Outros comandos: Ciclos `for`

Suponhamos que a linguagem imperativa continha um comando `for`

$$\texttt{for}\ x := E_1\ \texttt{until}\ E_2\ \texttt{do}\ C$$

cujo significado é:

- As expressões E_1 e E_2 são avaliadas no início, e sejam e_1 e e_2 os seus valores;
- Se $e_1 > e_2$ não se faz nada;
- Se $e_1 <= e_2$ o comando `for` é equivalente a

  $$x := e_1; C; x = e_1 + 1; C \ldots; x := e_2; C$$

  O ciclo é executado $(e_2 - e_1) + 1$ vezes

Ciclos `for`

Bastaria então uma regra para o `for` da forma:

$$\frac{\{\psi\}\, C\, \{\psi[x+1/x]\}}{\{\psi[E_1/x]\}\, \texttt{for}\, x := E_1\, \texttt{until}\, E_2\, \texttt{do}\, C\, \{\psi[E_2+1/x]\}}$$

Mas isto não chega porque:

- O comando C pode alterar o valor de x;
- O valor de E_1 pode ser menor que o valor de E_2.

Ciclos `for`

Lógica de Hoare

[for_p**-axioma**] Para o caso de $E_1 > E_2$

$$\{\phi\ \wedge\ E_2 < E_1\}\, \texttt{for}\, x := E_1\, \texttt{until}\, E_2\, \texttt{do} C\, \{\phi\}$$

[for_p]

$$\frac{\{\phi\ \wedge\ E_1 \leq x\ \wedge\ x \leq E_2\}\, C\, \{\phi[x+1/x]\}}{\{\phi[E_1/x]\ \wedge\ E_1 \leq E_2\}\, \texttt{for}\, x := E_1\, \texttt{until}\, E_2\, \texttt{do}\, C\, \{\phi[E_2+1/x]\}}$$

onde nem x, nem nenhuma variável que ocorra em E_1 ou E_2 é modificada no comando C.

Exemplo

$$\vdash_p \{x = 0 \ \wedge \ 1 \leq m\}\texttt{for } n := 1 \texttt{ until } m \texttt{ do } x := x + n\{x = m \times (m+1)\, div\, 2\}$$

Considera ϕ igual a $x = (n-1) \times n\, div\, 2$.

Arrays (*aliases*)

Se tivermos uma variável indexada $a[]$ a regra da atribuição não pode ser directamente aplicada a um elemento:

$$\{\phi[E_2/a[E_1]]\}a[E_1] := E_2\{\phi\}$$

porque as modificações em $a[E_1]$ podem alterar outras referências a a que ocorram em ϕ ou em E_2.

Por exemplo, o triplo $\{a[i] > 100\}a[i] := 10\{a[j] > 100\}$ seria derivado pelo axioma acima, mas não é válido: por exmplo se for avaliado num estado em que $i = j$.

A solução de T. Hoare foi considerar os *Arrays* como monolíticos, e uma atribuição

$$a := a[E_1 \leftarrow E_2]$$

que significa que a passou a ser um *array* igual ao anterior mas em que a posição E_1 passou a valer E_2.

Assim no caso anterior o valor de $a[i]$ e de $a[j]$ mudam os dois...porque muda o próprio array...

Arrays

Lógica de Hoare

$[array_p\ ]$

$$\{\psi[a[E_1 \leftarrow E_2]/a]\}\, a[E_1] := E_2\, \{\psi\}$$

onde E_1 é um inteiro.

E onde

$$\begin{array}{lcl} a[E_1 \leftarrow E_2][E_1] & = & E_2 \\ a[E_1 \leftarrow E_2][E_3] & = & a[E_3] \text{ se } E_3 \neq E_1. \end{array}$$

Exemplo

$$\begin{array}{l}\vdash_p \{a[x] = x \ \wedge \ a[y] = y\} \\ \quad r := a[x]; \\ \quad a[x] := a[y]; \\ \quad a[y] := r \\ \quad \{a[x] = y \ \wedge \ a[y] = x\}\end{array}$$

Nota: Actualmente nas implementações não se usa esta técnica porque é computacionalmente muito ineficiente!...

Cálculo para a correcção total

Na linguagem imperativa apresentada, o único comando que pode levar à não terminação é o comando `while`.

