Cálculo de Correção parcial $\mathcal{H}$

$[skip_p\ ]$

$$\{\varphi\}\,\texttt{skip}\,\{\varphi\}$$

$[ass_p\ ]$

$$\{\varphi[E/x]\}\,x \leftarrow E\,\{\varphi\}$$

$[comp_p\ ]$

$$\frac{\{\varphi\}\,C_1\,\{\eta\} \qquad \{\eta\}\,C_2\,\{\psi\}}{\{\varphi\}\,C_1; C_2\,\{\psi\}}$$

$[if_p\ ]$

$$\frac{\{\varphi\ \wedge\ B\}\,C_1\,\{\psi\} \qquad \{\varphi\ \wedge\ \neg B\}\,C_2\,\{\psi\}}{\{\varphi\}\,\texttt{if}\,B\,\texttt{then}\,C_1\,\texttt{else}\,C_2\,\{\psi\}}$$

$[if'_p\ ]$

$$\frac{\{\varphi_1\}\,C_1\,\{\psi\} \qquad \{\varphi_2\}\,C_2\,\{\psi\}}{\{(B \to \varphi_1)\ \wedge\ (\neg B \to \varphi_2)\}\,\texttt{if}\,B\,\texttt{then}\,C_1\,\texttt{else}\,C_2\,\{\psi\}}$$

$[while_p\ ]$

$$\frac{\{\psi\ \wedge\ B\}\,C\,\{\psi\}}{\{\psi\}\,\texttt{while}\,B\,\texttt{do}\,C\,\{\psi\ \wedge\ \neg B\}}$$

$[cons_p\ ]$

$$\frac{\vdash \varphi' \to \varphi \qquad \{\varphi\}\,C\,\{\psi\} \qquad \vdash \psi \to \psi'}{\{\varphi'\}\,C\,\{\psi'\}}$$

Integridade e Completude

Recordemos que $\models_{par} \{\varphi\}C\{\psi\}$ significa que para todos os estados que satisfazem φ, o estado que resulta de executar C satisfaz ψ, desde que C termine.

Para o sistema dedutivo de Hoare, vamos considerar duas propriedades usuais em sistemas lógicos:

- **Integridade:** Cada regra deve preservar validade. O que implica (por indução nas derivações) que os teoremas obtidos correspondem a asserções válidas de correcção parcial.

$$\vdash_p \{\varphi\}C\{\psi\} \qquad \Rightarrow \qquad \models_p \{\varphi\}C\{\psi\}.$$

- **Completude:** Gostaríamos que o sistema fosse suficientemente forte para inferir todas as asserções de correção parcial válidas.

$$\models_p \{\varphi\}C\{\psi\} \qquad \Rightarrow \qquad \vdash_p \{\varphi\}C\{\psi\}.$$

Vamos começar por formalizar a noção de execução/avaliação.

Estado de execução

Para a avaliação duma expressão é necessário saber o valor das variáveis. Um **estado** s é uma função que associa a cada variável um valor.

Representamos o conjunto de estados por

$$\mathbf{State} = \mathbf{Var} \to \mathbb{Z}$$

e $s \in \mathbf{State}$ tal que $s : \mathbf{Var} \to \mathbb{Z}$.

Seja $s\ x$ ou $s(x)$ o valor da variável x no estado s. Se $v \in \mathbb{Z}$,

$$s[v/x](y) = \begin{cases} s(y) & \text{se } y \neq x \\ v & \text{se } y = x \end{cases}$$

Semântica das expressões

Aexp - Expressões aritméticas

$\mathcal{A} : \mathbf{Aexp} \to (\mathbf{State} \to Z)$

$$\begin{aligned} \mathcal{A}[\![n]\!]s &= n \\ \mathcal{A}[\![x]\!]s &= s(x) \\ \mathcal{A}[\![E_1 + E_2]\!]s &= \mathcal{A}[\![E_1]\!]s + \mathcal{A}[\![E_2]\!]s \\ \mathcal{A}[\![E_1 - E_2]\!]s &= \mathcal{A}[\![E_1]\!]s - \mathcal{A}[\![E_2]\!]s \\ \mathcal{A}[\![E_1 \times E_2]\!]s &= \mathcal{A}[\![E_1]\!]s.\mathcal{A}[\![E_2]\!]s \end{aligned}$$

