Programming Languages

Bibhas Adhikari

IIT Kharagpur

October 9, 2020

Semantics

- The complexity of semantics arises from the need of exactness as well as for flexibility
  - Exactness : precise and unambiguous description of syntactically correct construct
  - Flexible : it must not anticipate choices that are to be made when the language is implemented
- It is enough to give the semantics of a language using a specific compiler on a specific architecture
- Semantic phenomena make formal description complex and not easily usable by anyone who does not have the appropriate skills

Semantics

- Formal methods for semantics divide into two main families:
  - Denotational semantics : Application to programming languages of techniques developed for the semantics of mathematical languages (logic based). Thus in this approach, a the meaning of a program is described by a function whose domain and codomain are suitable mathematical structures
  - Operational semantics : It specifies the behaviour of the abstract machine. It formally defines the interpreter, making reference to an abstract formalism at a much lower level. Various operational techniques can be distinguished by their choice of formalism; some semantics use formal automata, some use systems of logical mathematical rules, and the others prefer transition systems to specify the state transformations induced by a program.

Semantics: grammar of a simple imperative language

$$
\begin{aligned}
Num &::= 1 \mid 2 \mid 3 \mid \ldots \\
Var &::= X_1 \mid X_2 \mid X_3 \mid \ldots
\end{aligned}
$$

$$
\begin{aligned}
AExp &::= Num \mid Var \mid (AExp + AExp) \mid (AExp - AExp) \\
BExp &::= \mathbf{tt} \mid \mathbf{ff} \mid (AExp == AExp) \mid \neg BExp \mid (BExp \wedge BExp)
\end{aligned}
$$

$$
\begin{aligned}
Com ::= {} & \mathbf{skip} \mid Var := AExp \mid Com; Com \mid \\
& \mathbf{if}\ BExp\ \mathbf{then}\ Com\ \mathbf{else}\ Com \mid \ \mathbf{while}\ BExp\ \mathbf{do}\ Com
\end{aligned}
$$

Figure: Example of syntax of a simple imperative language

Observe:

- IMP, a small language of while programs. IMP is called an "imperative" language because program execution involves carrying out a series of explicit commands to change state
- infinite productions for the *Num* and *Var* non-terminal symbols
- *Num*: numeric constants; Var: variable; *AExp*: arithmetic expression; *BExp*: Boolean expression, **tt**- true, **ff** - false; *Com*: commands

Semantics: example of techniques based on transition systems to define semantics in a rudimentary programming language (Structured Operational Semantics)

State

- The semantics of a command corresponding to the language defined by the grammar above uses a memory model which store values of *Var*
- A state is a finite sequence of pairs of the form (X, n) which we can read as "in the current state, the variable X has the value n"
- The reference state of a command (which is a derivation tree) is a sequence of pairs which includes all *Var*s which are included in the command

References

- G.Winskel. The Formal Semantics of Programming Languages. MIT Press, Cambridge, 1993 (Read: Chapter 2)
- G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981

Semantics

Operation on states

- Given a state σ, a $VarX$ and a value v, write $\sigma[X \leftarrow v]$ to denote a new state that is the same as σ but differs from it by associating X with the value v
- Given a state σ, and a variable, X, write $\sigma(X)$ for the value that σ associates with X; this value is undefined if X does not appear in the domain of σ (σ is therefore a partial function)

Example

Let $\sigma = [(X, 3), (Y, 5)]$, we have $\sigma[X \leftarrow 7] = [(X, 7), (Y, 5)]$. We also have $\sigma(Y) = 5$ and $\sigma[X \leftarrow 7](X) = 7$; $\sigma(W)$ is undefined

Semantics

Transitions

Let c be a command, σ the starting state and τ the terminal state. The a simple transition is

$$\langle c, \sigma \rangle \rightarrow \tau$$

Meaning: if we start the execution of c in the state σ then the execution terminates (in a single step) with state τ

Example

The transition for **skip** command is given by

$$\langle \textbf{skip}, \sigma \rangle \rightarrow \sigma$$

Semantics: Example of Transition

In general, there are many little steps to reach to a final state from an initial state. Each such little step is of the form

$$\langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle.$$

Example: conditional command

A transition which defines a conditional command can be:

$$\langle \textbf{if tt then } c_1 \textbf{ else } c_2, \sigma \rangle \rightarrow \langle c_1, \sigma \rangle.$$

Meaning: if the boolean condition is true, the command in the then branch must be executed