Compilers

Author: Daria Shutina

23-02-23

There are several ways to define semantics of the language:

- write a document with standards
- Write an interpreter and a compiler
- Use formal semantics. You have a start state, a finish state and a transition function

Let us define an expression E which is a variable, an integer or a binary operation:

$E := X \mid N \mid Binop \;\times\; E\; E$, where $X = \{a, b, c, \dots\},\; \times = \{+, -, *, \backslash, \%, <, \dots\}$.

We can see any expression as a binary tree.

The semantics of the language is a total map:

$$\underbrace{[\,|\,.\,|\,]}_{\text{semantics}} : \underbrace{E}_{\text{expressions}} \to \underbrace{D}_{\text{semantics domain}}$$

$D = (X \to Z) \to Z$ -- we got a variable, turn it into an integer, then turn the integer into the result integer.

Interpreters

We have the input and the output. An interpreter takes a program and its input as arguments, and returns what the program would return.

A simple interpreter

$\sigma \in E$

- $n \in N;\ \sigma \xrightarrow{n} n$
- $x \in X;\ \sigma \xrightarrow{x} \sigma(x)$
- $\sigma \xrightarrow{l \times_1 r} x \times_2 y$

A smarter interpreter

$S = X \mid N \mid read \mid write \mid skip \mid Binop \mid \underbrace{S_1;\ S_2}_{\text{composition of expressions}}$

In this case, `;` is a concatenation operator

$[\,|\,.\,|\,]_s\ :\ S \to D$

The simplest idea for the output is a set of numbers.

- $c \xrightarrow{skip} c$
- $(\sigma, w) \xrightarrow{x:=e} (\sigma[x \leftarrow [\ |e|\ ]_\sigma, w)$
- $(\sigma, (i\ :\ is\ :\ o)) \xrightarrow{read(x)} (\sigma[x \leftarrow i], (is, o))$ $(i\ :\ is\ :\ o\ -\ i \text{ is head},\ is \text{ is tail})$

  We take the first number and assign it into x (that is what σ does).
- $(\sigma, (i, o)) \xrightarrow{write(e)} (\sigma, (i, o + +[|e|]_\sigma))$
- $\left.\begin{matrix} c \xrightarrow{s_1} c' \\ c' \xrightarrow{s_2} c'' \end{matrix}\right\} \Rightarrow\ c \xrightarrow{s_1; s_2} c''$

Example