Bexp - Expressões booleanas

$\mathbf{T} = \{\mathsf{true}, \mathsf{false}\}$

$\mathcal{B} : \mathbf{Bexp} \to (\mathbf{State} \to \mathbf{T})$

$$
\begin{array}{rcl}
\mathcal{B}[\![\mathtt{true}]\!]s & = & \mathsf{true} \\
\mathcal{B}[\![\mathtt{false}]\!]s & = & \mathsf{false} \\
\mathcal{B}[\![E_1 = E_2]\!]s & = & \begin{cases} \mathsf{true} & \text{se } \mathcal{A}[\![E_1]\!]s = \mathcal{A}[\![E_2]\!]s \\ \mathsf{false} & \text{se } \mathcal{A}[\![E_1]\!]s \neq \mathcal{A}[\![E_2]\!]s \end{cases} \\
\mathcal{B}[\![E_1 \leq E_2]\!]s & = & \begin{cases} \mathsf{true} & \text{se} \mathcal{A}[\![E_1]\!]s \leq \mathcal{A}[\![E_2]\!]s \\ \mathsf{false} & \text{se } \mathcal{A}[\![E_1]\!]s > \mathcal{A}[\![E_2]\!]s \end{cases} \\
\mathcal{B}[\![\neg b]\!]s & = & \begin{cases} \mathsf{true} & \text{se } \mathcal{B}[\![b]\!]s = \mathsf{false} \\ \mathsf{false} & \text{se} \mathcal{B}[\![b]\!]s = \mathsf{true} \end{cases} \\
\mathcal{B}[\![b_1 \ \wedge \ b_2]\!]s & = & \begin{cases} \mathsf{true} & \text{se } \mathcal{B}[\![b_1]\!]s = \mathsf{true} \text{ e } \mathcal{B}[\![b_2]\!]s = \mathsf{true} \\ \mathsf{false} & \text{se} \mathcal{B}[\![b_1]\!]s = \mathsf{false} \text{ ou } \mathcal{B}[\![b_2]\!]s = \mathsf{false} \end{cases}
\end{array}
$$

Semântica operacional natural (*big-step*)

Descreve a execução completa de cada comando.

Configurações: $\langle C, s\rangle$ ou s, onde C é um comando e s um estado $\Gamma = (\mathbf{Com} \times \mathbf{State}) \cup \mathbf{State}$

Configurações Finais: $s \in State$

Transições: $\langle C, s\rangle \ \longrightarrow \ s'$

Regras:

$$\frac{\langle C_1, s_1\rangle \ \longrightarrow \ s'_1 \ ... \ \langle C_n, s_n\rangle \ \longrightarrow \ s'_n}{\langle C, s\rangle \ \longrightarrow \ s'}$$

Hipóteses: $\langle C_i, s_i\rangle \ \longrightarrow \ s'_i$

Conclusão: $\langle C, s\rangle \ \longrightarrow \ s'$

Se $n = 0$ diz-se um **Axioma**.

Semântica operacional para comandos do While

$$
\begin{array}{ll}
\mathtt{att}_{sn} & \langle x \leftarrow E, s\rangle \ \longrightarrow \ s[\mathcal{A}[\![E]\!]s/x] \\
\mathtt{comp}_{sn} & \dfrac{\langle C_1, s\rangle \ \longrightarrow \ s' \ , \ \langle C_2, s'\rangle \ \longrightarrow \ s''}{\langle C_1; C_2, s\rangle \ \longrightarrow \ s''} \\
\mathtt{if}^{v}{}_{sn} & \dfrac{\langle C_1, s\rangle \ \longrightarrow \ s'}{\langle \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2, s\rangle \ \longrightarrow \ s'} \text{ se } \mathcal{B}[\![B]\!]s = \mathsf{true} \\
\mathtt{if}^{f}{}_{sn} & \dfrac{\langle C_2, s\rangle \ \longrightarrow \ s'}{\langle \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2, s\rangle \ \longrightarrow \ s'} \text{ se } \mathcal{B}[\![B]\!]s = \mathsf{false} \\
\mathtt{while}^{v}{}_{sn} & \dfrac{\langle C, s\rangle \ \longrightarrow \ s', \ \langle \mathtt{while}\ B\ \mathtt{do}\ C, s'\rangle \ \longrightarrow \ s''}{\langle \mathtt{while}\ B\ \mathtt{do}\ C, s\rangle \ \longrightarrow \ s''} \text{ se } \mathcal{B}[\![B]\!]s = \mathsf{true} \\
\mathtt{while}^{f}{}_{sn} & \langle \mathtt{while}\ B\ \mathtt{do}\ C, s\rangle \ \longrightarrow \ s \text{ se } \mathcal{B}[\![B]\!]s = \mathsf{false}
\end{array}
$$