O *cálculo* $\vdash_{tot}$ irá ser igual ao $\vdash_p$ excepto na regra $\texttt{while}_{tot}$.

Para demonstrar que um programa termina temos que lhe associar uma expressão estritamente decrescente, denominada variante.

No caso do `while`, podemos associar uma expressão inteira não negativa e mostrar que em cada iteração o valor dessa expressão diminui, mantendo-se não negativa: temos a certeza que `while` termina pois essa expressão só pode tomar um número finito de valores até chegar a zero!!!

No caso do factorial:

$y := 1; z := 0; \texttt{while}\ z\,! = x\ do\ (z := z + 1; y := y \times z)$

podemos tomar o variante $x - z$.

Cálculo para a correcção total

Lógica de Hoare (correcção total)

As regras ass_{tot}, $comp_{tot}$, if_{tot} e $cons_{tot}$ coincidem com as do *cálculo* para a correcção parcial.

$[while_{tot}\ ]$

$$\frac{\{\eta \ \wedge \ B \ \wedge \ 0 \leq E \ \wedge \ E = e_0\}\, C\, \{\eta \ \wedge \ 0 \leq E \ \wedge \ E < e_0\}}{\{\eta \ \wedge \ 0 \leq E\}\, \texttt{while}\, B\, \texttt{do}\, C\, \{\eta \ \wedge \ \neg B\}}$$

onde e_0 é uma variável lógica cujo valor é o da expressão E antes da execução do comando C.

Pré condição mais fraca - $while_{tot}$

$\{\phi\}$
$\{\eta \wedge 0 \leq E\}$
`while` B `do`
$\{\eta \wedge B \wedge 0 \leq E \wedge E = e_0\}$
C
$\{\eta \wedge 0 \leq E \wedge E < e_0\}$
$\{\eta \wedge \neg B\}$ $while_{tot}$
$\{\psi\}$ $cons_{tot}$

Exemplo

$\vdash_{tot} \{x \geq 0\} y := 1; z := 0; \texttt{while}\ z\,! = x\ do\ (z := z + 1; y := y \times z) \{y = x!\}$
$\{x \geq 0\}$
$\{1 = 0! \wedge 0 \leq x - 0\}$
`y` $:=$ `1`
$\{y = 0! \wedge 0 \leq x - 0\}$
`z` $:=$ `0`
$\{y = z! \wedge 0 \leq x - z\}$ ass_{tot}
`while` $z\,! = x\ do$
$\{$
$\{y = z! \wedge z! = x \wedge 0 \leq x - z \wedge x - z = e_0\}$ $cons_{tot}$
$\{y \times (z + 1) = (z + 1)! \wedge 0 \leq x - (z + 1) \wedge x - (z + 1) < e_0\}$ ass_{tot}
`z` $:=$ `z` $+$ `1`
$\{y \times z = z! \wedge 0 \leq x - z \wedge x - z < e_0\}$ ass_{tot}
`y` $:=$ `y` $\times$ `z`
$\{y = z! \wedge 0 \leq x - z \wedge x - z < e_0\}$
$\}$
$\{y = z! \wedge x = z\}$
$\{y = x!\}$

Como determinar um variante ?

Os variantes são mais difíceis de encontrar que os invariantes...porque não é possível saber genericamente se um programa termina

$$
\begin{array}{l}
\vdash_{tot} \{x > 0\} \\
\quad \mathtt{c} = \mathtt{x} \\
\quad \mathtt{while}(\mathtt{c}! = 1)\mathtt{do} \\
\quad\ \mathtt{if}(\mathtt{c}\%2 == 0)\mathtt{c} = \mathtt{c}/2 \\
\quad\ \mathtt{else}\ \mathtt{c} = 3 * \mathtt{c} + 1 \\
\quad \{\top\}
\end{array}
$$

Será que este triplo é válido? Neste caso este triplo só estabelecia a terminação do programa...

Mas não se sabe se termina ou não!

Exemplos

Exercício 16.1. *Mostra que*

$$
\begin{array}{r}
\vdash_{tot} \{y > 0\} \\
\mathtt{while}\, y \leq r\, \mathtt{do} \\
r := r - y; \\
q := q + 1 \\
\{\top\}
\end{array}
$$

$\diamond$