Programming Languages

Bibhas Adhikari

IIT Kharagpur

October 16, 2020

Semantics: Example of Conditional transitions

The rule of conditional transition is expressed as follows:

$$\frac{\langle c_1, \sigma_1 \rangle \rightarrow \langle c_1', \sigma_1' \rangle \quad \langle c_2, \sigma_2 \rangle \rightarrow \langle c_2', \sigma_2' \rangle}{\langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle}$$

Meaning: If the command c_1 starting in state σ_1 can transform itself into command c_1' in state σ_1', and if c_2 starting in σ_2 can transform itself into the command c_2' in state σ_2', then the command c starting in state σ can transform itself into the command c' in state σ'

Semantics: Rules for semantics of arithmetic expressions

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

$$\frac{}{\langle (n+m), \sigma \rangle \to \langle p, \sigma \rangle} \text{ where } p = n + m \qquad \frac{}{\langle (n-m), \sigma \rangle \to \langle p, \sigma \rangle} \text{ where } p = n - m$$

$$\frac{\langle a_1, \sigma \rangle \to \langle a', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \to \langle (a' + a_2), \sigma \rangle} \qquad \frac{\langle a_2, \sigma \rangle \to \langle a'', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \to \langle (a_1 + a''), \sigma \rangle}$$

$$\frac{\langle a_1, \sigma \rangle \to \langle a', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \to \langle (a' - a_2), \sigma \rangle} \qquad \frac{\langle a_2, \sigma \rangle \to \langle a'', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \to \langle (a_1 - a''), \sigma \rangle}$$

Figure: Semantics of arithmetic expressions

Here, the pairs (a, σ), where a is an arithmetic expression and σ is a state, arithmetic-expression configurations. $\langle a, \sigma \rangle \to \langle b, \sigma' \rangle$ means the expression a in state σ evaluates to b.

Semantics: Rules for semantics of Boolean expressions

$$\langle (n == m), \sigma \rangle \to \langle \mathbf{tt}, \sigma \rangle \quad \text{if } n = m \qquad \langle (n == m), \sigma \rangle \to \langle \mathbf{ff}, \sigma \rangle \quad \text{if } n \neq m$$

$$\langle (bv_1 \wedge bv_2), \sigma \rangle \to \langle bv, \sigma \rangle \quad \text{where } bv \text{ is the } \textit{and} \text{ of } bv_1 \text{ and } bv_2$$

$$\langle \neg \mathbf{tt}, \sigma \rangle \to \langle \mathbf{ff}, \sigma \rangle \qquad \langle \neg \mathbf{ff}, \sigma \rangle \to \langle \mathbf{tt}, \sigma \rangle$$

$$\frac{\langle a_1, \sigma \rangle \to \langle a', \sigma \rangle}{\langle (a_1 == a_2), \sigma \rangle \to \langle (a' == a_2), \sigma \rangle} \qquad \frac{\langle a_2, \sigma \rangle \to \langle a'', \sigma \rangle}{\langle (a_1 == a_2), \sigma \rangle \to \langle (a_1 == a''), \sigma \rangle}$$

$$\frac{\langle b_1, \sigma \rangle \to \langle b', \sigma \rangle}{\langle (b_1 \wedge b_2), \sigma \rangle \to \langle (b' \wedge b_2), \sigma \rangle} \qquad \frac{\langle b_2, \sigma \rangle \to \langle b'', \sigma \rangle}{\langle (b_1 \wedge b_2), \sigma \rangle \to \langle (b_1 \wedge b''), \sigma \rangle}$$

$$\frac{\langle b, \sigma \rangle \to \langle b', \sigma \rangle}{\langle \neg b, \sigma \rangle \to \langle \neg b', \sigma \rangle}$$

Figure: Semantics of Boolean expressions

Here, *bv* denotes the a Boolean value (**tt** or **ff**)

Semantics: Rules for semantics of commands

$$\langle \textbf{skip}, \sigma \rangle \to \sigma \quad (c1)$$

$$\langle X := n, \sigma \rangle \to \sigma[X \leftarrow n] \quad (c2) \qquad \frac{\langle a, \sigma \rangle \to \langle a', \sigma \rangle}{\langle X := a, \sigma \rangle \to \langle X := a', \sigma \rangle} \quad (c3)$$

$$\frac{\langle c_1, \sigma \rangle \to \sigma'}{\langle c_1; c_2, \sigma \rangle \to \langle c_2, \sigma' \rangle} \quad (c4) \qquad \frac{\langle c_1, \sigma \rangle \to \langle c_1', \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \to \langle c_1'; c_2, \sigma' \rangle} \quad (c5)$$

$$\langle \textbf{if tt then } c_1 \textbf{ else } c_2, \sigma \rangle \to \langle c_1, \sigma \rangle \quad (c6)$$
$$\langle \textbf{if ff then } c_1 \textbf{ else } c_2, \sigma \rangle \to \langle c_2, \sigma \rangle \quad (c7)$$

$$\frac{\langle b, \sigma \rangle \to \langle b', \sigma \rangle}{\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \sigma \rangle \to \langle \textbf{if } b' \textbf{ then } c_1 \textbf{ else } c_2, \sigma \rangle} \quad (c8)$$

$$\langle \textbf{while } b \textbf{ do } c, \sigma \rangle \to \langle \textbf{if } b \textbf{ then } c; \textbf{while } b \textbf{ do } c \textbf{ else skip}, \sigma \rangle \quad (c9)$$