Exemplo

Sendo $s_0 = [x = 5, y = 7]$ determinar o estado após a execução de:

$$(z \leftarrow x; x \leftarrow y); y \leftarrow z.$$

Para tal constrói-se uma **Árvore de derivação** com esse comando como raiz:

$$\frac{\dfrac{\langle z \leftarrow x, s_0\rangle \longrightarrow s_1 \quad \langle x \leftarrow y, s_1\rangle \longrightarrow s_2}{\langle z \leftarrow x; x \leftarrow y, s_0\rangle \longrightarrow s_2} \quad \langle y \leftarrow z, s_2\rangle \longrightarrow s_3}{\langle (z \leftarrow x; x \leftarrow y); y \leftarrow z, s_0\rangle \longrightarrow s_3}$$

onde,

$$\begin{aligned} s_1 &= s_0[5/z] \\ s_2 &= s_1[7/x] \\ s_3 &= s_2[5/y] \end{aligned}$$

Theorem 1 (Integridade)**.** *Para todas as asserções de correcção parcial* $\{\varphi\}C\{\psi\}$*,*

$$\vdash_p \{\varphi\}C\{\psi\} \textit{ implica } \models_p \{\varphi\}C\{\psi\}$$

A demonstração é por indução na árvore de inferência de $\vdash_p \{\varphi\}C\{\psi\}$:

- Mostrar que a propriedade se verifica para as árvores simples i.e os **axiomas** do sistema de inferência.
- Mostrar que a propriedade se verifica para as árvores de inferência compostas: para cada regra, supor que a propriedade se verifica para as premissas (e as condições se verificam) e mostrar que a propriedade também se verifica para a conclusão da regra.

Caso ass_p**.** Suponhamos que $\vdash_p \{\varphi[E/x]\}x \leftarrow E\{\varphi\}$.

Seja

$$\langle x \leftarrow E, s\rangle \longrightarrow s'$$

e $s \models \varphi[E/x]$ se e só se $s[\mathcal{A}[\![E]\!]s/x] \models \varphi$. (Exercício)

Temos que provar que $s' \models \varphi$.

Por $[ass_{sn}]$ temos que $s' = s[\mathcal{A}[\![E]\!]s/x]$, e portanto

$$s' \models \varphi \text{ sse } s[\mathcal{A}[\![E]\!]s/x] \models \varphi$$

Caso $comp_p$**.** Supomos que $\vdash_p \{\varphi\}\, C_1\, \{\eta\}$ e $\vdash_p \{\eta\}\, C_2\, \{\psi\}$. Por hip. de indução $\models_p \{\varphi\} C_1 \{\eta\}$ e $\models_p \{\eta\} C_2 \{\psi\}$.

Queremos mostrar que

$$\models_p \{\varphi\} C_1; C_2 \{\psi\}.$$

Sejam s e s'' estados, tal que $s \models \varphi$ e $\langle C_1; C_2, s\rangle \longrightarrow s''$. Pela regra $[comp_{sn}]$ existe s' tal que

$$\langle C_1, s\rangle \longrightarrow s' \text{ e } \langle C_2, s'\rangle \longrightarrow s''$$

De $\langle C_1, s\rangle \longrightarrow s'$, $s \models \varphi$ e $\models_p \{\varphi\} C_1 \{\eta\}$, temos que $s' \models \eta$.

