Formal Methods
4. Axiomatic Semantics

Nachum Dershowitz

March 2000

Since nondeterministic programs can return more than one result, it is best to view programs as binary input/output relations. We will make use of standard mathematical notation for sets and relations: union $\cup$, intersection $\cap$, composition (juxtaposition, or $\circ$, or ";"), reflexive-transitive closure R^*, inverse R^{-1}, etc.

We consider state-changing programs with assignment statements of the form $x := e$. For tests, we use a restriction of the identity relation $p? = \{\langle x, x\rangle | p(x)\}$.

The following are definitions of more familiar programming constructs:

$$\begin{array}{lcl}
\textbf{if } \alpha \textbf{ then } \beta \textbf{ else } \gamma & = & \alpha?\beta \cup (\neg\alpha)?\gamma \\
\textbf{while } \alpha \textbf{ do } \beta & = & (\alpha?\beta)^*(\neg\alpha)? \\
\textbf{skip} & = & I \text{ (the identity relation } T?) \\
\textbf{fail} & = & \emptyset \text{ (the empty relation } F?) \\
\textbf{loop} & = & I^* \\
a[j] := e & = & a := \lambda i. \textbf{if } i = j \textbf{ then } e \textbf{ else } a[i]
\end{array}$$

We will use the notation:

$$A \xrightarrow[R]{} B$$

to mean

$$\forall \bar{x}, \bar{z}\{A[\bar{x}] \wedge \bar{x}R\bar{z} \rightarrow B[\bar{z}]\}$$

That is, if A is true for state $\bar{x}$, then after executing program R, B will be true in the new state $\bar{z}$. Other notations for the same concept used in the literature include:

$$\begin{array}{cl} A\{R\}B & \text{(Hoare)} \\ \{A\}R\{B\} & \text{(Manna)} \\ A \rightarrow \mathit{wlp}(R,B) & \text{(Dijkstra)} \\ A \rightarrow [R]B & \text{(Harel)} \end{array}$$

Properties of programs that can be expressed in this manner include:

Output Correctness

$$A \underset{R}{\longrightarrow} B$$

Termination

$$\neg(A \underset{R}{\longrightarrow} F)$$

The semantics of basic statements can be defined by the following axioms:

$$\begin{array}{rcl} A & \underset{p?}{\longrightarrow} & A \wedge p \\ A[e] & \underset{v:=e}{\longrightarrow} & A[v] \end{array}$$

where v is a state variable appearing in formula A.

In addition we have the following equivalences:

$$\begin{array}{rcl} A \underset{I}{\longrightarrow} B & \Leftrightarrow & A \rightarrow B \\ (A \vee B) \underset{R \cup S}{\longrightarrow} (C \vee D) & \Leftrightarrow & A \underset{R}{\longrightarrow} C \wedge B \underset{S}{\longrightarrow} D \\ A \underset{RS}{\longrightarrow} C & \Leftrightarrow & A \underset{R}{\longrightarrow} [T \underset{S}{\longrightarrow} C] \\ A \underset{R^*}{\longrightarrow} A & \Leftrightarrow & A \underset{R}{\longrightarrow} A \\ A \underset{R^{-1}}{\longrightarrow} A & \Leftrightarrow & \neg(\neg B \underset{R}{\longrightarrow} \neg A) \end{array}$$

The above provides a compositional semantics for state-modifying iterative programs.

For concurrent programs, it is more convenient to look at the whole program as a state-transition relation. The one-step relation τ is described by a set of formulas that speak of state-variable values and program-statement labels. Computations are just sequences of state-transitions and we are interested in properties that can be expressed by formulas like

$$A \underset{\tau^*}{\longrightarrow} B$$

Since it is always the same relation that interests us, we can use instead formulas

$$\Box B$$

meaning

$$T \underset{\tau^*}{\longrightarrow} B$$

We can also define

$$\Diamond A \Leftrightarrow \neg(\Box \neg A)$$

meaning that there is a computation leading to a state in which A holds.