IMP

iohex

September 2021

1 IMP

IMP is a small lanuage of while programs, which called "imperative" lanuage. In the *programming paradigms*, *imperative lanuage* means program execution involves carrying out series of explicit commands to change state.

syntactic sets

Firstly, we give the syntactic sets associated with IMP:

- numbers **N:** the set of signed decimal numerals.
- truth value **T**
- location **Loc:** non-empty strings of letters or such strings follwed by digits.
- arithmetic expressions **Aexp**
- boolean expressions **Bexp**
- commands **Com**

We define the *formation rules* for **Aexp** by:

$$a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1.$$

The symbol "::=" should be read as "can be" (p.s. BNF isn't it?)
And for **Bexp**:

$$b ::= \textbf{true} \mid \textbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$$

we define the syntactic of commands:

$$c ::= \textbf{skip} \mid X \ ::= \ a \mid c_0; c_1 \mid \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1 | \textbf{while } b \textbf{ do } c$$

From a *set-theory* point of view, this notaion provides an *inductive definition* of the syntactic set of **IMP**

For the moment, this notation should be viewed as simply telling us how to construct elements of the syntactic sets.

The evaluation of arithmetic expression

We need some notation to express when two elements e_0, e_1. We use $e_0 \equiv e_1$ to mean e_0 is identical to e_1. The set of *states* Σ consists of functions σ : $\textbf{Loc} \to \textbf{N}$. Thuse $\sigma(X)$ is the value, or contents, of location $\textbf{X}$ in state σ.

Consider the evaluation of an arithmetic expression a in a state σ. We can represent the situation of expression a waiting to be evaluated in state σ by the pair $\langle a, \sigma \rangle$ We shall **define** an evaluation relation between such pairs and numbers:

$$\langle a, \sigma \rangle \to n$$

This is meaning: expression a in state σ evaluates to n. Call pairs $\langle a, \sigma \rangle$, where a is an arithmetic expression and σ is a state.

Evaluation of numbers:

$$\langle n, \sigma \rangle \to n$$

Evaluation of subtractions:

$$\langle x, \sigma \rangle \to \sigma(X)$$

Evaluation of sums:

$$\frac{\langle a_0, \sigma \rangle \to n_0 \langle a_1, \sigma \rangle \to n_1}{\langle a_0 * a_1, \sigma \rangle \to n}$$

Evaluation of productions:

$$\frac{\langle a_0, \sigma \rangle \to n_0 \langle a_1, \sigma \rangle \to n_1}{\langle a_0 * a_1, \sigma \rangle \to n}$$

n is the product of n_0 and n_1.

You can read it as: If $\langle a_0, \sigma \rangle \to n_0$ and $\langle a_1, \sigma \rangle \to n_1$ then $\langle a_0 * a_1, \sigma \rangle \to n$

derivation tree

Consider evaluating an arithmetic expression a in some state σ. This amounts to finding a derivation in which the left part of the conclusion matches $\langle a, \sigma \rangle$

The evaluation relation determines a natural equivalence relation on expressions. We define:

$$a_0 \sim a_1 \; iff \; (\forall n \in N \forall sigma \in \Sigma . \langle a_0, \sigma \rangle \to n \iff \langle a_1, \sigma \rangle \to n)$$

which makes two arithmetic expressions **equivalent** if they evaluate to the same value in all states.