Formal methods

Array assignment

FOR-command

Overview

- All the axioms and rules given so far were quite straightforward
  - may have given a false sense of simplicity
- Hard to give rules for anything other than *very* simple constructs
  - an incentive for using simple languages
- We already saw with the assignment axiom that our intuition over how to formulate a rule might be wrong
  - the assignment axiom can seem 'backwards'
- We now look at the remaing commands in our little language
  - array assignments
  - `FOR`-commands

Array assignments

- Syntax: $V(E_1)$:=E_2
- Semantics: the state is changed by assigning the value of the term E_2 to the E_1-th component of the array variable V
- Example: `A(X+1) := A(X)+2`
  - if the the value of `X` is x
  - and the value of the x-th component of `A` is n
  - then the value stored in the $(x+1)$-th component of `A` becomes $n+2$

Naive Assignment Axiom Fails

- The axiom

  $$\vdash \{P[E_2/A(E_1)]\}\ A(E_1) := E_2\ \{P\}$$

  doesn't work

- Take $P \equiv$ 'A(Y)=0', $E_1 \equiv$ 'X', $E_2 \equiv$ '1'
  - since A(X) does not occur in P
  - it follows that P[1/A(X)] = P
  - and hence the generalised axiom yields

    $$\vdash \{\mathtt{A(Y)=0}\}\ \mathtt{A(X):=1}\ \{\mathtt{A(Y)=0}\}$$

  - false if X=Y

- Must take into account possibility that changes to A(X) may change A(Y), A(Z), ...
  - since X might equal Y, Z, ...
  - i.e. aliasing

Idea of the Solution

- The naive array assignment axiom

$$\vdash \{P[E_2/A(E_1)]\}\ A(E_1) := E_2\ \{P\}$$

does not work: changes to $A(X)$ may also change $A(Y)$, $A(Z)$, ...

- The solution to this, due to Hoare, is to treat an array assignment

$$A(E_1){:=}E_2$$

as an ordinary assignment

$$A := A\{E_1 \leftarrow E_2\}$$

where the term $A\{E_1 \leftarrow E_2\}$ denotes an array identical to A, except that the E_1-th component is changed to have the value E_2

Array Assignment Axiom

- Array assignment is a special case of ordinary assignment

$$A := A\{E_1 \leftarrow E_2\}$$

- Array assignment axiom just ordinary assignment axiom

$$\vdash \ \{P[A\{E_1 \leftarrow E_2\}/A]\} \ A := A\{E_1 \leftarrow E_2\} \ \{P\}$$

- Thus:

The array assignment axiom

$$\vdash \ \{P[A\{E_1 \leftarrow E_2\}/A]\} \ A(E_1) := E_2 \ \{P\}$$

Where A is an array variable, E_1 is an integer valued expression, P is any statement and the notation $A\{E_1 \leftarrow E_2\}$ denotes the array identical to A, except that $A(E_1) = E_2$.

Array Axioms

- **In order to reason about arrays, the following axioms, which define the meaning of the notation $\mathbf{A}\{E_1 \leftarrow E_2\}$, are needed**

The array axioms

$$\vdash \mathbf{A}\{E_1 \leftarrow E_2\}\ (E_1) = E_2$$

$$\vdash E_1 \neq E_3 \Rightarrow \mathbf{A}\{E_1 \leftarrow E_2\}\ (E_3) = \mathbf{A}(E_3)$$

- **It is more convenient to use a derived rule in the proofs**

Derived assignment rule

$$\frac{\vdash P \Rightarrow Q[\mathbf{A}\{E_1 \leftarrow E_2\} / \mathbf{A}]}{\vdash \{P\}\ \ \mathbf{A}(E_1) := E_2\ \ \{Q\}}$$

FOR-command

- Syntax: `FOR` V`:=`E_1 `UNTIL` E_2 `DO` C
  - restriction: V must not occur in E_1 or E_2,
    or be the left hand side of an assignment in C
    (explained later)
- Semantics:
  - if the values of terms E_1 and E_2 are positive numbers e_1 and e_2
  - and if $e_1 \leq e_2$
  - then C is executed $(e_2 - e_1)+1$ times with the variable V taking on the sequence of values e_1, e_1+1, . . . , e_2 in succession
  - for any other values, the `FOR`-command has no effect
- Example: `FOR N:=1 UNTIL M DO X:=X+N`
  - if the value of the variable `M` is m and $m \geq 1$, then the command `X:=X+N` is repeatedly executed with `N` taking the sequence of values `1`, . . . , m
  - if $m < 1$ then the `FOR`-command does nothing

Semantics of FOR-command

- The semantics of

$$\texttt{FOR}\ V\texttt{:=}E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C$$

  is as follows

(i) The expressions E_1 and E_2 are evaluated once to get values e_1 and e_2, respectively.

(ii) If either e_1 or e_2 is not a number, or if $e_1 > e_2$, then nothing is done.

iii) If $e_1 \leq e_2$ the FOR-command is equivalent to:

$$\texttt{BEGIN VAR}\ V\texttt{;}\ V\texttt{:=}e_1\texttt{;}\ C\texttt{;}\ V\texttt{:=}e_1\texttt{+}1\texttt{;}\ C\ \texttt{;}\ \ldots\ \texttt{;}\ V\texttt{:=}e_2\texttt{;}\ C\ \texttt{END}$$

i.e. C is executed $(e_2{-}e_1){+}1$ times with V taking on the sequence of values e_1, $e_1{+}1$, ... , e_2

- If C doesn't modify V then FOR-command equivalent to:

$$\texttt{BEGIN VAR}\ V\texttt{;}\ V\texttt{:=}e_1\texttt{;}\ \ldots\ \underbrace{C\ \texttt{;}\ V\texttt{:=}V\texttt{+}1}_{\text{repeated}}\texttt{;}\ \ldots\ V\texttt{:=}e_2\texttt{;}\ C\ \texttt{END}$$

Reduction to WHILE-command

- **FOR-command**

  FOR V:=E_1 UNTIL E_2 DO C

- **is equivalent to program**

BEGIN VAR V;
  $V := E_1$;
  WHILE $V \geq E_1 \wedge V \leq E_2$ DO BEGIN
    C;
    $V := V+1$
  END
END

Annotating FOR-command

- **Annotating FOR-command**

  {P} FOR V:= E_1 UNTIL E_2 DO {R} C {Q}

- **we get an annotated WHILE program**

{P}

BEGIN VAR V;

  V:= E_1;

  WHILE $V \geq E_1 \wedge V \leq E_2$ DO {R} BEGIN

    C;

    V := V+1

  END

END

{Q}

R includes condition

$V \leq E_2$+1

FOR-rule

Derived FOR –rule

$$\vdash P \Rightarrow R[E_1/V] \qquad \vdash R[E_2{+}1/V] \Rightarrow Q \qquad \vdash P \wedge (E_2 < E_1) \Rightarrow Q$$

$$\frac{\vdash \{R \wedge (E_1 \leq V) \wedge (V \leq E_2)\}\ C\ \{R[V{+}1/V]\}}{\vdash \{P\}\ \textbf{FOR}\ V := E_1\ \textbf{UNTIL}\ E_2\ \textbf{DO}\ \{R\}C\ \{Q\}}$$