De $\langle C_2, s'\rangle \longrightarrow s''$, $s' \models \eta$ e $\models_p \{\eta\} C_2 \{\psi\}$, temos que $s'' \models \psi$. Que é o que queríamos.

Caso if_p**.** Supomos que $\vdash_p \{B \wedge \varphi\}\, C_1\, \{\psi\}$ e $\vdash_p \{\neg B \wedge \varphi\}\, C_2\, \{\psi\}$. Por hip. de indução $\models_p \{B \wedge \varphi\} C_1 \{\psi\}$ e $\models_p \{\neg B \wedge \varphi\} C_2 \{\psi\}$.

Para provar que

$$\models_p \{\varphi\}\, \texttt{if}\, B\, \texttt{then}\, C_1\, \texttt{else}\, C_2\, \{\psi\}$$

sejam s e s' estados tais que $s \models \varphi$ e

$$\langle \texttt{if}\, B\, \texttt{then}\, C_1\, \texttt{else}\, C_2, s\rangle \longrightarrow s'.$$

Se $\mathcal{B}[\![B]\!]s = \mathsf{true}$ então por $[if_{sn}]$, temos que $\langle C_1, s\rangle \longrightarrow s'$.

Então dado que

$$\models_p \{B \wedge \varphi\} C_1 \{\psi\}.$$

concluímos que $s' \models \psi$.

Analogamente se conclui, caso $\mathcal{B}[\![B]\!]s = \mathsf{false}$.

Caso $while_p$**.** Supomos que $\vdash_p \{B \wedge \varphi\}\, C\, \{\varphi\}$. Por hip. de indução

$$\models_p \{B \wedge \varphi\} C \{\varphi\}. \tag{1}$$

Para provar que

$$\models_p \{\varphi\}\, \texttt{while}\, B\, \texttt{do}\, C\, \{\neg B \wedge \varphi\},$$

sejam s e s'' estados tais que $s \models \varphi$ e

$$\langle \texttt{while}\, B\, \texttt{do}\, C, s\rangle \longrightarrow s''.$$

Temos que mostrar que $s'' \models \neg B \wedge \varphi$. Usamos indução na árvore de derivação da semântica natural.

Caso $while_p$**.** Há dois casos a considerar, consoante $[while_{sn}]$.

Se $\mathcal{B}[\![B]\!]s = \mathsf{false}$ então $s'' = s$ e $s'' \models (\neg B \ \wedge \ \varphi)$.

Senão, $\mathcal{B}[\![b]\!]s = \mathsf{true}$ e existe s' tal que $\langle C, s\rangle \ \longrightarrow \ s'$ e $\langle \texttt{while}\, B\, \texttt{do}\, C, s'\rangle \ \longrightarrow \ s''$.

Temos que $s \models (B \ \wedge \ \varphi)$ e pela hipótese (1) temos que $s' \models \varphi$.

Aplicando a hipótese de indução a

$$\langle \texttt{while}\, B\, \texttt{do}\, C, s'\rangle \ \longrightarrow \ s'',$$

temos que

$$s'' \models (\neg B \ \wedge \ \varphi),$$

como queríamos.

Caso $cons_p$**.** Por hip. de indução

$$\models_p \{\varphi'\}C\{\psi'\}, \ \models \varphi \to \varphi', \text{ e } \models \psi' \to \psi. \tag{2}$$

Para provar que

$$\models_p \{\varphi\}C\{\psi\},$$

sejam s e s' tal que $s \models \varphi$ e $\langle C, s\rangle \ \longrightarrow \ s'$.

Como $s \models \varphi$ e $\varphi \to \varphi'$ então $s \models \varphi'$ e pela hipótese (2), $s' \models \psi'$.

Mas como $s' \models \psi' \to \psi$, temos que $s' \models \psi$, como queríamos.

