only one rule matches, and that rule requires $n = s(x)$. Then, $\langle x,\, s\rangle \to \langle n,\, s\rangle$ also holds, using a small-step axiom. We conclude that $\langle x,\, s\rangle \to^* \langle n,\, s\rangle$ holds.
- Case $e = n$.
  In this case, $\langle n,\, s\rangle \to^* \langle n,\, s\rangle$ trivially holds because of reflexivity of $\to^*$.
- Case $e = e_1 + e_2$.
  This is an inductive case. We want to prove that, if $P(e_1)$ and $P(e_2)$ hold:

  $$P(e_1) : \langle e_1,\, s\rangle \Downarrow n_1 \text{ implies } \langle e_1,\, s\rangle \to^* \langle n_1,\, s\rangle P(e_2) : \langle e_2,\, s\rangle \Downarrow n_2 \text{ implies } \quad \langle e_2,\, s\rangle \to^* \langle n_1,\, s\rangle$$

  then $P(e)$ also holds:

  $$P(e) : \langle e_1 + e_2,\, s\rangle \Downarrow n \text{ implies } \langle e_1 + e_2,\, s\rangle \to^* \langle n,\, s\rangle$$

  Let's start with the premise in $P(e) : \langle e_1 + e_2,\, s\rangle \Downarrow n$. By inspecting the large-step semantic rules, we see that only one rule applies, and that it must be the case that $\langle e_1,\, s\rangle \Downarrow n_1$, $\langle e_2,\, s\rangle \Downarrow n_2$ for some n_1 and n_2 such that $n = n_1 + n_2$. We can now apply the inductive hypothesis $P(e_1)$ and $P(e_2)$ to conclude that: $\langle e_1,\, s\rangle \to^* \langle n_1,\, s\rangle$ and $\langle e_2,\, s\rangle \to^* \langle n_1,\, s\rangle$. From here, we can use the Lemma 1 below to determine that:

  $$\langle e_1 + e_2,\, s\rangle \to^* \langle n_1 + e_2,\, s\rangle \to^* \langle n_1 + n_2,\, s\rangle \to \langle n,\, s\rangle$$

  which proves this case.
- Case $e = e_1 * e_2$.
  Similar to the case above.

$\Rightarrow$ This implication follows from the Lemma 2 below.

Lemma 1. If $\langle e_1,\, s\rangle \to^k \langle n_1,\, s\rangle$ then $\langle e_1 + e_2,\, s\rangle \to^k \langle n_1 + e_2,\, s\rangle$ and $\langle e_1 * e_2,\, s\rangle \to^k \langle n_1 * e_2,\, s\rangle$. Similarly, if $\langle e_2,\, s\rangle \to^k \langle n_2,\, s\rangle$ then $\langle n_1 + e_2,\, s\rangle \to^k \langle n_1 + n_2,\, s\rangle$ and $\langle n_1 * e_2,\, s\rangle \to^k \langle n_1 * n_2,\, s\rangle$.

Proof. By (mathematical) induction on the number k of evaluation steps.

Lemma 2. For all e, e', s, n, if $\langle e,\, s\rangle \to \langle e',\, s\rangle$ and $\langle e',\, s\rangle \Downarrow n$, then $\langle e,\, s\rangle \Downarrow n$.