Formal Methods
4. Axiomatic Semantics

Nachum Dershowitz

28/3/2000

1 Introduction

A computer program contains a finite number of state variables. The combined states of all variables defines the current state of the program. In each computation step one state variable may change its value. Since nondeterministic programs can move from a certain state to a set of possible states by one computational step, it is best to view programs as binary input/output relations. When we deal with a deterministic program, the binary relation becomes functional. 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.

2 Syntax

We now show how to express statements in computer programs using these relations.
We consider the if statement : **if** α **then** β **else** γ . We would like to build a relation R_{if} to represent the if statement given the relation $R\alpha$, R_β and R_γ for the statements α, β and γ, respectively.
To represent a condition we define : $\alpha? = \{(\bar{x},\bar{x})|\alpha(\bar{x}\}$, where α is a predicate.
The relation $\alpha?$ transforms each state $\bar{x}$ where $\alpha(\bar{x}) = \text{true}$ to itself, the state is unchanged. We can now express the if statement as :
if α **then** β **else** γ $= \alpha?\beta \cup (\neg\alpha)?\gamma$

Similarly we can express the following statements :

while α **do** β	$=$	$(\alpha?\beta)^*(\neg\alpha)?$
skip	$=$	I (the identity relation $T?$)
fail	$=$	$\emptyset$ (the empty relation $F?$)
loop	$=$	I^*
$a[j] := e$	$=$	$a := \lambda i.$**if** $i = j$ **then** e **else** $a[i]$

3 Dijkstra's nondeterministic language

Dijkstra developed a language for nondeterministic computation.
The if statement for example is defined :

```
  if
     $C_1 \rightarrow S_1$
     $C_2 \rightarrow S_2$
     .
     .
     .
     $C_n \rightarrow S_n$
  fi
```

Line $C_i \rightarrow S_i$ means that if C_i holds we perform Si. The program chooses nondeterministicly one of the lines for which the condition is true.
Consider for example a program to compute the maximum of two numbers :

```
  if
     $x \geq y \rightarrow m := x$
     $x \leq y \rightarrow m := y$
  fi
```

If $x = y$ (both conditions are true) it does not matter which line is chosen to be done.

The do statement in Dijkstra's language is defined like this :

```
do
    $C_1 \rightarrow S_1$
    $C_2 \rightarrow S_2$
    .
    .
    .
    $C_n \rightarrow S_n$
od
```

The do statement is like a repetitive if statement. In each step we do one of the statements associated with a true predicate. Once all predicates return false, the loop terminates.

4 Program properties

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 holds 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:

$A\{R\}B$	(Hoare)
$\{A\}R\{B\}$	(Manna)
$A \rightarrow wlp(R, B)$	(Dijkstra)
$A \rightarrow [R]B$	(Harel)

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

- **Output Correctness**

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

- **Termination**

$$\neg(A \xrightarrow[R]{} F)$$

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

- **Test Axiom**

$$A \xrightarrow[p?]{} A \wedge p$$

- **Assign Axiom**
$$A[e] \xrightarrow[v:=e]{} A[v]$$
where v is a state variable appearing in formula A.
In addition we have the following equivalences:

- **Identity**
$$A \xrightarrow[I]{} B \Leftrightarrow A \rightarrow B$$

- **Union**
$$A \xrightarrow[R]{} B \wedge A \xrightarrow[S]{} B \Leftrightarrow A \xrightarrow[R \cup S]{} B$$

- **Composition**
$$A \xrightarrow[RS]{} B \Leftrightarrow A \xrightarrow[R]{} (T \xrightarrow[S]{} B)$$

- **Star**
$$A \xrightarrow[R]{} A \Leftrightarrow A \xrightarrow[R^*]{} A$$

-
$$A \xrightarrow[R]{} C \wedge B \xrightarrow[S]{} D \rightarrow (A \vee B) \xrightarrow[R \cup S]{} (C \vee D)$$

-
$$A \xrightarrow[R^{-1}]{} B \Leftrightarrow \neg B \xrightarrow[R]{} \neg A$$

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. For example, if we have two concurrent programs S and R represented by relations τ_S and τ_R respectively, the relation for the entire program is $\tau = \tau_R \cup \tau_S$. Computations are just sequences of state-transitions and we are interested in properties that can be expressed by formulas like

$$A \xrightarrow[\tau^*]{} B$$

We define $\Box B = T \xrightarrow[\tau^*]{} B$ which means that B is invariant throughout the program, with no regard to the initial state.
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.
In deterministic programs $\Box\Diamond A$ means that from every path we take in the program we can reach a state where A is true. We also have $\Diamond\Box A$ which means that starting from some place, A remains true always.