Completude da semântica axiomática

Theorem 2 (Incompletude de Gödel (1931))**.** *Não existe um sistema de demonstração para* ***PA*** *(aritmética), de tal forma que os teoremas coincidam com as asserções válidas de* ***PA****.*

Theorem 3 (Completude)**.** *Para todas as asserções de correcção parcial* $\{\varphi\}C\{\psi\}$*,*

$$\models_p \{\varphi\}C\{\psi\} \textit{ implica } \vdash_p \{\varphi\}C\{\psi\}$$

Note-se que $\models \psi$, se e só se $\models \{\texttt{true}\}\texttt{skip}\{\psi\}$. O que significa que a completude de $\vdash_p$ contraria o teorema de incompletude de Gödel.

Theorem 4. *Não existe um sistema de demonstração para asserções de correcção parcial, de tal forma que os teoremas coincidam com as asserções de correcção parcial válidas.*

Prova: Note-se que

$$\models \{\texttt{true}\}C\{\texttt{false}\}$$

se e só se o comando C diverge em todos os estados (não termina).

Um sistema de demonstração para asserções de correcção parcial, poderia ser usado para confirmar que o comando diverge em todos os estados. O que é impossível (*Halting Problem*).

Completude relativa

Theorem 5. *O sistema de prova para correcção parcial é relativamente completo, i.e. para qualquer asserção de correcção parcial* $\{\varphi\}C\{\psi\}$*:*

$$\vdash_p \{\varphi\}C\{\psi\} \textit{ se } \models_p \{\varphi\}C\{\psi\}$$

O resultado de correcção parcial relativa foi estabelecido por Stephen Cook (1978).

O facto de $\vdash_p \{\varphi\}C\{\psi\}$ ser uma prova depende do facto de certas proposisões em **PA** serem válidas (em especial as usadas na regra $cons_p$).

Para a demonstração de completude relativa ver Capítulo 7 [Win93]

Outros comandos: Ciclos `for`

Suponhamos que a linguagem imperativa continha um comando `for`

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

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:

$$x \leftarrow e_1; C; x \leftarrow e_1 + 1; C \ldots; x \leftarrow e_2; C$$

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

`for`

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

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

Mas isto não chega porque:

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

Lógica de Hoare

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

$$\{\varphi \ \wedge \ E_2 < E_1\}\, \texttt{for}\, x \leftarrow E_1\, \texttt{until}\, E_2\, \texttt{do} C\, \{\varphi\}$$

[for_p]

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

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

Exemplo

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

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

Arrays (*aliases*)

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

$$\{\varphi[E_2/u[E_1]]\}u[E_1] \leftarrow E_2\{\varphi\}$$

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

Por exemplo, $u[i] \leftarrow 10$ com a pré-condição $\{a[j] > 100\}$ e $i = j$.

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

$$u \leftarrow u[E_1 \triangleright E_2]$$

que significa que u 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 $u[i]$ e de $u[j]$ mudam os dois, porque muda o próprio array.

Sintaxe da linguagem While$^{\texttt{array}}$

Para $n \in \mathbf{Num}$, $x \in \mathbf{Var}$, $u \in \mathbf{Array}$

$$\begin{array}{rrl}
\textbf{ArrayExp} & A ::= & u \mid A[E \triangleright E] \\
\\
\textbf{AExp} & E ::= & n \mid x \mid -E \mid E + E \mid E - E \\
& & \mid E \times E \mid E \div E \\
& & \mid A[E] \\
\\
\textbf{BExp} & B ::= & \texttt{true} \mid \texttt{false} \mid \neg B \mid E = E \\
& & \mid B < E \mid B \leq B \mid B \ \wedge \ B \mid B \ \vee \ B
\end{array}$$

Semântica das expressões de While$^{\texttt{array}}$

Apenas temos que definir a semântica de **ArrayExp**. Um array é uma função $\mathbb{Z} \to \mathbb{Z}$ então

$$\mathbf{State} = \mathbf{Var} \to \mathbb{Z} \cup \mathbf{Array} \to (\mathbb{Z} \to \mathbb{Z})$$

$$\begin{array}{rcl}
\mathcal{A}[\![u]\!]s & = & s(u) \\
\mathcal{A}[\![A[E \triangleright E']]\!]s & = & \mathcal{A}[\![A]\!]s[\mathcal{A}[\![E']\!]s/\mathcal{A}[\![E]\!]s] \\
\mathcal{A}[\![A[E]]\!]s & = & \mathcal{A}[\![A]\!]s(\mathcal{A}[\![E]\!]s)
\end{array}$$

Partial Correctness for Arrays

$[array_p(assign)\ ]$

$$\{\psi[u[E_1 \triangleright E_2]/u]\}\, u[E_1] \leftarrow E_2\, \{\psi\}$$

onde E_1 é um inteiro.