Figure: Semantics of commands

Semantics: computation

Computation

It is a sequence of transitions that cannot be extended further by another transition

Example

Consider the program c as
$X := 1$; **while** $\neg(X == 0)$ **do** $X := (X - 1)$,
where the later part after $';'$ is the command c'

Semantics: example of terminated computation

$$
\begin{aligned}
&\langle c, \sigma \rangle \\
&\rightarrow \langle c', \sigma[X \leftarrow 1] \rangle \\
&\rightarrow \langle \textbf{if } \neg(X == 0) \textbf{ then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 1] \rangle \\
&\rightarrow \langle \textbf{if } \neg(1 == 0) \textbf{ then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 1] \rangle \\
&\rightarrow \langle \textbf{if } \neg\textbf{ff then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 1] \rangle \\
&\rightarrow \langle \textbf{if tt then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 1] \rangle \\
&\rightarrow \langle X := (X - 1); c', \sigma[X \leftarrow 1] \rangle \\
&\rightarrow \langle X := (1 - 1); c', \sigma[X \leftarrow 1] \rangle \\
&\rightarrow \langle X := 0; c', \sigma[X \leftarrow 1] \rangle \\
&\rightarrow \langle c', \sigma[X \leftarrow 0] \rangle \\
&\rightarrow \langle \textbf{if } \neg(X == 0) \textbf{ then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 0] \rangle \\
&\rightarrow \langle \textbf{if } \neg(0 == 0) \textbf{ then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 0] \rangle \\
&\rightarrow \langle \textbf{if } \neg\textbf{tt then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 0] \rangle \\
&\rightarrow \langle \textbf{if ff then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 0] \rangle \\
&\rightarrow \langle \textbf{skip}, \sigma[X \leftarrow 0] \rangle \\
&\rightarrow \sigma[X \leftarrow 0]
\end{aligned}
$$

Semantics: example of divergent computation

Consider the program d as
$X := 1$; **while** $(X == 1)$ **do skip**
where the later part after $';'$ is the command d'

$$
\begin{aligned}
&\langle d, \tau \rangle \\
&\rightarrow \langle d', \tau[X \leftarrow 1] \rangle \\
&\rightarrow \langle \textbf{if } (X == 1) \textbf{ then } \textbf{ skip } ; d' \textbf{ else skip}, \tau[X \leftarrow 1] \rangle \\
&\rightarrow \langle \textbf{if } (1 == 1) \textbf{ then } \textbf{ skip } ; d' \textbf{ else skip}, \tau[X \leftarrow 1] \rangle \\
&\rightarrow \langle \textbf{if tt then } \textbf{ skip } ; d' \textbf{ else skip}, \tau[X \leftarrow 1] \rangle \\
&\rightarrow \langle \textbf{ skip } ; d', \tau[X \leftarrow 1] \rangle \\
&\rightarrow \langle d', \tau[X \leftarrow 1] \rangle \\
&\rightarrow \ldots
\end{aligned}
$$

Pragmatics

Recall

- syntax: described using a context-free grammar
- semantics: described with verifiable contextual constraints using the program text and it gives meaning to a program during execution

Pragmatics

what is the purpose of this command?

The pragmatics part of a programming language is dealt with software engineering

The Halting Problem

Let $\mathcal{L}$ be a programming language, and P be a program written in $\mathcal{L}$. Then we ask the following question.

Does there exist a program, H, such that, having been given an input a program P and its input data x, will terminate and print "yes" if $P(x)$ terminates, and terminate and print "no" if $P(x)$ loops infinitely?

We show that such a program H can not exist!

Reference

J. E. Hopcroft, R. Motwani, and J. Ullman. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, 2001 (Chapter 8, 9)

The Halting Problem

Method of contradiction:

1. Suppose we have a program H with the properties stated above
2. Then we can define a program K utilizing the program H such that: Given an input P, the program K terminates printing "yes" if $H(P, P)$ prints "no"; and it goes into an infinite loop if $H(P, P)$ prints "yes". Thus

$$K(P) = \begin{cases} \text{"yes" if } P(P) \text{ does not terminate} \\ \text{does not terminate if } P(P) \text{ terminates} \end{cases} \tag{1}$$

3. If we substitute K in place of P in equation (2) then

$$K(P) = \begin{cases} \text{"yes" if } K(K) \text{ does not terminate} \\ \text{does not terminate if } K(K) \text{ terminates} \end{cases} \tag{2}$$

But this is absurd!!!

Halting Problem

Undecidable problems

There exists no program such that

- the program accepts arbitrary arguments
- the program always terminates, and
- the program determines which arguments are solutions to the problem and which are not

Which functions are computable?

- A function is computable in a language $\mathcal{L}$ if there exists a program P written in $\mathcal{L}$ that computes it.
- Turing Machine (developed by the mathematician Alan Turing, 1930)
- Church's Thesis: every computable function is computed by a Turing Machine

Turing Machine

The question raised by David Hilbert

For which classes of problems is it possible to find an algorithm?

A related question

Given a logical system defined by axioms and rules, can all possible propositions in that system be proved? or at least in principle, can it be judged as true or false?

1930, Kurt Godel : NOT Possible! There are propositions in any logical system that can not be proved or disproved using the axioms and rules of the logical system - it sets the limits for possibilities of a classical computer