Partial correctness $\mathcal{H}$

[$skip_p$]

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

[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]

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

[$while_p$]

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

[$cons_p$]

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

Soundness and Completeness

Recall that $\models_{par} \{\varphi\}C\{\psi\}$ means that for all states that satisfy φ, the state that results from the execution of C satisfies ψ, if C terminates.

- **Soundness:** Each rule must preserve the validity.

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

- **Completeness:** The system should infer all the valid partial correctness assertions.

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

Execution State

For the evaluation of an expression we need the values of the variables. A state s is a function that assigns a value to a variable

The set of states is

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

and $s \in \textbf{State}$ such that $s : \textbf{Var} \to \mathbb{Z}$.

Let $s\ x$ or $s(x)$ be the value of x in the state s. If $v \in \mathbb{Z}$,

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

Semantics of expressions

Aexp - Arithmetic expressions

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

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

Bexp - Boolean Expressions

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

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

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

Natural semantics (*big-step*)

Describes the complete execution of a command.

Configurations: $\langle C, s\rangle$ or s, where C is a command and s a state $\Gamma = (\mathbf{Com} \times \mathbf{State}) \cup \mathbf{State}$

Final configurations: $s \in \mathit{State}$

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

Rules:

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

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

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

If $n = 0$ the rule is an **Axiom**.

Natural semantics for commands While

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

Example

If $s_0 = [x = 5, y = 7]$ compute the state after the execution of:

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

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

where,

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

Theorem 1 (Soundness)**.** *For all* $\{\varphi\}C\{\psi\}$,

$$\vdash_p \{\varphi\}C\{\psi\} \text{ implies } \models_p \{\varphi\}C\{\psi\}$$

The proof is by induction in the size of the inference tree of $\vdash_p \{\varphi\}C\{\psi\}$:

- Show that the property holds for the axioms.
- Show that the property holds for compound trees: for each rule, assume that the property holds for the premises and show that the property holds for the conclusion.

Case ass_p**.** Assume that $\vdash_p \{\varphi[E/x]\}x \leftarrow E\{\varphi\}$.

Let

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

and $s \models \varphi[E/x]$ iff $s[\mathcal{A}[\![E]\!]s/x] \models \varphi$. (Exercise)

We need to prove that $s' \models \varphi$.

By $[ass_{sn}]$ we have $s' = s[\mathcal{A}[\![E]\!]s/x]$, and thus

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

Case $comp_p$**.** Assume that $\vdash_p \{\varphi\}\, C_1\, \{\eta\}$ and $\vdash_p \{\eta\}\, C_2\, \{\psi\}$. By the ind. hyp. $\models_p \{\varphi\}C_1\{\eta\}$ and $\models_p \{\eta\}C_2\{\psi\}$.

We want

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

Let s and s'' be states, such that $s \models \varphi$ and $\langle C_1; C_2, s\rangle \;\longrightarrow\; s''$. By $[comp_{sn}]$ there exists s' such that

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

From $\langle C_1, s\rangle \longrightarrow s'$, $s \models \varphi$ and $\models_p \{\varphi\}C_1\{\eta\}$, we have that $s' \models \eta$.

From $\langle C_2, s'\rangle \longrightarrow s''$, $s' \models \eta$ and $\models_p \{\eta\}C_2\{\psi\}$, we have $s'' \models \psi$. As we wanted.

Case if_p**.** Assume that $\vdash_p \{B \wedge \varphi\}\, C_1\, \{\psi\}$ and $\vdash_p \{\neg B \wedge \varphi\}\, C_2\, \{\psi\}$. By the ind. hyp. $\models_p \{B \wedge \varphi\}C_1\{\psi\}$ and $\models_p \{\neg B \wedge \varphi\}C_2\{\psi\}$.

To prove that

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

let s and s' be states such that $s \models \varphi$ and

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

If $\mathcal{B}[\![B]\!]s = \textsf{true}$ by $[if_{sn}]$, we have that $\langle C_1, s\rangle \longrightarrow s'$.

Given that

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

we conclude that $s' \models \psi$.

In the same way, we prove for $\mathcal{B}[\![B]\!]s = \textsf{false}$.

Caso $while_p$**.** Assume that $\vdash_p \{B \wedge \varphi\}\, C\, \{\varphi\}$. By induction

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

To prove that

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

let s and s'' be states such that $s \models \varphi$ and

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

We need to prove $s'' \models \neg B \wedge \varphi$. We use induction on the derivation tree of the natural semantics

Case $while_p$**.** There two cases, for $[while_{sn}]$.

If $\mathcal{B}[\![B]\!]s = \textsf{false}$ then $s'' = s$ and $s'' \models (\neg B \wedge \varphi)$.

If not, $\mathcal{B}[\![b]\!]s = \texttt{true}$ and there exists s' such that $\langle C, s\rangle \longrightarrow s'$ and $\langle \texttt{while}\, B\, \texttt{do}\, C, s'\rangle \longrightarrow s''$.

We have $s \models (B \wedge \varphi)$ and by (1) we have $s' \models \varphi$.

Applying the ind. hyp. to

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

we have

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

as wanted.

Case $cons_p$**.** Suppose that

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

To prove

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

let s and s' such that $s \models \varphi$ and $\langle C, s\rangle \longrightarrow s'$.