E onde

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

Exemplo

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

O tableaux fica:

$$\begin{array}{l} \{a[x] = x \ \wedge\ a[y] = y\} \\ \{a[x \triangleright a[y]][y \triangleright a[x]][x] = y \ \wedge\ a[x] = x\} \\ r \leftarrow a[x]; \\ \{a[x \triangleright a[y]][y \triangleright r][x] = y \ \wedge\ r = x\} \\ a[x] \leftarrow a[y]; \\ \{a[y \triangleright r][x] = y \ \wedge\ r = x\} \\ \{a[y \triangleright r][x] = y \ \wedge\ a[y \triangleright r] = x\} \\ a[y] \leftarrow r \\ \{a[x] = y \ \wedge\ a[y] = x\} \end{array}$$

Onde $a[x \triangleright a[y]][y \triangleright a[x]][x] = a[y]$.

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:

$$y \leftarrow 1; z \leftarrow 0; \texttt{while}\ z \neq x\ do\ (z \leftarrow z+1; y \leftarrow y \times z)$$

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.

Tableaux-$while_{tot}$

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

Exemplo

$\vdash_{tot} \{x \geq 0\} y \leftarrow 1; z \leftarrow 0; \texttt{while}\ z \neq x\ do\ (z \leftarrow z+1; y \leftarrow y \times z) \{y = x!\}$

$\{x \geq 0\}$

$\{1 = 0! \wedge 0 \leq x - 0\}$

$\texttt{y} \leftarrow \texttt{1}$

$\{y = 0! \wedge 0 \leq x - 0\}$

$\texttt{z} \leftarrow \texttt{0}$

$\{y = z! \wedge 0 \leq x - z\}$ ass_{tot}

$\texttt{while}\ z \neq x\ do$

$\{$

$\{y = z! \wedge z \neq x \wedge 0 \leq x - z \wedge x - z = e_0\}$ $cons_{tot}$

$\{y \times (z+1) = (z+1)! \wedge 0 \leq x - (z+1) \wedge x - (z+1) < e_0\}$ ass_{tot}

$\texttt{z} \leftarrow \texttt{z} + \texttt{1}$

$\{y \times z = z! \wedge 0 \leq x - z \wedge x - z < e_0\}$ ass_{tot}

$\texttt{y} \leftarrow \texttt{y} \times \texttt{z}$

$\{y = z! \wedge 0 \leq x - z \wedge x - z < e_0\}$

$\}$

$\{y = z! \wedge z = x\}$

$\{y = x!\}$

Como determinar um variante?

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

Considera esta asserção

$\vdash_{tot}$

Require: $\{x > 0\}$

  $c \leftarrow x$;

  while $c \neq 1$ **do**

    if $c\%2 == 0$ **then**

      $c \leftarrow c/2$

    else

      $c \leftarrow 3 * c + 1$

Ensure: $\{\mathsf{true}\}$

Será que este triplo é válido? Neste caso este triplo só estabelecia a terminação do programa. Mas não se sabe se termina ou não! (Collatz conjecture).

Exerc. 2.1. *Mostra que*

$$\begin{aligned} &\vdash_{tot} \{y > 0\} \\ &\texttt{while}\, y \leq r\, \texttt{do} \\ &\quad r \leftarrow r - y; \\ &\quad q \leftarrow q + 1 \\ &\{\mathsf{true}\} \end{aligned}$$

$\diamond$

1 Bibliografia

Referências

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. CUP, 2004.

[NN07] H. Nielson and F. Nielson. *Semantics with Applications: an appetizer*. Springer, 2007.

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.