As $s \models \varphi$ and $\varphi \to \varphi'$ then $s \models \varphi'$ and by (2), $s' \models \psi'$.
But $s' \models \psi' \to \psi$, we have $s' \models \psi$, as wanted.

Completeness of axiomatic semantics

Theorem 2 (Incompleteness of Gödel (1931))**.** *There is no deductive system for **PA** (arithmetics), in such a way that the theorems are the valid formulae of**PA**.*

Theorem 3 (Completeness)**.** *For all partial correctness assertions* $\{\varphi\}C\{\psi\}$*,*

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

Note that $\models \psi$, iff $\models \{\texttt{true}\}\texttt{skip}\{\psi\}$. This means that the completeness of $\vdash_p$ contradicts the Incompleteness theorem of Gödel.

Theorem 4. *There is no deductive system for partial correctness assertions such that the theorems coincide with the valid partial correctness assertions.*

Proof Note that

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

iff the command C does not terminate for all states (diverge).

A deductive system could be used to assert that a command diverge which is impossible by the undecidability of the (*Halting Problem*).

Relative completeness

Theorem 5. *The proof system of partial correctness is relatively complete, i.e. for any partial correctness assertion* $\{\varphi\}C\{\psi\}$*:*

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

This reult is due to Stephen Cook (1978).

The fact that $\vdash_p \{\varphi\}C\{\psi\}$ depends on some propositions in **PA** be valid..

See Chap. 7 [Win93]

Cycle `for`

We can add to the language the command `for`

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

the meaning is:

- The expressions E_1 and E_2 are evaluated at the beginning, and let e_1 and e_2 be their values;
- If $e_1 > e_2$ do nothing;
- If $e_1 <= e_2$ the command `for` is equivalent to:

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

  The cycle executes $(e_2 - e_1) + 1$ times.

`for`

One could have the rule `for` :

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

But it is not enough:

- The command C can modify the value of x;
- The value of E_1 can be greater then the value of E_2.

Lógica de Hoare

[for_p**-axiom**] If $E_1 > E_2$

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

[for_p]

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

where neither x, or any variable that occurs in E_1 or E_2 is modified by the command C.

Example

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

Consider φ equal to $x = (n-1) \times n\,div\,2$.

Arrays (*aliases*)

If we have an array $u[]$ the assignment rule cannot be directly applied:

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

as modifications in $a[E_1]$can (should) change other references to (aliases) u that can occur in φ or in E_2.

For instance, $u[i] \leftarrow 10$ with pre-condition $\{a[j] > 100\}$ and $i = j$.

T. Hoare solution was to consider the *arrays* monolitic, and an assignment

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

means that u is a new *array* equal to the previous one where the position E_1 has value E_2.

Thus in the example the values of $u[i]$ and of $u[j]$ both change because the array itself has changed.

Syntax of the language While$^{\mathsf{array}}$

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

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

Semantics for expressions of While$^{\mathsf{array}}$

We only need to define the semântica for expressions **ArrayExp**. An array is a function $\mathbb{Z} \to \mathbb{Z}$ thus

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

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

Partial Correctness for Arrays

$[array_p(assign)\ ]$

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

where E_1 is a positive integer.

And

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

Example

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

The tableaux is

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

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

Note: In implementations this technique is not used as it is very inefficient

Calculus for total correctness

In the language **while** the only command that can lead to non termination is the command `while`.

The calculus $\vdash_{tot}$ coincides with $\vdash_p$ except in the rule $\texttt{while}_{tot}$.

To prove that a program terminates we need to associate a strictly decreasing expression called the *variant*.

For the `while` we associate a non negative expression and in each iteration we show that its value diminish maintaining non negative:in this way we ensure that in a finite number of times it will be zero.

For the factorial

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

the variant is $x - z$.

Calculus for total correctness

Hoare logic

The rules ass_{tot}, $comp_{tot}$, if_{tot} e $cons_{tot}$ are same as for $\vdash_p$

$[while_{tot}\ ]$

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

where e_0 is a logic variable whose value is the value of E before the execution of the command C.

Tableaux-$while_{tot}$

$\{\varphi\}$

$\{\eta \wedge 0 \leq E\}$

`while` B `do`

$\{\eta \wedge B \wedge 0 \leq E \wedge E = e_0\}$

C

$\{\eta \wedge 0 \leq E \wedge E < e_0\}$

$\{\eta \wedge \neg B\}$ $while_{tot}$

$\{\psi\}$ $cons_{tot}$

Example

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

$\{x \geq 0\}$

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

`y` $\leftarrow$ `1`

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

`z` $\leftarrow$ `0`

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

`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}

`z` $\leftarrow$ `z` $+$ `1`

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

`y` $\leftarrow$ `y` $\times$ `z`

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

}

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

$\{y = x!\}$

How to find a variant?

Variants are harder to find as it is not possible to know, in general, that a program terminates.

Consider this assertion

$\vdash_{tot}$

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

Is this triple valid? In this case the assertion would only ensure termination. But we do not know if the program terminates! (Collatz conjecture).

Exerc. 2.1. *Show*

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

$\diamond$

1 Bibliografia

References

[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.